Étude des performances de méthodes de groupement dynamiques dans les bases de données orientées objet

Jérôme Darmont

N° d'Ordre : D.U. 1100
EDSPIC : 190

UNIVERSITÉ BLAISE PASCAL
ÉCOLE DOCTORALE SCIENCES POUR L'INGÉNIEUR DE CLERMONT-FERRAND

THÈSE

présentée par

Jérôme DARMONT
Ingénieur en Génie Informatique de l'ISI-CUST

pour obtenir le grade de

DOCTEUR D'UNIVERSITÉ

Spécialité : INFORMATIQUE

Étude des performances de méthodes de groupement dynamiques dans les bases de données orientées objet

Soutenue publiquement le 18 janvier 1999 devant le jury :

M^{me} Véronique BENZAKEN	**Rapporteur**
M. Henri BRIAND	**Rapporteur**
M. Michel SCHNEIDER	**Directeur de Thèse**
M. Lotfi LAKHAL	**Examinateur**
M. Jean-Marc PETIT	**Examinateur**
M. Alain QUILLIOT	**Examinateur**

« I don’t fear computers. I fear the lack of them. »
Isaac Asimov

Remerciements

J’exprime tout d’abord mes plus vifs remerciements à mon directeur de thèse Monsieur Michel Schneider, dont les conseils judicieux m’ont été précieux tout au long de mon doctorat. Qu’il trouve ici l’expression de ma profonde reconnaissance.

Je tiens également à exprimer ma gratitude envers Madame Véronique Benzaken et Monsieur Henri Briand, qui m’ont fait l’honneur de porter intérêt à mon travail et d’en être les rapporteurs. Je remercie de même les autres membres qui ont composé mon jury : Messieurs Lotfi Lakhal, Jean-Marc Petit et Alain Quilliot.

Je dois aussi un grand merci à Madame Le Gruenwald, de l’Université d’Oklahoma, sans qui ce travail n’aurait sans doute pas vu le jour.

Mes remerciements et ma reconnaissance vont également aux enseignants en informatique à l’Université Blaise Pascal, où j’ai eu la chance d’exercer les fonctions de vacataire, puis de moniteur, durant trois ans. J’ai énormément apprécié cette expérience. À nouveau un grand merci à Monsieur Lotfi Lakhal pour ses conseils amicaux et pour la confiance qu’il m’a d’emblée témoignée. Je tiens également à remercier Madame Jeanne-Marie Gourgand, pour sa disponibilité dans son rôle de tuteur.

Des pensées particulières vont à tous ceux que j’ai côtoyés au quotidien pendant ces années de thèse et qui ont entretenu une ambiance toujours conviviale : Alain, Amina, Bertrand, les deux Christophe, les deux Corinne, David, Hal9000, Jean-Yves, Kitsana, Laurent, Nathalie, Nicolas, Nicolay, Pat, les deux Philippe, Quentin, Rafik, Samah, Sébastien, Sereg, Stid, Yahia et Madame Seguy. J’espère n’oublier personne !

Enfin, je tiens à remercier sincèrement tous ceux qui m’ont soutenu et encouragé, même sans le savoir, en dehors de l’université et du laboratoire, et en premier lieu mon épouse Anne-Gaëlle, mes parents, mes grands-parents, mes beaux-parents et mon beauf’ adoré Michaël. Un immense merci aussi à la bande de La Chapelle (David, Caroline et Paul ; Philippe, Francette et Vincent ; Fred et Aurélie ; et nos chevaux sans bride Laurent et Stéphane), à Didier, Gillou, Miguel, et Arnaud. Et quelques gratouilles sur le ventre de Cendre pour avoir déterré toutes les plantes du salon…

Table des matières

Liste des figures

Liste des tables

Introduction

Ce chapitre d'introduction est en premier lieu destiné à définir notre domaine d'intervention : l'évaluation de performance des Systèmes de Gestion de Bases de Données Orientés Objet (SGBDOO) et plus particulièrement l'évaluation de l'efficacité de méthodes de groupement d'objets, qui permettent d'optimiser notablement les performances des SGBDOO. Nous discutons tout d'abord de la nécessité d'améliorer ces performances, puis indiquons les techniques principalement mises en œuvre pour y parvenir. Nous présentons ensuite les moyens actuellement disponibles pour évaluer les performances des SGBDOO et l'impact que peuvent avoir diverses méthodes d'optimisation sur celles-ci. Nous montrons les enjeux ainsi que les problèmes existant dans ce domaine et les objectifs que nous nous sommes fixés pour y répondre. Nous exposons finalement en détail l'organisation de ce mémoire.

1. Problèmes de performances des SGBDOO

Les Systèmes de Gestion de Bases de Données Orientés Objets sont apparus à la fin des années 80. L'idée sous-jacente à leur conception était d'allier les concepts de l'approche orientée objet aux aspects fondamentaux des SGBD : sécurité, cohérence et protection des données, souplesse et puissance de consultation, mécanismes de reprise en cas de panne, etc.

L'approche objet est, elle, apparue dans les années 60 avec le langage Simula [Dahl et Nygaard 66], puis s'est considérablement développée à partir de 1980 avec l'apparition de langages comme Smalltalk-80 [Goldberg et Robson 81] ou C++ [Stroustrup 97]. Elle est en passe de devenir un standard en programmation, en raison, entre autres, de sa grande capacité d'abstraction et de sa puissance de modélisation des données complexes.

C'est justement de telles caractéristiques dont ont besoin de « nouvelles » applications de bases de données, relativement éloignées des applications de gestion traditionnelles. Par exemple, les Ateliers de Génie Logiciel (AGL) doivent permettre la conception, la spécification, l'analyse, le débogage, la maintenance et l'évolution de programmes et de leur documentation. Les applications de Conception Assistée par Ordinateur (CAO) ou de Publication Assistée par Ordinateur (PAO) à base d'hypertextes utilisent des objets complexes à comportement dynamique, pouvant exister en plusieurs versions et eux-mêmes constitués de nombreux autres objets présentant ces mêmes caractéristiques. Les

applications multimédia doivent également gérer des données (photographies, sons, films, documents textuels…) volumineuses, de taille variable et, surtout, de structures variées et complexes.

Or, le modèle relationnel, qui connaît toujours un vif succès de part sa simplicité et de part l'efficacité des SGBD qui le mettent en œuvre, s'est avéré insuffisant dans ces nouveaux domaines. En effet, une relation ne permet de représenter des données que sous forme « plate », si bien qu'un effort d'analyse est requis pour modéliser des hiérarchies. Pour ce faire, il est nécessaire de décomposer de façon simple, en tables, des données dont la structure est en réalité complexe. Cet éclatement d'une entité en plusieurs tables diminue sa lisibilité et sa compréhension. De plus, les types de base disponibles dans le modèle relationnel standard (les domaines) sont trop simples et ne permettent pas le stockage aisé de sons, de graphismes ou de séquences vidéo, en tant qu'attributs d'une table. Il n'est également pas possible de prendre en compte des types nouveaux, définis par les utilisateurs. Les SGBDOO ont pour objectif de répondre à ces besoins et de combler ces lacunes.

Cependant, les SGBDOO, qui sont pourtant le fruit de nombreux travaux de recherche, ne bénéficient pas d'acquis technologiques mûrs et éprouvés. Leur construction a été difficile et demeure délicate. De nombreux problèmes technologiques ont dû être résolus, mais certains d'entre eux n'ont toujours pas trouvé de solution réellement satisfaisante. D'ailleurs, la réussite des SGBDOO apparus au début des années 90 a souvent été très difficile sur le plan commercial [Delobel et al. 91]. En effet, les applications multimédia ou de CAO, par exemple, gèrent des volumes de données importants, ce qui s'est avéré un obstacle difficile à surmonter pour des systèmes encore immatures à leur entrée sur le marché [Hurson et al. 93]. Pour finalement s'imposer dans les domaines qui leur sont adaptés, les SGBDOO doivent donc encore, comme l'ont fait les systèmes relationnels vingt ans plus tôt en matière de gestion, définir un standard, asseoir la pertinence de leurs modèles et, surtout, proposer des outils performants. En effet, les apports sémantiques du modèle objet ne peuvent seuls compenser des performances insuffisantes.

2. Optimisation des performances des SGBDOO

L'optimisation des performances est un objectif clé pour tous les SGBD. Il s'avère primordial en ce qui concerne les SGBDOO. L'objectif est d'améliorer les performances des systèmes en termes de débit (nombre de transactions types à la seconde) et de temps de réponse (temps d'attente moyen pour une requête type). Il s'agit, d'une part, de minimiser le temps passé à l'exécution des tâches multiples accomplies par le système et, d'autre part, d'optimiser la gestion des accès concurrents aux données dans le cas de systèmes multi-utilisateurs, afin que les transactions ne passent pas plus de temps à attendre qu'à s'exécuter.

Par ailleurs, les performances des SGBD dépendent directement de l'usage qui est fait de la mémoire secondaire. En effet, chaque entrée-sortie (E/S) sur disque nécessite jusqu'à une dizaine de milli-secondes. L'accès à la mémoire secondaire constitue donc de fait un véritable goulot d'étranglement. De plus, au coût directement engendré par l'exécution des entrées-sorties physiques s'ajoutent des coûts induits proportionnels au nombre de pages disques requises par les applications :

- place mémoire occupée dans le cache ;
- nombre de verrous gérés par le contrôle de concurrence ;
- nombre d'écritures dans les journaux pour la reprise sur panne ;
- nombre d'interactions entre le client et le serveur ;
- nombre de conflits transactionnels.

La diminution du nombre d'entrées-sorties permet donc une réduction de l'ensemble de ces coûts induits, qui sont loin d'être négligeables, en particulier dans le cas d'applications réparties.

Les travaux concernant l'optimisation des performances des SGBDOO sont principalement basés sur des techniques éprouvées, héritées des SGBD relationnels ou des systèmes d'exploitation. Cependant, des adaptations parfois importantes sont nécessaires afin d'exploiter au mieux les particularités sémantiques des bases de données orientées objet. Les quelques éléments suivants influencent notablement les performances des SGBDOO (nous y reviendrons plus en détail dans le Chapitre 2) :

- le groupement des objets sur le disque (*clustering*) ;
- la gestion des caches mémoire (*buffering*) ;
- les stratégies de gestion de la concurrence ;
- l'optimisation des requêtes ;
- les méthodes d'accès aux objets (codage des identificateurs d'objets, index, hachage…).

De nombreuses heuristiques ont été proposées pour prendre ces éléments en compte et améliorer les performances globales des SGBDOO. Ce sont les techniques de regroupement d'objets qui nous concernent plus particulièrement pour ce travail.

3. Évaluation des performances des SGBDOO

En raison des spécificités du modèle objet, les stratégies d'optimisation des performances pour SGBDOO sont souvent complexes. De plus, pour un problème d'optimisation donné, il existe plusieurs solutions. Différentes stratégies peuvent donc voir le jour, donnant lieu à différentes heuristiques.

Il apparaît indispensable d'étudier l'impact réel de ces heuristiques sur les performances globales du système. En effet, si d'un côté, ces méthodes permettent une amélioration des performances globales, d'un autre, elles engendrent souvent une surcharge de traitement pour le système. Il est donc primordial d'évaluer quel est le gain effectif autorisé par ces techniques, c'est-à-dire la différence entre le gain de performance attendu et le surcoût dû à l'utilisation de la méthode. De plus, il est intéressant de pouvoir comparer entre elles différentes stratégies et pas seulement de constater qu'elles améliorent les performances individuellement. Il est ainsi possible d'apprécier quelle méthode est la meilleure dans des conditions données.

Il existe actuellement trois approches principales pour évaluer les performances des SGBDOO ou de composants particuliers dans les SGBDOO.

- Les bancs d'essais (*benchmarks*, dans la terminologie anglo-saxonne) sont couramment employés pour évaluer les performances globales des SGBDOO. Ils proposent en général une base de données type et des transactions à exécuter sur cette base. Leur principe est de mesurer directement la réponse du système.
- L'analyse mathématique est plus souvent utilisée pour définir de façon exacte la complexité ou le coût (grâce à des modèles de coût) d'algorithmes spécifiques.
- La simulation aléatoire à événements discrets est, elle, très peu utilisée dans le domaine des SGBDOO. Elle l'est principalement sous la forme de modèles de simulations dédiés à l'étude d'une heuristique particulière. Cependant, des modèles plus globaux sont parfaitement envisageables.

Ces moyens d'évaluation ne sont pas entièrement satisfaisants tels quels, pour deux raisons principales.

Premièrement, il est difficile d'évaluer les performances d'un système *a priori* (par exemple, les performances d'un prototype de recherche, en se basant seulement sur ses spécifications). L'utilisation d'un banc d'essais implique de disposer du système étudié. L'analyse mathématique possède des limites lorsqu'il faut appréhender le comportement global d'un système. En effet, ce comportement est régi par un tel nombre de paramètres qu'il est nécessaire de poser des hypothèses simplificatrices qui font diverger les résultats finaux par rapport à la réalité. La simulation aléatoire à événements discrets est une solution envisageable en matière de modélisation *a priori,* mais souffre actuellement de défauts qui nous amènent au second point : les approches pour évaluer les performances des SGBDOO sont actuellement très ciblées sur un type de système ou d'application, et non génériques.

Les modèles de simulation présentés dans la littérature sont en très grande majorité dédiés à l'étude d'une heuristique ou d'un système particuliers. L'analyse mathématique qui, comme nous l'avons déjà évoqué, a recours a des hypothèses simplificatrices lorsqu'elle est confrontée à des systèmes complexes, n'est souvent employée que pour des problèmes restreints (par exemple, un algorithme d'optimisation particulier). Pour

terminer, même les bancs d'essais dits généralistes sont clairement orientés vers les applications d'ingénierie comme la CAO ou les AGL.

C'est donc sur ces deux points : généricité et évaluation des performances *a priori*, par simulation, que portent nos recherches.

4. Objectifs et contributions

Notre objectif principal consiste à fournir des outils génériques et valides pour l'évaluation des performances des SGBDOO en général et des techniques de regroupement d'objets dans les SGBDOO en particulier. L'aspect générique de notre approche doit nous permettre d'effectuer des études de performance sur n'importe quel type de SGBDOO. En particularisant nos outils, nous pouvons traiter plus spécifiquement les problèmes de groupement d'objets qui nous intéressent. En effet, prendre en compte tous les aspects qui influencent les performances d'un SGBDOO demanderait un travail considérable et sortirait du cadre de cette thèse.

Notre travail s'articule donc en deux axes :

- la conception d'un banc d'essais générique nommé OCB (ce qui signifiait à l'origine *Object Clustering Benchmark*, bien que le banc d'essais ne soit plus dorénavant uniquement destiné à étudier des techniques de groupement d'objets). OCB, grâce à un ensemble de paramètres, est à même de générer tout type de base de données orientée objet et d'exécuter sur elle différentes classes de transactions modélisant différentes applications utilisant cette base d'objets ;
- la proposition d'une méthodologie de modélisation des SGBDOO en vue d'obtenir des modèles de simulation. Cette approche a pour objectif de systématiser l'analyse des systèmes afin d'obtenir des modèles de simulation fiables. Elle a été employée pour produire le modèle VOODB (*Virtual Object-Oriented Database*), qui permet des évaluations de performance *a priori* de SGBDOO et plus précisément d'algorithmes de groupement au sein de ces systèmes. VOODB intègre le banc d'essais OCB pour ses tests de performance.

Il est par ailleurs indispensable que ces deux outils soient valides, c'est-à-dire qu'ils fournissent des résultats fiables. Nous devons donc nous attacher à le démontrer grâce à diverses expérimentations. VOODB doit, notamment, fournir des résultats équivalents au banc d'essais OCB pour l'étude d'un système existant donné.

5. Organisation de la thèse

Ce mémoire est organisé comme suit.

Le Chapitre 1 présente quelques-unes des techniques les plus employées en matière d'optimisation des performances pour les SGBDOO. Elle ne prétend pas faire un état de l'art complet du domaine, ce qui sortirait du cadre de nos travaux. Nous nous y attachons cependant à décrire en détail quelques méthodes de regroupement d'objets représentatives de la recherche actuelle, puisque nos efforts d'évaluation de performance portent essentiellement sur ces techniques.

Le Chapitre 2 traite de l'importance d'évaluer les performances des SGBDOO, identifie et détaille les divers moyens dont nous disposons actuellement pour le faire, à savoir l'utilisation de bancs d'essais (*benchmarking*), l'analyse mathématique et la simulation aléatoire à événements discrets. Les bancs d'essais orientés objet les plus utilisés sont décrits, ainsi que plusieurs études mathématiques concernant le groupement d'objets, puis différents modèles de simulation employés pour évaluer les performances globales d'un système ou celles d'une heuristique en particulier. Une analyse critique de chacune de ces méthodes est dressée.

Le Chapitre 3 présente le premier outil que nous avons mis en œuvre pour évaluer les performances des SGBDOO en général et des méthodes de groupement d'objets en particulier : le banc d'essais OCB. Nous abordons en détail les motivations qui nous ont poussées à le concevoir, présentons l'architecture et les spécifications complètes d'OCB, ainsi qu'un bilan comparatif vis-à-vis des autres bancs d'essais existants.

Le Chapitre 4 expose point par point la méthodologie de modélisation que nous préconisons pour la production de modèles de simulation à événements discrets valides pour l'évaluation des performances des SGBDOO, sa justification, ainsi que son application pour obtenir le modèle VOODB, qui est entièrement détaillé.

Le Chapitre 5 présente en détail DESP-C++, l'outil de simulation que nous avons employé dans nos expériences, les motivations qui ont poussé à sa conception, son architecture, un guide d'utilisation et des éléments de validation.

Le Chapitre 6 décrit le SGBDOO O_2, le gestionnaire d'objets persistants Texas et la technique de regroupement DSTC, qui tiennent une place prépondérante dans les tests et les expériences de validation de nos outils d'évaluation des performances.

Le Chapitre 7 décrit finalement les expériences que nous avons menées afin de valider notre banc d'essais OCB et notre modèle de simulation VOODB. L'ensemble des résultats est présenté et interprété. Ils permettent de conclure favorablement sur de nombreux points et amènent par ailleurs diverses perspectives de travail.

La conclusion de ce mémoire porte tout d'abord sur un rappel des principales contributions de notre travail. Un bilan des résultats obtenus en rapport des objectifs fixés est dressé, indiquant les difficultés que nous avons rencontrées et les points faibles que nous attribuons à nos travaux. Finalement, nous indiquons les perspectives d'évolution de ces recherches.

Chapitre 1
Optimisation des performances des SGBDOO

Nous avons évoqué en introduction les lacunes que présentent les SGBDOO en matière de performance, notamment lorsqu'ils sont comparés sur ce point aux systèmes relationnels, en usage depuis les années 70 et parfaitement optimisés. L'objectif de ce chapitre est de présenter quelques éléments clés des SGBDOO qui influencent leurs performances. Nous ne prétendons pas établir un état de l'art sur le sujet de la gestion d'objets dans les SGBD (le lecteur intéressé pourra se référer aux articles et ouvrages suivants : [Benzaken et al. 88], [Steffen 90], [Delobel et al. 91], [Amiel et al. 92], [Chabridon et al. 92], [Bouzeghoub et al. 94]), mais insistons surtout sur les composants qui peuvent exercer une influence sur la définition et la mise en place de stratégies de groupement d'objets.

Nous présentons ensuite les principales méthodes d'optimisation des performances mises en œuvre dans les SGBDOO, en insistant particulièrement sur le groupement d'objets, pour lequel nous détaillons plusieurs techniques représentatives des recherches qui ont été effectuées sur le sujet.

1. Éléments d'un SGBDOO influençant les performances

1.1. Architecture fonctionnelle

L'architecture fonctionnelle d'un SGBD décrit le système en terme de couches logicielles comprenant essentiellement [Bouzeghoub et al. 94] :

- les outils et l'environnement de développement offerts ;
- les langages et interfaces disponibles : langages de requêtes, définition de schéma, programmation, interfaces externes (avec SQL, par exemple) ;
- le gestionnaire d'objets persistants.

Nous rappelons ci-après les fonctions communément attribuées aux gestionnaires d'objets persistants [Benzaken et al. 88].

1) *Création et manipulation d'objets persistants :* Mise en œuvre de mécanismes introduisant la persistance des objets, conformément au modèle de données supporté.
2) *Gestion de la mémoire primaire :* Mise en œuvre de mécanismes d'adressage des objets, pagination basée sur un mécanisme de mémoire virtuelle contrôlée par le gestionnaire d'objets, gestion de cache mémoire, ramasse-miettes.
3) *Gestion de la mémoire persistante :* Manipulation d'enregistrements physiques, de pages disques, de fichiers, gestion des objets longs (de taille supérieure à celle d'une page disque), maintenance de structure d'accès aux objets (index). Allocation d'espaces physiques, gestion des entrées-sorties avec cache de pages au niveau des disques.
4) *Gestion transactionnelle :* Mise en œuvre du concept de transaction (respectant les propriétés ACID[*]) avec inclusion d'un mécanisme de reprise après panne basé sur des techniques de journalisation.
5) *Maintien de l'intégrité des données :* Mise en œuvre d'un protocole de verrouillage pour le contrôle de concurrence et de mécanismes de contrôle des types et de vérification des contraintes d'intégrité sémantiques.
6) *Gestion de schéma :* Support des concepts de classes, attributs, méthodes, héritage, etc. Définition et évolution de schéma.
7) *Gestion de versions* (optionnel)

1.2. Architectures client-serveur

L'aspect opérationnel, par opposition à l'aspect fonctionnel, se rapporte à l'architecture matérielle du système et diffère essentiellement sur la mise en œuvre du modèle client-serveur. Apparu dans les années 80 au sein des SGBD relationnels, ce modèle est désormais un standard pour les SGBDOO. Les fonctionnalités offertes aux utilisateurs sont prises en charge par des processus distincts qui jouent le rôle de serveurs et de clients. Un serveur est capable de supporter plusieurs clients simultanément. Les clients sont totalement indépendants les uns des autres. Clients et serveurs sont généralement répartis sur des sites géographiquement distants, mais ce n'est pas systématique. Un serveur peut être remplacé par un autre sans impact sur les clients, notamment en cas de panne réseau.

La question essentielle posée pour les SGBDOO est la répartition des fonctionnalités entre le client et le serveur. La répartition de ces fonctionnalités (cf. Section 1.1) diffère selon l'architecture adoptée. Leur mise en œuvre et l'architecture matérielle du SGBD

[*] ACID : Atomicité – Cohérence – Isolation – Durabilité.

doivent être adaptées aux domaines d'application visés et, surtout, à leurs exigences en terme de performance. Parallèlement, la puissance sans cesse croissante des stations de travail tend à déporter de plus en plus de fonctionnalités vers les clients. Nous présentons brièvement les architectures les plus courantes dans cette section. Une étude plus approfondie est disponible dans [DeWitt et al. 90].

[Loomis 92] identifie trois types d'architectures pour les SGBD en général : l'approche *serveur d'objets*, l'approche *serveur de pages* et l'approche *serveur de bases de données*. La Table 1.1 illustre, pour chacune de ces trois architectures, la répartition des fonctionnalités entre le client et le serveur.

Architecture	Répartition des fonctionnalités
Serveur d'objets	CLIENT – SERVEUR
Serveur de pages	CLIENT – serveur
Serveur de bases de données	client – SERVEUR

Table 1.1 : Répartition du travail entre client et serveur

Dans l'approche serveur d'objets, les fonctionnalités sont réparties assez équitablement entre le client et le serveur. Cette architecture a été employée dans les premiers SGBDOO, ainsi que dans les SGBD relationnels étendus à l'objet. L'unité de transfert est l'objet (ou le n-uplet, dans le cas du relationnel étendu), parfois le groupe d'objets. Les premières versions des SGBD O_2 [Bancilhon et al. 88], ORION [Kim et al. 88] et GemStone [Copeland et Maier 84] ont mis en œuvre un serveur d'objets. ONTOS [Andrews et al. 91] et VERSANT [Houdas 93] utilisent également ce type d'architecture.

L'approche serveur de pages concentre les fonctionnalités du système sur le client. Aucune des couches hautes du SGBD ne se situe au niveau du serveur. L'unité de transfert entre le serveur et le client est la page. ObjectStore [Lamb et al. 91] et les dernières versions d'O_2 [Deux et al. 91] utilisent un serveur de page.

L'approche serveur de bases de données représente la tendance inverse, en déportant la majeure partie des fonctions bases de données sur le serveur. Le client a quasiment pour seul rôle de transmettre des requêtes aux serveurs et de mettre les résultats à disposition de l'application émettrice. Ce type d'architecture correspond à la génération de SGBD liée aux gros systèmes et aux mini-ordinateurs. Il n'est plus beaucoup utilisé de nos jours car il ne prend pas en compte la puissance des stations de travail actuelles.

Il existe également d'autres types d'architectures que nous évoquons seulement ici, comme l'approche *serveur de fichiers* [DeWitt et al. 90], l'approche *multi-serveur* implantée dans les dernières versions de GemStone [Servio Corporation 92] ou la démarche proposée par l'INRIA pour le système EOS [Gruber 92].

Nous nous intéressons ici plus particulièrement aux architectures serveur d'objets et serveur de pages qui constituent les deux alternatives les plus couramment discutées.

1.2.1. Serveurs d'objets

1.2.1.1. Architecture

La Figure 1.1 illustre la répartition des fonctionnalités du gestionnaire d'objets dans une architecture de type serveur d'objets. La gestion d'objet « pure » (persistance, adressage des objets, gestion de cache d'objets) est entièrement dupliquée sur le client et le serveur. Par contre, les fonctionnalités bases de données incombent en totalité au serveur. Ce dernier gère également un cache de pages, les entrées-sorties s'effectuant systématiquement par pages. Néanmoins, le serveur ne transmet au client que les objets demandés, après les avoir sauvés dans son cache d'objets. Parallèlement, le client ne fait appel au serveur qu'après avoir vérifié l'absence de l'objet requis dans son propre cache.

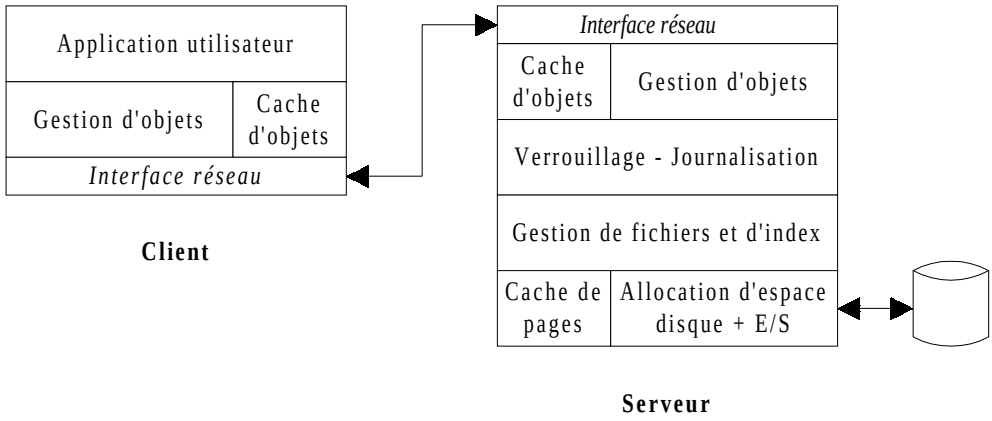

Figure 1.1 : Architecture serveur d'objets

1.2.1.2. Considérations de performance

Les principaux avantages du serveur d'objets proviennent de sa capacité à appréhender et à manipuler le concept d'objet, ce qui a plusieurs conséquences bénéfiques.

- Les méthodes des objets peuvent être exécutées sur le serveur, ce qui favorise la sélection puis le transfert des seuls objets utiles à l'application, lors d'un accès soumis à condition, par exemple. Les coûts de communication sont ainsi réduits. Les accès associatifs ou ensemblistes sont très performants sur ce type d'architecture.
- Le cache du client ne contient que des objets réellement utilisés par l'application, ce qui rentabilise l'espace mémoire occupé.
- Le verrouillage et la journalisation peuvent être effectués au niveau de l'objet. Ceci permet d'augmenter le taux de concurrence entre plusieurs utilisateurs et le nombre de clients potentiels du serveur.

Cependant, les nombreuses fonctionnalités assurées par le serveur dans ce type d'architecture font également apparaître de gros inconvénients.

- Le serveur doit coordonner deux types de travaux différents : le contrôle de concurrence et le recouvrement d'une part, et l'exécution des requêtes, la recherche et le transfert des objets (avec recopie dans le cache d'objets du serveur avant le transfert) d'autre part. Même un serveur très puissant peut rapidement devenir un goulot d'étranglement face aux nombreuses demandes de stations clientes dont la puissance cumulée dépasse celle du serveur.
- Cette architecture est beaucoup mieux adaptée aux accès associatifs (type relationnel) qu'aux accès navigationnels (parcours de hiérarchies d'objets complexes par utilisation de références inter-objets), qui sont les plus nombreux dans les SGBDOO.
- La possibilité d'exécuter une méthode sur le serveur peut s'avérer dangereuse et occasionner des problèmes ou des pannes en cas de manque de fiabilité du code utilisateur.
- La présence à la fois d'un cache d'objets et d'un cache de pages sur le serveur induit un grand nombre de copies d'objets.

1.2.2. Serveurs de pages

1.2.2.1. Architecture

La Figure 1.2 illustre l'architecture serveur de pages. La gestion d'objets est entièrement supportée par le client, de même que la gestion de fichiers. Le serveur gère simplement le verrouillage et la journalisation, les entrées-sorties effectuées sur le disque et un cache de pages. Par contre, il n'a aucune connaissance des objets contenus dans ces pages et ne fait que transmettre des pages explicitement demandées par les clients. Le client possède également un cache de pages destiné à recevoir les pages transmises par le serveur. Généralement, le client possède aussi un cache d'objets dans lequel il recopie les seuls objets utilisés par l'application. Les modifications d'objets doivent alors être répercutées dans le cache de pages, avant retour des pages contenant les objets au serveur.

1.2.2.2. Considérations de performance

Les principaux avantages du serveur de pages proviennent de la décentralisation d'un grand nombre de fonctionnalités au niveau des clients. Les activités de gestion d'objets et de fichiers, qui sont les plus consommatrices de temps CPU, sont en totalité à la charge des clients. Ceci induit plusieurs conséquences bénéfiques.

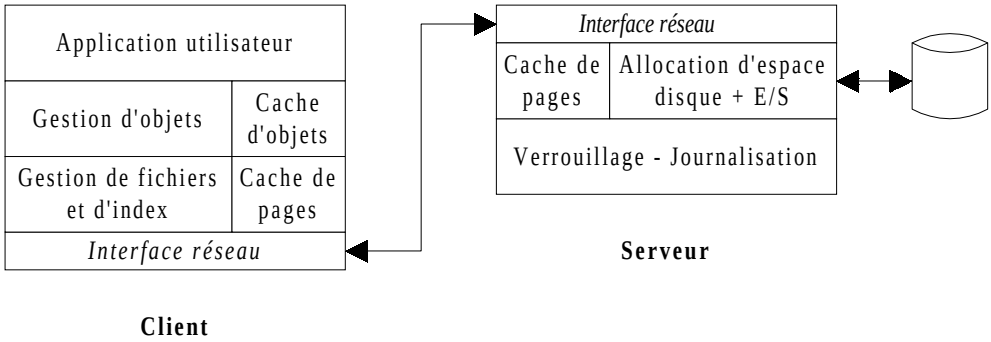

Figure 1.2 : Architecture serveur de pages

- Le serveur est déchargé de nombreuses tâches et peut se consacrer plus efficacement à l'interface E/S et au contrôle de l'utilisation des pages par les clients. Ceci permet souvent de se satisfaire d'une architecture ne comportant qu'un seul serveur, ce qui est bien plus simple à gérer.
- Les tâches les plus lourdes sont réparties sur les stations de travail, qui peuvent exploiter pleinement leur puissance de calcul.
- Les communications peuvent être optimisées car les transferts sont faits par page et plusieurs objets utiles à une application peuvent se trouver dans la même page. De plus, le coût engendré par l'envoi d'une page par réseau n'est guère supérieur à celui de l'envoi d'un objet isolé.
- L'implémentation d'une architecture serveur de pages est plus simple à réaliser. En particulier, la reprise après panne est plus simple à implanter dans le cadre d'une gestion (journalisation, entre autres) basée sur la page.
- Le serveur est moins sujet aux pannes du fait que les requêtes utilisateurs sont exécutées sur les clients.

Malgré ces avantages importants, le serveur de pages présente les inconvénients suivants.

- Une granularité de verrouillage trop forte (la page) entraîne une faible capacité à gérer les applications fortement concurrentes traitant des objets de fine granularité (beaucoup d'objets sur une même page). Un verrouillage au niveau de l'objet est très délicat à mettre en œuvre dans une telle architecture, bien que proposé et démontré comme une perspective intéressante [Carey et al. 94a].
- Un serveur de page ne « connaît » pas le concept d'objet. Il ne peut pas, au contraire d'un serveur d'objets, effectuer de présélection pour répondre à une requête ensembliste. Il faut, par exemple, pour parcourir séquentiellement les instances d'une classe, transférer toutes les pages pouvant contenir des objets de la classe et donc aussi des objets non utiles aux applications.

1.3. Contrôle de concurrence

Le contrôle de concurrence a pour fonction d'optimiser la gestion des accès concurrents aux données, afin que les transactions ne passent pas plus de temps à attendre qu'à s'exécuter. Les SGBDOO utilisent généralement un protocole de verrouillage pour le contrôle de la concurrence. Le point essentiel concerne la granularité de verrouillage, qui est souvent liée à l'architecture du système. Dans le cas des architectures client-serveur évoquées dans la Section 1.2, des études semblent privilégier l'efficacité des serveurs de pages [DeWitt et al. 90]. Cependant, les serveurs d'objets, avec une granularité fine de verrouillage, augmentent la capacité d'exécution concurrente. Les travaux de [Carey et al. 94a] tentent d'ailleurs d'exploiter la possibilité de verrouiller à l'objet dans le cadre d'un serveur de pages.

Les optimisations essentielles en matière de contrôle de concurrence portent sur la minimisation du nombre moyen de verrous gérés par le système et la maximisation de la capacité d'exécution concurrente des transactions. De nombreux travaux, intimement liés aux architectures des SGBD, traitent de ces aspects [Kim 90] [Bertino et Martino 91]. Nous ne les détaillons pas plus avant car ils n'influent pas sur la conception ou l'implantation de méthodes de groupement d'objets.

1.4. Stockage et accès aux objets

La mise en place de bonnes stratégies de stockage et d'accès aux objets est primordiale pour les performances des SGBDOO. Ces derniers utilisent généralement un modèle de stockage fixe qui est propre à chaque système. Il existe plusieurs stratégies concurrentes et/ou complémentaires pour mémoriser la structure complexe des objets [Valduriez et al. 86]. La difficulté réside en un conflit entre deux objectifs : soit faciliter l'accès à l'objet complexe entier (parcours du graphe d'objet) ; soit faciliter l'accès aux composants individuels de l'objet.

Les méthodes d'accès aux objets sont traditionnellement basées sur les index et les techniques de hachage. L'accès aux objets par référence est influencé par la façon dont les identificateurs d'objets sont implémentés [Bouzeghoub et al. 94].

1.4.1. Identification des objets

Le concept d'indentificateur d'objets [Khoshafian et Copeland 86] garantit l'unicité de l'identifiant d'un objet. Ce concept est indispensable aux gestionnaires d'objets pour accéder aux objets sur disque et en mémoire primaire. Il permet par ailleurs le partage d'objets, la composition d'objets complexes et la représentation d'un ensemble d'objets liés sous forme de graphe. Cet identifiant est communément appelé OID (*Object IDentifier*). Plusieurs techniques ont été mises en œuvre pour implémenter ce concept dans

les SGBDOO. Le type d'implémentation de l'OID influe sur l'adressage des objets et sur les performances du gestionnaire d'objets [Delobel et al. 91]. Parmi les nombreuses techniques proposées, nous retenons les deux catégories les plus représentatives, qui mettent respectivement en œuvre des *OID physiques* et des *OID logiques*.

1.4.1.1. OID physiques

Un OID physique est composé de l'adresse physique de l'objet, sur disque ou en mémoire virtuelle. Son usage est de plus en plus répandu, pour des raisons manifestes d'efficacité d'adressage des objets (les accès par référence bénéficient immédiatement de la localisation de l'objet référencé car elle est implicitement contenue dans son OID). Cette approche a notamment été défendue par le projet Altaïr avec le SGBD O_2 [Deux et al. 90]. Le gestionnaire d'objets persistants Texas [Singhal et al. 92] utilise également l'adresse physique (offset) comme identifiant de l'objet, de même qu'ODE [Agrawal et Gehani 89] et EXODUS [Carey et al. 88]. ObjectStore [Lamb et al. 91], QuickStore [White et DeWitt 94] et Cricket [Shekita et Zwiling 95] utilisent l'adresse en mémoire virtuelle de l'objet.

L'inconvénient majeur de l'OID physique provient de l'absence d'orthogonalité existant entre l'identificateur de l'objet et sa localisation physique. Ceci a des conséquences négatives sur toute action génératrice de déplacements physiques d'objets, occasionnée lors de modification de schéma ou lors de toute procédure de réorganisation de l'espace physique (réorganisation de groupement d'objets, par exemple), spécialement lorsque celles-ci sont dynamiques. Ceci implique la mise en œuvre de mécanismes particuliers pour pallier l'incohérence des références entre objets induite par les déplacements d'objets (par exemple, la technique des pointeurs de suivi [Delobel et al. 91]).

1.4.1.2. OID logiques

Un OID logique est un identifiant indépendant de la valeur de l'objet et de sa localisation, par exemple, un numéro logique dont l'attribution est gérée par le système. L'accès par référence n'est alors plus immédiat : l'OID de l'objet référencé est une entrée dans une *table d'objets* (qui doit être une structure persistante) qui donne l'adresse de l'objet sur le disque. Toute activation d'objet nécessite donc une indirection pouvant engendrer des accès disque supplémentaires. Néanmoins, l'orthogonalité de l'identificateur de l'objet par rapport à sa localisation est garantie et les déplacements physiques de d'objets sont aisés. GemStone [Copeland et Maier 84] et ORION [Kim et al. 88], notamment, présentent un identifiant logique.

Les reproches majeurs qui peuvent être faits à ce type d'approche sont les suivants :

- la table d'objets peut être trop importante pour envisager son chargement en mémoire. Il faut alors utiliser une technique d'accès qui peut se révéler coûteuse ;

- cette table est partagée par toutes les transactions et peut constituer un goulot d'étranglement ;
- au fur et à mesure de la vie de la base, les numéros logiques attribués comme identifiants à des objets supprimés créent des « trous ». Des réorganisations périodiques entraînent également la modification des OID des objets et donc les problèmes d'intégrité signalés pour les OID physiques au niveau des références inter-objets.

1.4.2. Nature de l'espace d'adressage

L'émergence des SGBDOO est souvent comparée à un mariage de raison entre les langages de programmation orientés objet et les bases de données. Ce mariage a donné lieu à deux constats : les structures bases de données sont inadaptées à une utilisation en mémoire centrale ; les structures propres aux langages de programmation sont inadaptées à un stockage persistant [Delobel et al. 91]. Les SGBDOO ont été confrontés à la présence de ces deux niveaux de représentation différents. Dans la perspective de l'intégration de ces technologies sont nés différents types de systèmes présentés ci-après. La Figure 1.3 synthétise l'ensemble de ces approches.

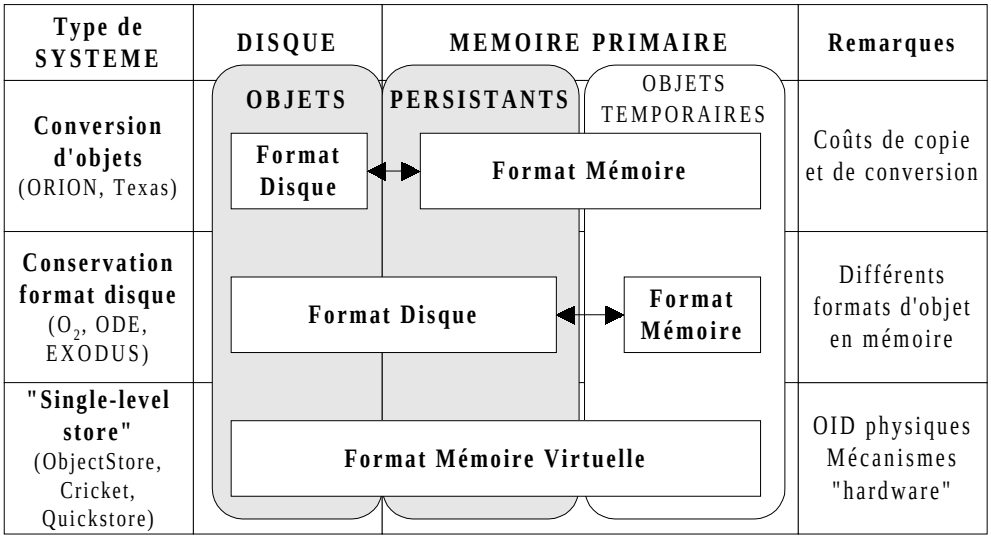

Figure 1.3 : Niveaux de représentation des objets dans un SGBDOO

1.4.2.1. Systèmes à deux niveaux de représentation

Les objets ne sont pas représentés sur disque sous un format identique à celui manipulé par un langage de programmation en mémoire primaire. Chaque langage de programmation utilise de surcroît des structures qui lui sont propres pour manipuler les objets. Le format disque et le format mémoire diffèrent sur deux points. En premier lieu, la structure même des objets est adaptée soit à un stockage disque, soit à une manipulation en

mémoire. Par ailleurs, les références inter-objets appartiennent à deux référentiels distincts, selon que les objets sont sur disque ou en mémoire. Sur disque, les références sont représentées par l'OID de l'objet référencé. En mémoire, elles sont directement représentées par l'adresse de cet objet. Ces deux niveaux de représentation cohabitent dans la plupart des SGBD. On distingue cependant les deux techniques de mise en œuvre suivantes [Copeland et al. 90].

Conversion d'objets

Chaque fois qu'un objet migre du disque en mémoire, il y a conversion de l'objet en format mémoire (structure et références inter-objets) et vice versa chaque fois qu'un objet retourne sur le disque. Cette approche est utilisée, par exemple, dans le SGBDOO ORION [Kim et al. 88]. Le gestionnaire d'objets persistants Texas [Singhal et al. 92] effectue également une conversion des références inter-objets lors du chargement d'une page en mémoire (technique du *swizzling*, dans la terminologie anglo-saxonne). L'atout majeur de cette approche provient du fait que le langage de programmation exploite les données temporaires et persistantes sous une forme homogène en mémoire (cf. Figure 1.3).

Elle comporte cependant certains désavantages du point de vue des performances :

- les conversions du format disque au format mémoire sont nombreuses et coûteuses ;
- lorsqu'il y a gestion de caches d'objets, les coûts de copies d'objets dans le cache (après conversion) sont importants ;
- du fait de la dualité de représentation des références inter-objets (OID sur disque et pointeurs en mémoire), un mécanisme logiciel de détection de conversion peut s'avérer nécessaire. En effet, suivant les techniques de conversion employées, les identifiants potentiellement référençables peuvent être convertis soit systématiquement, soit au coup par coup. Il est alors nécessaire de vérifier si une référence est sous forme convertie (*swizzled*) ou non (*unswizzled*) avant d'accéder à l'objet correspondant.

Conservation du format disque

Cette approche a pour but d'éviter les conversions, malgré la présence des deux niveaux de représentation non homogènes. Les objets sont chargés en mémoire en conservant leur format disque. Des fonctions particulières détournent alors l'accès aux objets en mémoire de sorte à présenter ces objets au langage de programmation sous un format compréhensible. Cette approche a été utilisée en particulier dans les systèmes O_2 [Deux et al. 91], ODE [Agrawal et Gehani 89] et EXODUS [Carey et al. 88].

Les avantages de cette solution sont simples : il n'existe ni copie, ni conversion de format et les accès aux objets sur disque sont facilités. Cependant, la représentation des objets demeure à deux niveaux car le système doit distinguer deux types d'objets différents

en mémoire : les objets persistants au format disque et les objets temporaires au format mémoire (cf. Figure 1.3). Par ailleurs, des structures de données doivent être gérées pour établir la correspondance entre l'adresse mémoire d'un objet persistant et son OID, car les références vers ces objets sont conservées sous forme d'OID.

1.4.2.2. Les systèmes à un niveau de représentation

Dans ces systèmes, dénommés *single-level stores* dans la terminologie anglo-saxonne, la représentation des objets (structure et références inter-objets) est dite uniforme [Copeland et al. 90]. Les objets sont stockés tels qu'ils sont manipulés en mémoire. Les objets persistants stockés sur disque, les objets persistants chargés en mémoire et les objets temporaires utilisent le même format.

Cette approche est assez délicate à implémenter car elle est basée sur l'utilisation des mécanismes internes au système d'exploitation pour gérer la mémoire virtuelle. Le principe est conceptuellement simple : la base de données est directement « mappée » dans l'espace d'adressage virtuel. Chaque objet est créé dans un segment de la mémoire virtuelle, de façon à ce que son OID et son adresse mémoire soient homogènes. Lorsqu'un objet est activé, la page qui le contient sur disque est transférée dans le segment réservé en mémoire virtuelle. Avec l'aide des mécanismes de protection d'accès du système d'exploitation, le SGBD intercepte les défauts de page qui interviennent lorsqu'une référence produit une violation d'accès (quand l'objet et sa page ne sont pas en mémoire) [Bouzeghoub et al. 94].

On attribue à cette approche à la fois une transparence des opérations de chargement et de déchargement des objets, une facilité de programmation accrue et une utilisation optimale des ressources mémoire, conjointes à une amélioration des performances. Elle est qualifiée de *hardware*. Plusieurs systèmes la mettent en œuvre : ObjectStore [Lamb et al. 91], QuickStore [White et DeWitt 94] et Cricket [Shekita et Zwiling 95].

Les qualités de cette approche en terme de performance sont indéniables [Amiel et al. 92] :

- les coûts indiqués pour les systèmes de représentation à deux niveaux sont supprimés ;
- la gestion d'objets est considérablement simplifiée.

Les défauts attribués dans un premier temps à cette technique relevaient de la limitation de l'espace persistant à la taille de l'espace d'adressage virtuel. L'émergence des systèmes d'exploitation capables de gérer des espaces adressables de plus en plus grands (adresses sur 32 ou 64 bits) et offrant des primitives de gestion mémoire efficaces a levé ce type d'objections. Par contre, cette approche impose certaines contraintes qui ont des conséquences négatives :

- les OID sont par définition physiques, ce qui induit automatiquement les inconvénients décrits à la Section 1.4.1.1 ;
- l'utilisation des mécanismes internes au système d'exploitation rend la gestion d'objets plus transparente, mais en contrepartie difficilement contrôlable ou influençable de façon logicielle ;
- lorsque le nombre d'objets chargés par une transaction dépasse la taille de la mémoire vive réelle, le mécanisme de gestion de la mémoire virtuelle replace certaines pages sur le disque. Ces pages figurent toujours en mémoire virtuelle, mais ne sont plus en mémoire vive et provoquent donc une entrée-sortie si elles sont accédées. Ce fonctionnement est également difficilement contrôlable.

1.4.3. Accès aux objets persistants

1.4.3.1. Types d'accès

[McIver et King 94] et [Bullat 96] ont identifié trois types d'accès à un objet persistant.

1) *Accès aléatoire* à un objet racine de persistance : Les objets racines sont nommés et répertoriés dans une structure persistante. Leur accès implique la recherche de l'OID de l'objet dans cette structure, par l'intermédiaire du nom de l'objet racine.
2) *Accès ensembliste* ou *associatif* : Ce type d'accès est analogue aux opérations fournies dans les SGBD relationnels. Il suppose la gestion explicite de collections d'objets manipulables au sein du langage en une seule et même opération. On parle d'*extension de classe* [Beech et Özbütün 91] lorsque ces groupes d'objets rassemblent toutes les instances d'une même classe. Certains systèmes, comme O_2, autorisent la gestion de collections nommées d'objets quelconques. Le parcours de ces collections met en jeu l'utilisation de structures persistantes comportant les OID des objets concernés par chaque collection.
3) *Accès navigationnel* ou *par référence* : Ces accès conduisent le SGBD à rechercher un objet β référencé par un autre objet α. Un des attributs de l'objet α a pour valeur l'OID de l'objet β recherché.

1.4.3.2. Mécanismes mis en jeu

Quel que soit le cas de figure dans lequel on se place parmi les trois cas évoqués à la Section 1.4.3.1, tout accès à un objet persistant se concrétise par les traitements suivants (Figure 1.4).

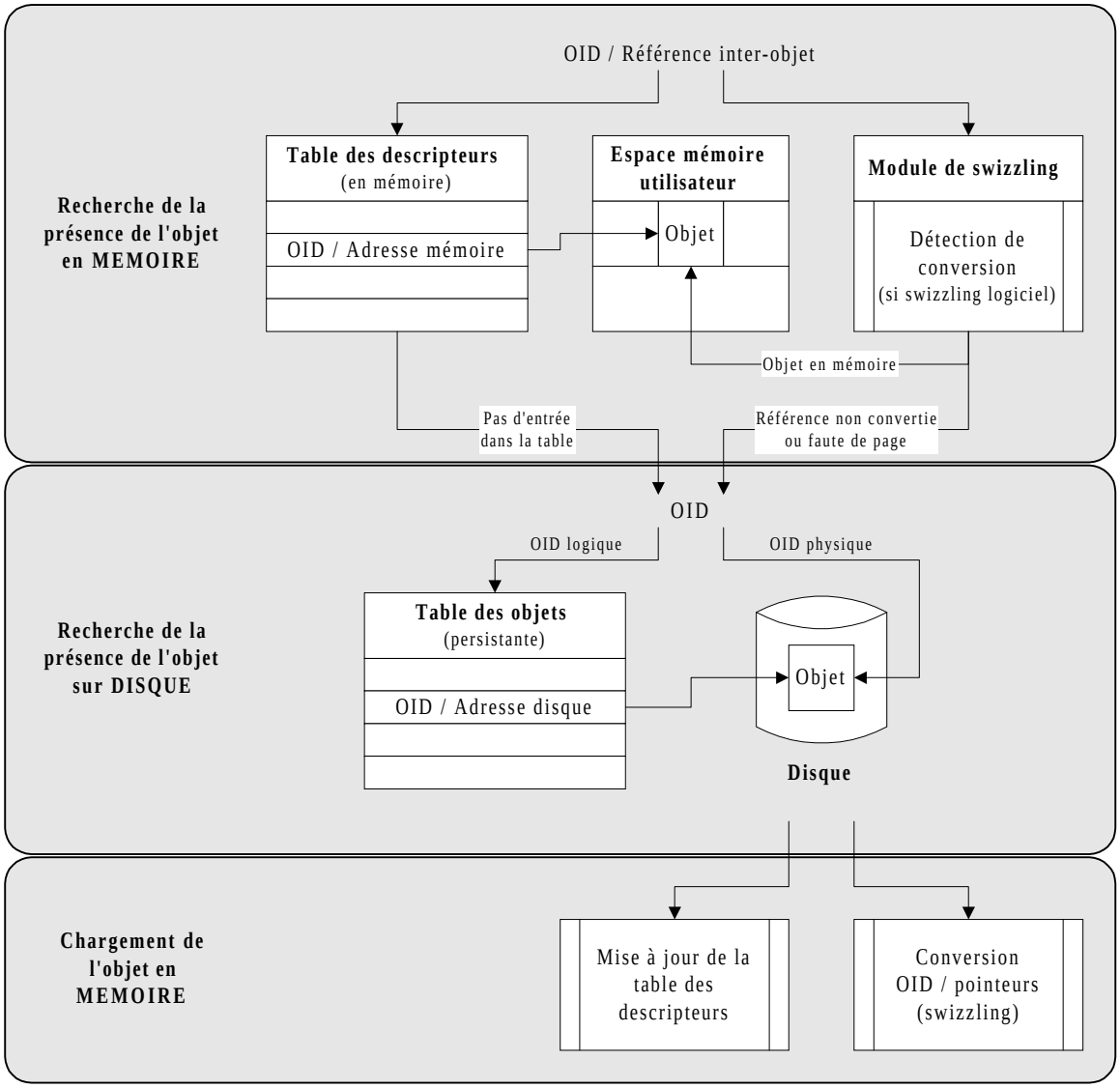

Figure 1.4 : Mécanismes mis en jeu pour l'accès aux objets dans un SGBDOO

1) Localiser l'objet sur disque

Lorsque l'objet n'a pas encore été chargé en mémoire, il faut rechercher sa localisation sur disque à partir de son identificateur (OID). Si l'OID est physique, l'accès sur disque est direct. Si l'OID est logique, il faut accéder à la *table d'objets* pour connaître l'adresse physique de l'objet sur disque.

2) Charger l'objet en mémoire

3) Permettre les futures localisations en mémoire

Pour des raisons évidentes de performance, toute demande d'utilisation d'un objet déjà présent en mémoire doit automatiquement aboutir à un accès direct à son emplacement mémoire afin d'éviter un accès disque. Pour cela, il existe trois solutions distinctes, suivant le type de système employé (cf. Section 1.4.2) :

- mémoriser la correspondance entre l'OID d'un objet et son adresse mémoire courante dans une table temporaire appelée *table des descripteurs* et n'effectuer aucune conversion OID-pointeur ;
- utiliser la technique du *swizzling* en convertissant tout ou partie des OID potentiellement déréférençables ;
- utiliser un niveau de représentation uniforme en conservant pour les OID des objets leur adresse en mémoire virtuelle.

1.4.3.3. Gestion de descripteurs

Nous avons vu que certains systèmes (O_2, par exemple) choisissent de conserver le format disque des objets lorsqu'ils sont chargés en mémoire. Afin d'assurer l'adressage des objets en mémoire, ceux-ci doivent mémoriser la correspondance OID-pointeur des objets chargés en mémoire. Au sein d'un objet α, un pointeur vers un objet β contient systématiquement l'OID de l'objet référencé β, que l'objet α réside sur disque ou en mémoire. Lorsque ce pointeur est déréférencé, c'est-à-dire que l'on tente d'accéder à l'objet β, un processus relativement coûteux recherche au sein de la *table des descripteurs* (contenue en mémoire et indexée sur l'OID) la localisation actuelle de l'objet, s'il est présent en mémoire.

1.4.3.4. Techniques de swizzling

Afin d'optimiser les accès, de nombreux SGBDOO utilisent la technique du *swizzling*. Il existe différentes manières de réaliser ces conversions OID-pointeurs [Eliot et Moss 92] [White 94] qui permettent d'éviter les recherches systématiques dans la table des descripteurs à chaque utilisation d'un objet persistant. Cette approche conduit, dans le meilleur des cas, à un accès mémoire direct à l'objet référencé, par simple déréférencement du pointeur converti. Lorsque cette technique est utilisée, le système doit savoir à chaque accès par référence si la référence inter-objet a déjà été convertie ou non. En d'autres termes, le système doit détecter si la référence pointe sur un objet déjà présent en mémoire ou bien contient encore l'OID d'un objet à rechercher sur le disque. Pour cela, deux solutions sont possibles.

Détection logicielle

Les systèmes traditionnels, comme ORION [Kim et al. 88], utilisent des procédures logicielles pour déterminer si un pointeur a été ou non converti.

Conversion hardware

Certains systèmes plus récents, comme ObjectStore [Lamb et al. 91], Texas [Singhal et al. 92] et Cricket [Shekita et Zwiling 95], utilisent les mécanismes de mémoire virtuelle pour déclencher le *swizzling*. Le principe technique est le même que celui exposé dans la

Section 1.4.2.2. Dans Texas, cependant, le niveau de représentation des objets n'est pas uniforme, comme dans ObjectStore et Cricket, car les OID des objets sont représentés par leur adresse sur disque. Toute page chargée en mémoire suite à la détection d'une violation d'accès (faute de page) voit l'ensemble de ses références converties en adresses en mémoire virtuelle. Cette adresse pointe soit sur l'objet correspondant s'il est déjà en mémoire, soit sur un espace mémoire qui lui a été réservé et dont l'accès est protégé à son tour. Ainsi, les références inter-objets sont toujours de type converti (*swizzled*) en mémoire. De cette façon, une fois qu'un objet est chargé en mémoire, son accès est aussi rapide qu'un accès à un objet temporaire, puisque la procédure de vérification logicielle est évitée.

Le reproche fait communément en retour est le coût excessif des réservations réalisées en mémoire (pour tout objet référençable dont l'utilisation ne sera pas forcément effective). C'est pourquoi le système Mneme [Eliot et Moss 90] n'effectue pas de conversion systématique à la page, mais uniquement lorsque l'objet est utilisé pour la première fois. Dans ce cas, le SGBD doit avoir recours à un contrôle de conversion logicielle avant l'accès à tout objet.

Selon [White 94], le coût engendré par les conversions *hardware* est bien inférieur aux gains de performance obtenus. Cependant, il est important de souligner que cette approche complique la mise en œuvre de certaines fonctionnalités du SGBD, notamment la vérification des contraintes d'intégrité, le recouvrement après panne, une granularité fine de verrouillage et la gestion de cache mémoire.

2. Méthodes d'optimisation des performances

2.1. Groupement d'objets

Le groupement d'objets (*object clustering*, dans la terminologie anglo-saxonne) est reconnu comme une technique cruciale permettant de réduire le nombre d'entrées-sorties effectuées par un SGBDOO et de conduire à une amélioration sensible des performances [Tsangaris et Naughton 91]. Le principe du groupement d'objets est de stocker les objets logiquement liés (c'est-à-dire, susceptibles d'être utilisés fréquemment ensemble) proches les uns des autres en mémoire secondaire. Ainsi, lorsqu'un objet est chargé depuis le disque, tous les objets qui lui sont liés sont chargés en même temps en mémoire primaire. Les accès ultérieurs à ces objets se font donc en mémoire primaire, plutôt que de nécessiter une entrée-sortie bien plus coûteuse en temps.

Un groupement judicieux des objets minimise en fait non seulement le nombre de pages disques à lire, mais aussi le nombre d'objets non pertinents chargés en mémoire, en maximisant la probabilité que la portion de base active tienne en mémoire primaire. De plus, si les objets fréquemment accédés ensemble sont stockés de façon contiguë sur

disque, l'amplitude des déplacements physiques de bras du disque est également minimisée.

Dans le cadre des systèmes relationnels, de telles optimisations sont principalement destinées à accélérer les coûteuses opérations de jointures, grâce à la création de *clusters* [Grandi et Scalas 93]. Dans une base de données objet, le regroupement pose des problèmes spécifiques, dont les origines sont essentiellement dues à la nature du modèle objet. Les objets ont des structures complexes, ils peuvent être de taille variable, ils peuvent être liés à d'autres objets (par un lien de composition, par exemple) et, finalement, ils sont les instances d'une classe qui elle-même s'intègre à une hiérarchie d'héritage. Dans ce contexte, la mise en place d'une politique de regroupement judicieuse est complexe. La nature particulière des relations structurelles et logiques existant entre les objets ainsi que les besoins spécifiques des nouveaux domaines d'application nécessitent la mise au point de nouvelles stratégies et de nouveaux algorithmes de regroupement d'objets adaptés au modèle objet. De nombreuses heuristiques ont été proposées en ce sens. Les plus représentatives des recherches effectuées en la matière sont présentées en détail dans la Section 3.

2.2. Gestion de cache mémoire

2.2.1. Principe

Toujours dans un souci d'économiser les coûts d'entrées-sorties, une gestion appropriée de cache mémoire (*buffering*, dans la terminologie anglo-saxonne) s'avère efficace. Ces caches en mémoire centrale permettent de conserver dans l'espace mémoire des utilisateurs les données les plus susceptibles d'être utilisées par leurs applications. La probabilité que les accès se fassent en mémoire et non sur disque est ainsi augmentée [Effelsberg et Haerder 84]. Un cache mémoire joue le rôle d'interface entre la mémoire principale et le disque et permet la manipulation des données en mémoire avant leur retour sur disque. La gestion de ces caches est effectuée grâce au principe de la mémoire paginée. La mémoire centrale est découpée en pages (ou blocs) et alimentée au fur et à mesure des besoins des applications. Lorsque le cache est plein, une technique de remplacement de page est mise en œuvre.

2.2.2. Techniques de remplacement de page

Ces techniques sont inspirées de celles traditionnellement mises en œuvre dans les systèmes d'exploitation. L'objectif est de remplacer les pages qui ont le moins de chance d'être référencées dans un futur proche. Nous présentons rapidement quelques-unes des techniques traditionnellement citées dans la littérature. Le détail de ces techniques est exposé dans [Effelsberg et Haerder 84].

- *Algorithme RANDOM :* La page remplacée dans le cache est choisie au hasard. Cette solution est de loin la plus facile à mettre en place, mais elle ne tient compte d'aucune considération d'efficacité.
- *Algorithme FIFO (First In, First Out) :* La page remplacée dans le cache est celle qui y a passé le plus de temps, quelle que soit l'utilisation qui en a été faite par la suite. Cette approche est uniquement appropriée au traitements de type séquentiel.
- *Algorithmes LFU (Least Frequently Used) :* La page remplacée dans le cache est la moins accédée (un compteur d'utilisation par page est maintenu). L'« âge » de la page n'est pas du tout pris en compte. Cet algorithme semble peu adapté au contexte bases de données, car une page fréquemment sollicitée pour un traitement donné va obtenir un score très élevé, alors qu'elle peut n'avoir aucune chance d'être réutilisée par la suite.
- *Algorithme LRU (Least Recently Used) :* La page remplacée dans le cache est celle dont la date d'utilisation est la plus ancienne. La technique utilisée est dite de file LRU : dès qu'une page est demandée, elle est placée dans la file. Elle en sort poussée par d'autres pages quand le temps a passé, sauf si elle y a été remise entre temps (si elle a été utilisée de nouveau).
- *Algorithme LRU-K* : LRU a été généralisé par [O'Neil et al 93]. Partant du principe que LRU ne prend pas en compte la fréquence d'utilisation des pages (il base ses décisions de remplacement de pages sur la datation de la dernière utilisation de chaque page, seulement), une page très peu souvent utilisée peut rester longtemps dans le cache, tant qu'elle n'a pas la date la plus ancienne parmi toutes les autres pages, même si elle présente une probabilité de réutilisation proche de zéro. L'idée de LRU-K est de dater non pas la dernière utilisation de chaque page, mais les K dernières utilisations. Le concept de *distance d'utilisation* est alors introduit cumulant, pour une page P, les distances individuelles (en âge) vers les K dernières utilisations de P. La page élue pour le remplacement lorsque le cache est plein est celle qui possède la distance d'utilisation la plus forte. Des travaux de simulation ont démontré que LRU-K est plus efficace que LRU(-1). Cependant, plus la valeur de K est élevée (et donc plus les informations à stocker et à manipuler sont nombreuses), plus cet algorithme est coûteux à l'exécution.
- *Algorithme CLOCK (ou « seconde chance ») :* Cet algorithme est une variation de l'algorithme FIFO. Il tente de reproduire le comportement de l'algorithme LRU, mais avec un coût d'implémentation moindre. À chaque page est associé un flag d'utilisation. Ce flag est marqué à 1 lorsque la page est utilisée. Pour choisir une page à remplacer dans le cache, toutes les pages sont examinées tour à tour, dans l'ordre (concept d'horloge ou CLOCK) à partir de la page dernièrement examinée la fois précédente. La page remplacée est la première à avoir son flag à 0. Lorsqu'un flag positionné à 1 est rencontré, il est remis à 0 pour le tour suivant, mais la page est conservée dans le cache pour cette fois.

- *Algorithme GCLOCK (Generalized CLOCK) :* Cet algorithme combine les concepts de LFU et CLOCK. Le flag d'utilisation associé à chaque page dans CLOCK est ici remplacé par un compteur d'utilisations. Lors des « tours d'horloge », ces compteurs sont décrémentés. La page à remplacer dans le cache est la première page trouvée avec un compteur égal à zéro. GCLOCK est très sensiblement meilleur que LFU.

2.2.3. Gestion de cache et groupement d'objets

L'utilisation conjointe de la gestion de cache et du regroupement d'objets permet d'accroître l'efficacité de ces deux techniques [Cheng et Hurson 91]. La nature des liens existant entre les objets nécessite la mise au point d'algorithmes de gestion mémoire adaptés. Dans le contexte objet, les techniques de remplacement de pages, en particulier, ont grand intérêt à exploiter la présence de ces liens et à tenir compte du regroupement physique des objets.

2.3. Autres méthodes usuelles

Parmi les diverses voies possibles pour réduire le nombre des entrées-sorties et conduire à une amélioration des performances, l'optimisation de requêtes [Lanzelotte et al. 92] [Cheiney et Lanzelotte 93] est représentative. Son but principal est de déterminer plusieurs plans d'exécution possibles afin d'exécuter la requête suivant le plan le moins coûteux (en entrées-sorties, essentiellement). La mise au point d'optimiseurs automatiques efficaces a été un des apports majeurs de la technologie relationnelle [Jarke et Koch 84]. Elle s'est essentiellement concentrée sur l'exécution des coûteuses opérations de jointure.

L'apparition des SGBDOO a posé des problèmes nouveaux dans ce domaine. En effet, l'accès à des objets complexes combine des expressions de navigation (parcours d'objets complexes) et des accès associatifs (jointures exprimées sur des valeurs de propriété). De plus, le modèle de coût utilisé par l'optimiseur doit prendre en compte les diverses options de stockage physique des objets sur le disque (placement et index). Des travaux récents [Bellatreche et al. 97] préconisent un *partitionnement horizontal* des classes en fragments d'instances, de façon à ce que les instances les plus utilisées soient placées dans le même fragment, et ce afin de réduire le nombre d'objets non pertinents chargés en mémoire. Des modèles de coût [Bellatreche et al. 98] ont montré que le *partitionnement horizontal* pouvait réduire le coût d'exécution des requêtes en terme d'entrées-sorties lorsque de grandes quantités de données sont manipulées, par exemple, dans les domaines de l'extraction de connaissances (*data mining*), des entrepôts de données (*data warehousing*) ou des Systèmes d'Information Géographiques (SIG). Néanmoins, cette technique n'est pas encore totalement au point, le *partitionnement horizontal* pouvant également dégrader

les performances du système en raison de la surcharge engendrée par l'application d'opérations d'union pour reconstruire le résultat d'une requête, après partitionnement.

Il existe beaucoup d'autres travaux concernant l'amélioration des performances des SGBDOO. Nous ne prétendons pas en dresser ici un panorama exhaustif. Nous retiendrons, par exemple : le principe du ramasse-miettes, destiné à libérer automatiquement l'espace mémoire (primaire ou secondaire) occupé par des objets devenus inaccessibles [Butler 87] [Bancilhon 88] ; les techniques de *swizzling* évoquées à la Section 1.4.3.4, qui sont destinées à accélérer le parcours des objets complexes [Eliot et Moss 92] [White 94]. Signalons également que beaucoup d'initiatives destinées à l'amélioration des performances ne peuvent pas s'adapter à tous les types de systèmes. En effet, une solution donnée peut s'avérer performante dans un contexte donné et inadaptée à un autre. Par exemple, la technique de *mapping* en mémoire virtuelle développée dans ObjectStore [Lamb et al. 91] et visant les avantages du *swizzling*, ne semble pas compatible avec une architecture serveur d'objets. Le surcoût occasionné par les conversions OID-adresse mémoire n'est justifiable que dans le cas d'une grande localité de référence aux objets (comme dans les applications de CAO, par exemple).

3. Méthodes de groupement d'objets

3.1. Introduction

Des techniques de partitionnement et de *clustering* sont utilisées dans de nombreux domaines et pour résoudre un large éventail de problèmes différents. Parmi les domaines qui utilisent une forme de groupement se trouvent la théorie des graphes, l'analyse financière, l'allocation de ressources, le traitement d'images, le test de programmes informatiques, l'étude des galaxies, la conception de puces électroniques, les statistiques, l'économie, la biologie et, bien sûr, les bases de données (regroupement physique sur disque, groupement conceptuel dans le domaine de l'extraction de données…) [Wiggerts 97].

[Wiggerts 97] classe ces algorithmes de groupement en quatre grandes catégories.

1) *Algorithmes de la théorie des graphes.* Ces algorithmes travaillent sur des graphes, dont les nœuds représentent des entités et les arcs des relations entre ces entités. Leur objectif est de trouver des sous-graphes qui formeront les *clusters*.

2) *Algorithmes de construction.* Ces algorithmes assignent les entités à des *clusters* en une seule passe en utilisant des *techniques géographiques* exploitant des informations comme la localisation des entités dans un plan à deux dimensions. Les *clusters* peuvent être prédéfinis ou construits « au vol » lorsque des entités leur sont assignées.

3) *Algorithmes d'optimisation.* Ces algorithmes partent d'un groupement initial et essayent de l'améliorer de façon itérative en se fiant à une heuristique.

4) *Algorithmes hiérarchiques.* Ces algorithmes construisent une hiérarchie de groupements telle que chaque niveau contient les mêmes clusters qu'au niveau immédiatement inférieur, sauf pour deux clusters qui se joignent pour en former un seul. La Figure 1.5 donne un exemple de hiérarchie pour trois entités.

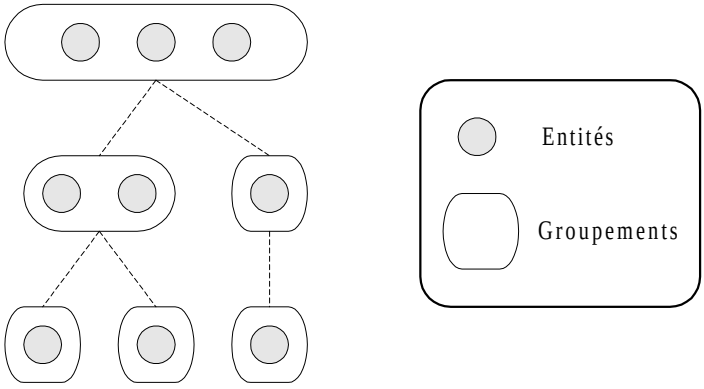

Figure 1.5 : Hiérarchie de groupements pour trois entités [Wiggerts 97]

Quelques algorithmes de regroupement physique dans le contexte des bases de données orientées objet se rangent plutôt dans la première catégorie : les algorithmes de la théorie des graphes. En effet, le problème du regroupement d'objets peut être vu comme un problème de partitionnement de graphe (en *clusters*), les nœuds du graphe étant des objets et les arcs des liens (références) entre ces objets.

Cependant, de nombreuses heuristiques traitant du groupement d'objets dans les SGBDOO sont très spécifiques et, soit ne rentrent pas dans les catégories présentées ci-dessus, soit doivent être considérées comme hybrides.

Pour présenter les techniques de groupement ci-après, nous nous appuyons sur les études bibliographiques de [Chabridon et al. 93], [Bertino et al. 94] et [Darmont et Gruenwald 96b], ainsi que sur l'état de l'art plus complet présenté dans [Bullat 95]. Nous distinguons les techniques de groupement implantées dans des systèmes existants ou des prototypes, qui sont pour la plupart statiques et activées par l'utilisateur, de propositions plus récentes, encore du domaine de la recherche et qui visent à rendre le groupement d'objets dynamique (avec reconsidération des groupements obsolètes) et transparent à l'utilisateur (automatique). Certains produits commerciaux comme MATISSE [Moller 93] et VERSANT [Houdas 93] indiquent mettre en œuvre une politique de regroupement d'objets dynamique s'adaptant au profil d'accès constatés en cours d'exploitation. Malheureusement, aucune publication ne semble faire état de la technique employée. Cependant, ceci montre que les travaux sur les produits commercialisés vont dans le sens d'une solution dynamique et automatique.

3.2. Techniques de groupement implantées dans des SGBDOO

3.2.1. Groupement selon les hiérarchies composites (ORION/Itasca)

La dénomination ORION regroupe en fait une série de prototypes de SGBDOO construits par la société MCC (Microeletronics Computer Technology Corporation) de 1986 à 1989 [Kim et al. 90]. La version commercialisée d'ORION porte le nom d'Itasca. Fonctionnant sous Unix, ce système est orienté Intelligence Artificielle (IA), CAO et Systèmes d'Information Administratifs (*Office Information Systems – OIS*) [Banerjee et al. 87]. Il s'appuie entièrement sur le langage Common Lisp, ce qui en fait un produit très « ciblé ». ORION et Itasca proposent une architecture client-serveur et une interface avec les langages C et C++. Ils sont construits au-dessus du sous-système de stockage WiSS (*Wisconsin Storage System*) [Chou et al. 85] qui permet le stockage d'enregistrements hétérogènes dans une même page disque, la spécification de la localisation d'un nouvel objet à stocker et l'allocation d'espaces physiques contigus. L'utilisation de Lisp rend très facile les évolutions de schéma dynamiques. ORION et Itasca gèrent également des versions d'objets.

ORION et Itasca sont basés sur le concept d'objet composite et fondent leur stratégie d'optimisation autour de ce concept (regroupement d'objets composites et verrouillage d'objets composites). Les liens de composition peuvent être dépendants ou indépendants, exclusifs ou partagés [Kim et al. 89]. Un lien exclusif indique qu'un objet référencé par un lien de composition ne peut être lié qu'à un seul objet « père ». Un lien partagé autorise le partage d'objets. Une référence composite dépendante indique que l'existence de l'objet référencé dépend de l'objet référençant. La destruction d'un objet entraîne la suppression de tous ses composants exclusifs et dépendants.

Les possibilités de regroupement d'objets sont les suivantes [Kim et al. 87] [Banerjee et al. 87] [Bouzeghoub et al. 94] : le système regroupe par défaut les instances d'une même classe dans un segment physique (ensemble de pages contiguës, de taille fixe) particulier organisé sous forme de B-arbre. Par ailleurs, les objets composites exclusifs et dépendants peuvent bénéficier de regroupements, grâce à la création de segments multi-classes organisés pour faciliter l'accès aux objets composants à partir de l'objet racine. C'est à l'utilisateur de spécifier quelles classes partagent le même segment.

3.2.2. Groupement exploitant des statistiques d'utilisation (Cactis, ZEITGEIST)

Cactis est un SGBDOO multi-utilisateurs dont la construction a débuté en 1985 à l'Université du Colorado [Hudson et King 89]. Cactis est conçu plus particulièrement pour supporter des applications de CAO, de PAO, des AGL et des logiciels d'aide à la conception de composants électroniques. Il fonctionne sous Unix. Sa caractéristique majeure est de proposer une stratégie d'optimisation auto-évolutive basée sur l'apprentissage. Cela suppose essentiellement une gestion de statistiques d'utilisation de la

base de données et la mise au point d'algorithmes auto-adaptatifs pour la gestion des objets.

La stratégie de regroupement physique des objets de Cactis est basée sur l'examen des accès effectués sur la base [Hudson et King 89] [Drew et al. 90]. L'objectif est de regrouper dans un même bloc (page disque) les objets qui sont fréquemment référencés ensemble. Les regroupements sont effectués au moyen d'une procédure statique, applicable à la demande, en fonction de l'état des statistiques maintenues et lorsque la base de données est inactive. À chaque réorganisation, l'espace physique est reconstruit en fonction de l'état de deux compteurs : le nombre total d'accès à chaque objet de la base et le nombre de fois où un lien de référence entre deux objets a été parcouru. D'après [Tsangaris 92], ces compteurs sont stockés au sein de chaque objet sous forme d'attributs supplémentaires (un pour la fréquence d'utilisation de l'objet et un par lien de référence partant de l'objet). La réorganisation de la base de données est incrémentale. Elle s'effectue selon l'algorithme glouton présenté en Figure 1.6. Cet algorithme a également été implémenté dans le système ZEITGEIST [Ford et al. 88].

```
Répéter :
     Choisir l'objet le plus référencé dans la bases de données qui n'ait pas été
     assigné à un bloc.
     Placer cet objet dans un nouveau bloc.
     Répéter :
          Choisir une relation appartenant à un objet présent dans le bloc,
          telle que :
               (1) la relation est connectée à un objet non assigné à un bloc
                    et hors du bloc et
               (2) le compteur d'utilisation pour cette relation est le plus
                    grand.
          Assigner l'objet attaché à cette relation au bloc courant.
     Jusqu'à ce que le bloc soit plein.
Jusqu'à ce que tous les objets soient assignés à des blocs.
```

Figure 1.6 : Algorithme de groupement de Cactis [Hudson et King 89]

Des études basées sur un banc d'essais [Drew et al. 90] concluent à une amélioration maximale des performances de 60 % après regroupement. Il faut cependant noter que ces chiffres ne sont pas pondérés par le coût de la réorganisation, qui entraîne l'indisponibilité complète de la base, ni par la fréquence souhaitable de ces réorganisations.

3.2.3. Groupement basé sur des arbres de placement (O_2)

O_2 [Deux et al. 91] [Vélez et al. 91] a été développé de 1986 à 1990 dans le cadre du Groupement d'Intérêt Public Altaïr comprenant l'INRIA, l'Université Paris XI, le CNRS, Bull et IN2. Il a été développé et commercialisé depuis 1991 par la société O_2 Technology, qui a par la suite fusionné avec les sociétés Unidata et VMARK pour former la compagnie Ardent Software. O_2 s'adresse à tout type d'application orientée objet et propose un

environnement de développement graphique très puissant incluant un générateur d'interfaces graphiques (O_2Look) et un outil de conception de base de données orientée objet (O_2Graph). O_2 possède des interfaces vers différents langages tels C, C++ et Java et supporte les standards ODMG, Xopen/XA, ANSI SQL et ODBC. O_2 offre également un modèle et un langage de requêtes (OQL) propres. Comme ORION, O_2 est bâti au-dessus du sous-système de stockage WiSS [Chou et al. 85]. Le noyau du système (O_2Engine) prend en charge le placement des données sur le disque.

Le regroupement d'objets dans O_2 a fait l'objet d'une implantation dans la version 4.5 du produit (qui en est actuellement à sa version 5.0) [O_2 Technology 94]. La stratégie de regroupement proposée est basée sur le concept d'*arbre de placement* [Benzaken 90a] [Benzaken et Delobel 90]. Ces arbres sont déclarés par l'administrateur au moyen d'un langage de définition et constituent un schéma de placement à partir duquel O_2 regroupe les objets automatiquement, à chaque création d'un nouvel objet (en fonction des liens de composition et de référence composant les arbres de placement). La modification des arbres agit sur le stockage des futurs objets, mais ne provoque aucune réorganisation dynamique. [Benzaken 90b] propose un modèle de coût permettant d'évaluer le mérite de chaque arbre de placement et d'assister l'administrateur dans la construction d'arbres optimaux. Cette évaluation est basée sur l'examen des méthodes de classes. Elle n'a pas encore été implantée dans O_2.

Un arbre de placement caractérise la façon dont un objet complexe et ses objets composants doivent être physiquement regroupés. Il correspond à un sous-arbre extrait du *graphe des types*. Un exemple de graphe des types est présenté dans la Figure 1.7, ainsi que les descriptions O_2 correspondantes. Le graphe de la classe *Ville* est réduit à un seul nœud car toutes ses propriétés sont de type atomique.

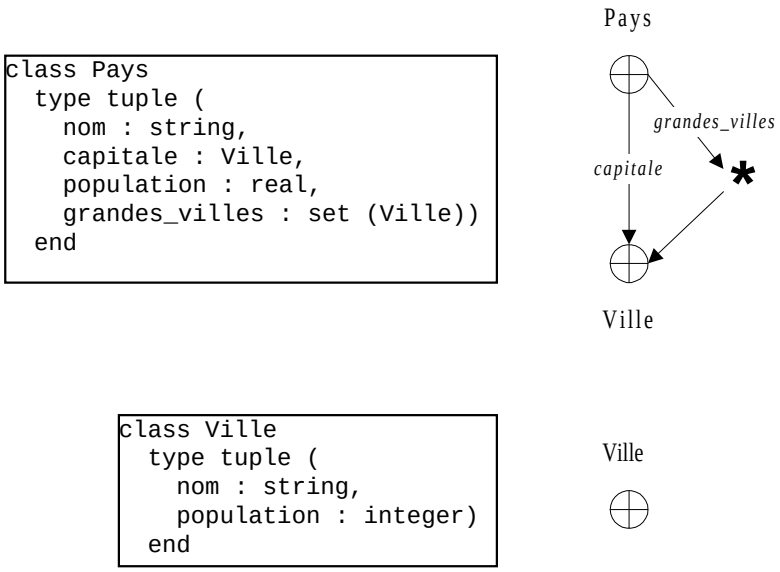

Figure 1.7 : Exemple de graphe des types O_2

Une classe donnée ne peut être racine que d'un arbre de placement, mais elle peut figurer dans plusieurs arbres. Un fichier est associé à chaque arbre de placement. Les objets n'appartenant à aucun arbre sont stockés dans un fichier particulier. Deux arbres de placement envisageables pour le schéma de la Figure 1.7 sont présentés sur la Figure 1.8. L'arbre de placement n° 1 indique que la *capitale* et chacune des *grandes_villes* appartenant au pays seront stockées près du pays d'appartenance. La solution proposée par l'arbre de placement n° 2 groupe uniquement les *grandes_villes* avec leur pays d'appartenance. Les objets de la classe *Ville* qui ne sont pas des *grandes_villes* sont dans ce cas stockés selon la règle régissant les objets ne bénéficiant pas de regroupement.

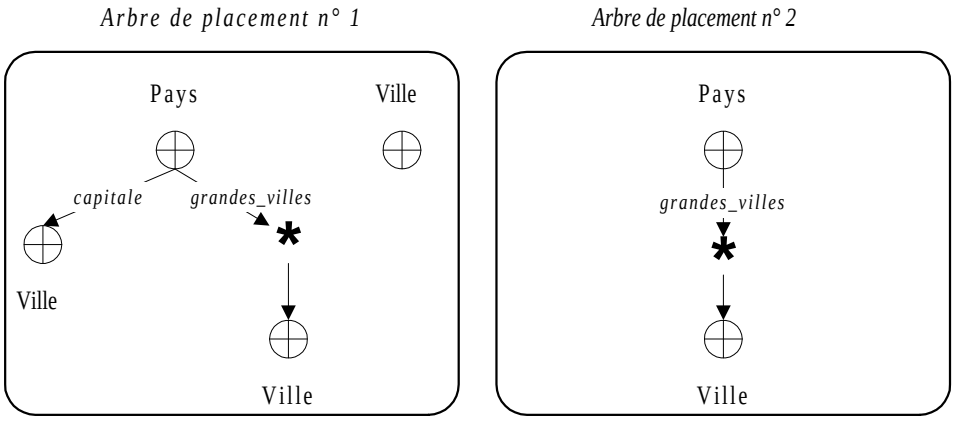

Figure 1.8 : Exemples d'arbres de placement O_2

L'algorithme de regroupement de O_2 est détaillé dans [Benzaken 90a]. La sémantique des commandes permettant la création, la suppression et la visualisation des arbres de placement est présentée dans [O_2 Technology 94]. Les conflits dus aux éventuels partages d'objets sont réglés par l'administrateur, qui définit des priorités. Un arbre défini au niveau d'une superclasse est valable au niveau des sous-classes. Les conflits occasionnés en cas d'héritage multiple sont également résolus par l'affectation de priorités.

3.2.4. Groupement explicite contrôlé par l'utilisateur (GemStone, ObjectStore, ONTOS)

GemStone [Maier et al. 86] [Butterworth et al. 91] est un SGBDOO client-serveur, développé à partir de 1983 et commercialisé depuis 1988 par la société Servio Logic. Basé sur le langage Smalltalk, il se situe sur le marché de la bureautique, de la gestion et de la CAO. GemStone est disponible sous Vax/VMS, Unix et sur micro-ordinateurs PC ou Macintosh en ce qui concerne la partie client. Il offre une interface graphique évoluée permettant la visualisation des objets et du schéma de la base de données.

ObjectStore [Lamb et al. 91] est un SGBDOO client-serveur développé depuis 1988 et commercialisé depuis 1991 par la société Object Design. Il s'adresse plus particulièrement

aux domaines de la CAO et de la PAO. Le système est écrit avec les langages C et C++ et utilise Motif, OpenLook ou Windows pour la partie interface. Il utilise les mécanismes de mémoire virtuelle pour l'accès aux objets persistants, ce qui lui confère d'excellentes performances de navigation inter-objets. Il permet par ailleurs la gestion de versions et offre une gestion transactionnelle très poussée prenant en compte des transactions longues et/ou imbriquées. Le produit est disponible sous Unix et Windows et connaît un bon succès commercial.

ONTOS [Ontologic Cie 90] [Andrews et al. 91] est un SGBDOO construit autour de C++ et commercialisé depuis 1989 par la société ONTOS Inc. Il a succédé au produit VBase [Andrews et Harris 87], commercialisé par la même société (dénommée alors Ontologic) de 1987 à 1989. Les performances de VBase étaient liées à l'efficacité de son gestionnaire d'objets supportant un gros cache d'objets. ONTOS est plutôt dédié aux applications techniques (administration de réseaux, ingénierie…). Le système est écrit en C++ et fonctionne sous Unix et OS/2. Il offre une bibliothèque de classes administrant la persistance, la gestion de schémas et le traitement des exceptions. Il permet l'évolution dynamique du schéma et la gestion de transactions longues.

Ces trois systèmes ont pour point commun de proposer des mécanismes de bas niveau pour regrouper les objets sur disque. Les regroupements sont laissés à la seule initiative du programmeur, qui dispose pour cela de deux options : spécifier auprès de quel autre objet persistant stocker cet objet ou spécifier dans quelle région stocker un objet. Dans le cas d'ObjectStore, la région de regroupement est le segment (ensemble de pages contiguës) [Lamb et al. 91] [Kemper et Moerkotte 94]. Pour GemStone, la région de regroupement peut être la page ou un ensemble de pages (*bucket*) [Khoshafian et Abnous 90] [Servio Corporation 92]. ONTOS est un peu plus souple puisqu'il gère différentes régions de regroupement : segment, bloc (*chunk*), zone (*area*), groupe [Ontologic Cie 90].

3.3. Techniques relevant du domaine de la recherche

3.3.1. Stratégies exploitant des statistiques d'utilisation

3.3.1.1. Proposition de McIver et King

Cette méthode a été proposée par William McIver Jr. et Roger King, de l'Université du Colorado, Boulder. Elle est spécifiquement adaptée aux applications d'ingénierie (CAO, AGL, Systèmes d'Information Administratifs). Elle se base sur deux constatations : de nombreux algorithmes de regroupement d'objets sont statiques et ne peuvent pas s'adapter à des changements dans l'utilisation de la base de données ; les algorithmes adaptatifs ne traitent pas le problème des objets complexes de façon à s'adapter à des méthodes d'accès variées à ces objets.

Pour répondre à ces problèmes et offrir un groupement des objets dynamique (*on-line*) et adaptatif, [McIver et King 94] propose de découpler les phases de détermination du placement des objets sur disque et leur placement effectif. La stratégie repose sur l'exploitation de trois modules s'exécutant de façon concurrente.

- *Collection de statistiques :* L'algorithme combine la détection des statistiques d'utilisation de la base utilisées dans Cactis (cf. Section 3.2.2) ainsi que des renseignements concernant l'usage sélectif de parcours en profondeur ou en largeur d'abord, qui sont respectivement assimilés à des accès navigationnels et associatifs.
- *Analyse du groupement :* L'algorithme d'analyse des *clusters* utilise une variation de l'algorithme de groupement de Cactis (Figure 1.6). Ce dernier détermine d'abord l'objet le plus référencé dans la base de données. Puis, les objets qui lui sont liés par référence sont groupés sur la même page disque que lui en profondeur d'abord, par ordre décroissant de fréquence de co-utilisation avec cet objet. La variation préconisée est d'utiliser des parcours en profondeur d'abord lorsque les accès navigationnels sont prépondérants et des parcours en largeur d'abord lorsque ce sont les accès associatifs qui sont prépondérants. Le type d'accès à sélectionner est déterminé par les statistiques d'utilisation de la base.
- *Réorganisation :* Les objets sont réarrangés sur le disque pour que l'organisation physique de la base de données corresponde avec les assignations de page suggérées par l'analyse du groupement.

La collecte des statistiques est un processus multitâche qui fonctionne de façon asynchrone par rapport à l'exécution des transactions et aux autres éléments de la méthode de regroupement. L'analyse du groupement est déclenchée après collecte d'une quantité représentative de statistiques par un mécanisme de *trigger*. Une phase de réorganisation n'est pas forcément nécessaire après chaque analyse de groupement. Le cas échéant, elle est déclenchée par un mécanisme de *trigger* et ne concerne que les objets qui n'ont pas déjà été regroupés.

Les performances de cette méthode ont été évaluées grâce au banc d'essais *Trouble Ticket Benchmark* [McGlenaghan 91]. L'étude met en évidence la pertinence des statistiques collectées et du groupement préconisé, mais aussi une surcharge importante engendrée par les phases de réorganisation de la base, qui verrouillent la base et provoquent la suspension de l'exécution des transactions.

3.3.1.2. Propositions de Chang et Katz / Gay et Gruenwald

Cette stratégie a été proposée à l'origine par Ellis Chang et Randy Katz, de l'Université de Californie, Berkeley. Elle a été motivée par des besoins spécifiques à la CAO et par le souhait de fonder une politique de groupement sur des critères autres que des ordres de placement utilisateur, jugés trop aléatoires. Ses concepteurs insistent sur le

fait que les regroupements doivent tenir compte des liens structurels entre objets, particulièrement en CAO.

La stratégie de regroupement proposée, qualifiée d'intelligente [Chang et Katz 89] [Chang et Katz 90], propose un regroupement physique basé sur un lien d'héritage particulier dénommé *instance à instance* ainsi que sur la déclaration des fréquences d'accès estimées associées à chacun des trois types de liens pris en compte (composition, équivalence, versions). Ces fréquences d'accès et le calcul des coûts d'obtention des attributs hérités permettent d'identifier la page d'accueil d'un nouvel objet. Un couplage entre une gestion de cache appropriée et le regroupement, destiné à déclencher des réorganisations dynamiques, est également proposé. La méthode présentée est basée sur les trois points suivants.

1) La relation d'héritage classique (IS-A) est étendue au concept d'héritage nommé *instance à instance*. L'idée avancée ici est de rendre possible l'héritage d'informations le long de tout type d'attribut et donc, particulièrement, de lien inter-objets. Par exemple, il semble intéressant que, lors de la création d'une nouvelle version d'un objet, celle-ci hérite automatiquement des liens de composition qu'a son ancêtre avec d'autres objets. Les auteurs préconisent ainsi un héritage de type *instance à instance* entre objets composites et objets composants, entre versions ancêtres et versions descendantes, et entre objets équivalents (qui sont différentes représentations d'un même objet réel). Les informations héritées ne sont stockées qu'une seule dois, ce qui permet un gain d'espace disque important, mais impose un placement physique des objets au plus près des informations héritées. Lorsqu'un objet est créé, un calcul de coût permet de choisir entre une implémentation par copie ou par référence.

2) Un regroupement physique est basé sur le lien d'héritage d'*instance à instance* ainsi que sur des fréquences d'accès par type de lien (composition, équivalence, versions) estimées par l'utilisateur. Les fréquences estimées et les coûts d'obtention des attributs hérités permettent d'identifier la page ciblée pour le placement d'un nouvel objet. Si la page cible est pleine, le système a le choix entre effectuer une opération de division de cette page ou élire la prochaine page identifiée comme convenable.

3) Un couplage est effectué entre regroupement et gestion de cache. La technique de remplacement de pages utilisée est une adaptation de l'algorithme LRU permettant de mieux tirer parti des regroupements existants [Chang 89]. Elle se base sur l'affectation de priorités sur les pages présentes en mémoire. Les pages fréquemment utilisées voient leur priorité croître, ainsi que les pages structurellement reliées à elles, au contraire des pages non utilisées, qui voient leur priorité diminuer avec le temps.

Aucune implantation de cette méthode n'a été effectuée, hormis dans des modèles de simulation [Chang 89] [Darmont et Gruenwald 96a] [Gay et Gruenwald 97], qui indiquent

une amélioration potentielle des performances pouvant atteindre 200 % sous certaines conditions.

Jean-Yves Gay et Le Gruenwald, de l'Université d'Oklahoma, Norman, ont récemment proposé une extension de cette méthode. Son principe [Gay et Gruenwald 97] est de remplacer les fréquences d'accès estimées par l'utilisateur pour les trois types de liens (composition, équivalence, versions) par des statistiques d'utilisation, plus fiables. Des statistiques concernant l'accès des objets en lecture ou en écriture ont également été ajoutées. Le regroupement est effectué de manière automatique, à la création ou à la modification des objets et quand un mauvais groupement est détecté. L'utilisateur a la possibilité d'influer sur le processus de regroupement à l'aide d'un ensemble de paramètres.

Un mauvais groupement est détecté lorsque le rapport entre le nombre de blocs (ensemble de pages contiguës) lus dans le cache mémoire et le nombre de blocs lus sur disque est inférieur à un seuil déterminé par le système et que le nombre de statistiques collectées est suffisant. Cette détection met fin à la collecte des statistiques et initie une phase de détermination d'un nouveau regroupement qui spécifie quels objets sont susceptibles d'être regroupés (c'est-à-dire, quels objets présentent des statistiques d'utilisation satisfaisantes). Le placement physique des objets utilise un algorithme similaire à celui de Chang et Katz, mais supporte en plus la duplication d'objets. En effet, les objets peuvent être dupliqués afin d'augmenter la localité de référence. La décision de duplication ou non d'un objet est régie par les statistiques d'accès en écriture et en lecture à cet objet. Un objet plus consulté que modifié sera candidat à la duplication.

L'efficacité de cette méthode a été comparée par simulation à celle d'ORION et de Cactis [Gay et Gruenwald 97], mais pas avec celle de Chang et Katz sur laquelle elle se base. Cette étude conclut à la supériorité de la méthode proposée par rapport à celle d'ORION et de Cactis. Les résultats sont d'ailleurs très similaires à ceux obtenus par [Darmont et Gruenwald 96a] pour la comparaison des méthodes de Chang et Katz, ORION et Cactis. Aussi est-il difficile de se prononcer sur les réelles améliorations de cette extension par rapport à la proposition originale de Chang et Katz, en matière de performance.

Une implémentation de cette technique au sein du gestionnaire d'objets persistants Texas [Singhal et al. 92] a récemment été effectuée à l'Université d'Oklahoma [Régnier 98b]. Cependant, elle n'inclut pas à ce jour toutes les fonctionnalités prévues dans [Gay et Gruenwald 97] (duplication des objets, par exemple) et aucune évaluation de performance probante n'a encore été menée à bien.

3.3.1.3. DSTC

La stratégie de regroupement DSTC (*Dynamic, Statistical and Tunable Clustering*) a été proposée par Frédérique Bullat, de l'Université Blaise Pascal, Clermont-Ferrand II,

dans le cadre de sa Thèse de Doctorat d'Université [Bullat 96]. DSTC comprend en fait une politique dynamique de regroupement physique d'objets, ainsi qu'une politique de remplacement de pages dans le cache mémoire et une stratégie de préchargement orientées groupement.

L'objectif de base de la méthode est de regrouper ensemble sur le disque les objets qui sont utilisés conjointement (à des instants proches dans le temps). La solution proposée [Bullat et Schneider 96] consiste à capter des informations sur l'utilisation des objets, tout en respectant les contraintes suivantes : minimiser la quantité des informations gérées, maximiser la pertinence des informations collectées, réduire les coûts de stockage persistant de ces informations et minimiser la perturbation exercée sur les transactions en cours. Cet objectif est atteint en étageant les informations collectées à différents niveaux et en utilisant des filtres progressifs sur des statistiques maintenues en mémoire primaire. Ainsi, il est possible de ne stocker de façon persistante que des informations susceptibles d'être significatives.

Les statistiques d'utilisation de la base de données concernent la fréquence d'accès aux objets ainsi que la fréquence d'utilisation des références inter-objets. Tous les types de liens sont considérés comme des références physiques, qu'il s'agisse de liens structurels construits au niveau du schéma, de liens logiques dépendant des applications ou de liens issus de la fragmentation physique des objets (due à l'implémentation de certains SGBDOO). Tout accès physique d'un objet à partir d'un autre est détecté et comptabilisé.

La réorganisation physique des objets est déclenchée par un mécanisme de *trigger*. L'algorithme qui organise le stockage des objets sur le disque est orienté ordonnancement. Il propose la formation d'une séquence linéaire d'objets respectant les « forces d'attraction » s'exerçant entre les objets. Cette séquence est transcrite séquentiellement dans un *cluster*, c'est-à-dire un segment contigu de taille variable. Cet algorithme s'inspire entre autres de la solution proposée par [Cheng et Hurson 91] (cf. Section 3.3.2).

La flexibilité de cette approche est obtenue par divers paramètres permettant d'adapter la réactivité du système au comportement de la base. La détermination de ces paramètres revient à l'administrateur de la base de données.

La mise en œuvre de la stratégie DSTC s'articule autour de cinq phases.

1) *Phase d'Observation :* Pendant une *Période d'Observation* prédéfinie, les statistiques d'utilisation des objets sont collectées et stockées en mémoire primaire dans la *Matrice d'Observation*.

2) *Phase de Sélection :* Les données stockées dans la Matrice d'Observation sont triées et filtrées. Seules les statistiques significatives sont conservées.

3) *Phase de Consolidation :* Le résultat de la Phase de Sélection est utilisé pour mettre à jour les données collectées lors des précédentes Périodes d'Observation, qui sont stockées de façon persistante dans la *Matrice Consolidée*.

4) *Réorganisation dynamique des clusters :* Les statistiques de la Matrice Consolidée sont utilisées pour construire des *Unités de Regroupement* ou modifier des Unités de regroupement existantes.

5) *Réorganisation physique de la base :* Les Unités de Regroupement sont finalement utilisées pour examiner un nouveau placement des objets sur disque. Cette phase est déclenchée lorsque le système est inactif.

La gestion de cache associée à DSTC a pour principe, lors de l'accès par une transaction à un objet appartenant à un *cluster*, d'occasionner le chargement du *cluster* entier et pas uniquement de la page contenant cet objet, puisque les objets du *cluster* auront de grandes chances d'être utilisés par la transaction. Un algorithme de remplacement de pages dénommé LRU-C est également proposé. Son principe consiste à dater uniquement les *clusters* présents dans le cache plutôt que de dater l'utilisation des pages.

La stratégie DSTC a été implantée dans le gestionnaire d'objets persistants Texas [Singhal et al. 92], d'abord sur station de travail Sun (version 0.2 de Texas) [Mallordy 96], puis sur PC sous Linux (version 0.5 de Texas) [Régnier 98a] [Robert 98]. DSTC a fait l'objet d'une étude de performance à l'aide d'un banc d'essais baptisé DSTC-CluB [Mallordy 96] [Bullat 96]. Ces travaux ont démontré l'efficacité de DSTC sur des cas simples, mais ils s'avèrent insuffisants pour cerner correctement l'influence des nombreux paramètres de la méthode sur son comportement.

3.3.2. Regroupement multi-critères

Cette stratégie a été proposée par Jia-bing Cheng et A.R. Hurson, de l'Université de Pennsylvanie, Philadelphie. Elle a été motivée par les besoins propres à la CAO, en déplorant le fait que les stratégies existantes soient généralement basées sur un critère de regroupement unique.

Le regroupement proposé est qualifié de multi-niveaux (*multi-level*) [Cheng et Hurson 91]. Il permet de regrouper les objets selon plusieurs critères à la fois. La méthode consiste à affecter un coefficient d'importance aux trois critères présentés ci-après. Le regroupement est alors effectué en tenant compte de ces trois types de liens, de sorte que le degré de proximité entre deux objets soit proportionnel à l'importance des liens qui les relient. Les regroupements sont effectués par le système lui-même, sans requérir de spécification externe. Par ailleurs, cette stratégie est dotée d'un modèle de coût évaluant le bénéfice d'une réorganisation dynamique éventuelle.

Les critères pris en compte par cette stratégie sont les trois types de relations identifiées par [Chang et Katz 89] : l'équivalence entre objets (différentes représentations d'un même objet réel peuvent être des instances de différentes classes), les relations de composition et les liens entre versions d'un même objet. La méthode est présentée en détail dans [Cheng et Hurson 91]. Elle travaille à partir d'un graphe des références entre objets dans lequel les arcs modélisent les trois types de liens que nous venons de citer et les

nœuds les objets à regrouper. Chaque arc possède un poids représentant l'importance estimée du lien inter-objets correspondant. Par exemple, un poids de 1 peut être attribué aux relations d'équivalence, de 2 aux liens de composition et de 3 aux liens entre versions, considérant ces derniers comme les plus importants. Le graphe est orienté et peut comporter des cycles.

L'algorithme de regroupement (Figure 1.9) a pour objectif d'ordonnancer les nœuds (objets) du graphe de façon à ce que le degré de proximité entre deux objets soit proportionnel à l'importance des liens qui les relient. L'ordonnancement obtenu correspond à une séquence physique de stockage intéressante. Le fonctionnement de l'algorithme est montré par la Figure 1.10. La séquence de regroupement obtenue après exécution de l'algorithme (*G H E A B C D E F I* sur notre exemple) présente deux propriétés intéressantes :

- tout objet accessible sur le graphe de départ à partir d'un objet donné est placé derrière celui-ci dans la séquence ;
- plus deux objets sont proches dans la séquence, plus les liens existants entre eux sont forts (poids de l'arc).

```
Tant que le graphe n'est pas réduit à un seul super-nœud :
     Identifier la liste L des arcs de poids le plus fort.
     Pour chaque arc A appartenant à L :
          Rassembler les nœuds reliés par l'arc A en un même super-nœud, de
          sorte que le nœud de départ précède le nœud d'arrivée dans le
          super-nœud.
     Fin Pour
     Pour chaque super-nœud S formé :
          Pour chaque nœud N (simple ou super-nœud) relié à S par un arc :
               Ne conserver que l'arc de poids le plus fort parmi tous les arcs
               reliant S à N
          Fin Pour
     Fin Pour
Fin Tant que
L'ordonnancement recherché est égal à la séquence formant le super-nœud final.
```

Figure 1.9 : Algorithme de regroupement multi-critères [Cheng et Hurson 91]

Cette proposition n'a fait l'objet d'aucune implantation, bien qu'elle ait inspiré une partie de la méthode DSTC (cf. Section 3.3.1.3). Par contre, elle a été étendue au domaine des bases de données réparties par [Min et al. 93], dont l'idée est de traiter les problèmes d'allocation d'objets sur différents sites et d'organiser au mieux l'espace physique sur ces sites. Les objets sont alloués et regroupés sur un site en fonction de statistiques d'utilisation en lecture et en écriture de ces objets. Dans le cas général, le choix du site d'allocation tient compte à la fois des fréquences d'utilisation et des liens entre objets. Le site présentant le moindre coût est choisi.

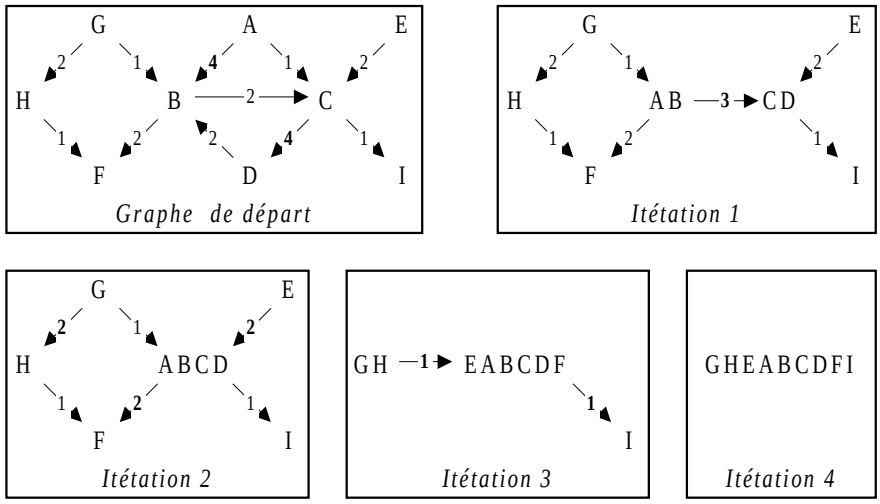

Figure 1.10 : Exemple de fonctionnement de l'algorithme de Cheng et Hurson

Si aucune statistique n'est disponible pour un objet donné, il est alloué sur le site où sont stockés les objets avec lesquels il est le plus fréquemment lié. L'organisation des groupements sur le site suit l'algorithme de la Figure 1.9. Il est possible d'autoriser la duplication d'objets sur différents sites.

3.3.3. Autres propositions

3.3.3.1. Partitionnement de graphe

La première proposition issue de la théorie des graphes a été effectuée par Emmanuel-Manolis Tsangaris, de l'Université du Wisconsin, Madison, dans le cadre de son *Doctorate of Philosophy* [Tsangaris 92]. L'idée de base est d'assimiler une base de données orientée objet à un graphe dont les nœuds sont les objets et les arcs les liens (références) existants entre ces objets. Les arcs sont pondérés. Le problème du regroupement d'objets revient alors à un problème de partitionnement de graphe en sous-graphes qui constitueront des *clusters*.

[Tsangaris et Naughton 91] propose deux modèles d'accès aux objets (Figure 1.11). Pour le premier (IID), qui attribue à chaque objet une probabilité d'être référencé, il existe des solutions de partitionnement exactes issues de la théorie des graphes, mais le modèle est considéré comme trop simple. Pour le second modèle d'accès (SMC), où les probabilités concernent l'accès à un objet à partir du précédent objet référencé, le problème est NP-complet [Garey et Johnson 79] [Tsangaris 92], mais il existe de bonnes heuristiques pour le résoudre. [Tsangaris et Naughton 91] propose une nouvelle heuristique utilisant une approche stochastique, basée sur l'heuristique de [Kernighan et Lin 70]. Cet algorithme glouton maximise la localité des objets en groupant ensemble les objets reliés par des arcs de poids fort. Une étude de simulation [Tsangaris et Naughton 92] a montré

l'efficacité de cette approche stochastique par rapport aux heuristiques existantes de la théorie des graphes.

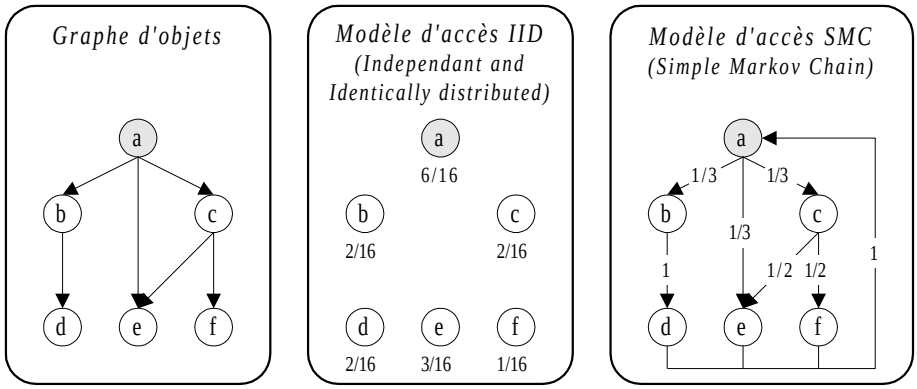

Figure 1.11 : Modèles d'accès aux objets de [Tsangaris et Naughton 91]

Dans [Gerlhof et al. 93], le choix de l'algorithme de Kernighan et Lin par E.-M. Tsangaris comme base de travail est critiqué, car il présente une trop grande complexité d'exécution, ce qui rend son application impossible sur des bases d'objets de taille importante. En conséquence, une nouvelle classe d'algorithmes gloutons de partitionnement est proposée. Ces algorithmes sont de moindre complexité, mais les expériences menées par leurs auteurs montrent qu'ils fournissent des résultats de bonne qualité. [Wierzyk 96] reprend également cette approche et propose encore un nouvel algorithme stochastique très efficace, de complexité O(*N*) (linéaire).

Nous formulons plusieurs réserves à l'égard de ces travaux, qui ne peuvent pas être utilisés tels quels dans un contexte bases de données. En effet, ces algorithmes de partitionnement de graphe présupposent la pondération des arcs connue. Il n'est pas fait allusion à la façon d'obtenir le poids des arcs, bien qu'il puisse sans doute être calculé grâce à des statistiques d'utilisation de la base. Ces algorithmes fonctionnent également uniquement sur des graphes orientés acycliques. Or, des références entre objets dans une base de données réelle peuvent parfaitement présenter des cycles. Mais le plus gros reproche qui peut être attribué à ces méthodes est qu'elles ne s'appliquent que si la taille des objets est fixe, ce qui n'est pas réaliste. Les bases d'objets réelles contiennent des instances de différentes classes et donc des objets de tailles variées et éventuellement très hétérogènes. Dans [Gerlhof et al. 93], ce problème n'est abordé que de façon très superficielle : les variations de taille d'objet consistent en fait à modifier la taille des objets en fonction de la taille des pages disque, tout en gardant une taille d'objet homogène pour toute la base (par exemple, des tests de partitionnement sont effectués en regroupement respectivement 5 et 20 objets par page). Finalement, aucune implantation de ces techniques, par ailleurs statiques, n'est proposée. Ces travaux restent théoriques.

3.3.3.2. Utilisation de la technique du ramasse-miettes

Olivier Gruber, de l'INRIA, a proposé que le regroupement des objets dans l'environnement réparti EOS [Gruber et al. 92] s'appuie sur une exploitation du glaneur de cellules de ce système [Amsaleg 95] afin de détecter les placements d'objets inopportuns. Cette solution [Gruber 92] [Gruber et Amsaleg 92] doit permettre la mise en œuvre d'une politique de regroupement dynamique avec un surcoût de traitement extrêmement faible. Les spécifications de groupement sont indiquées par l'administrateur de la base de données, qui pondère les arcs du graphe d'agrégation des classes en fonction des probabilités d'accès estimées. Les objets sont placés à leur création près du parent de poids le plus fort. Les placements sont ensuite réévalués par le processus chargé de « ramasser les miettes » sur le disque et éventuellement modifiés en différé.

Cette proposition n'a fait l'objet d'aucune implémentation. L'auteur donne dans [Gruber 92] des éléments quant à la faisabilité et le faible coût de mise en œuvre de cette technique. Cependant, celle-ci est intimement liée, pour réorganiser la base, à la présence d'un glaneur de cellules agissant en permanence sur le disque. Ceci relève d'une technique très coûteuse encore peu expérimentée dans les SGBDOO existants. Bien que [Gourhant et al. 92] propose aussi une approche fondée sur l'utilisation d'un glaneur de cellules pour le système Amadeus, seul GemStone [Copeland et Maier 84] dispose à notre connaissance d'un tel outil.

3.3.3.3. Propositions de Jukka Teuhola

Jukka Teuhola, de l'Université de Turku, en Finlande, a récemment proposé plusieurs idées innovantes concernant le regroupement d'objets.

Le premier concept avancé [Teuhola 95a] [Teuhola 95b] est d'utiliser les OID logiques pour transporter des informations concernant le groupement des objets. L'objectif est de permettre l'accès aux objets par des techniques de hachage en générant des OID logiques structurés (*structured surrogates*) pouvant être utilisés comme *pseudoclés* d'une fonction de hachage. Néanmoins, ces contributions se focalisent sur l'accès à des objets regroupés et non sur les critères de regroupement à leur appliquer. De plus, de l'aveu même de l'auteur, ces techniques ne supportent que le regroupement statique.

Une seconde proposition prometteuse consiste à grouper ensemble des objets qui partagent le plus de valeurs d'attributs possibles, c'est-à-dire des objets qui participent à des relations de type *plusieurs à plusieurs* [Teuhola 97]. Comme le problème général du regroupement multi-critères est NP-complet, une heuristique simple et rapide est proposée. Cependant, cette approche reste locale, en ce sens que seuls les objets d'une même classe peuvent être regroupés. Elle gagnerait à être couplée avec d'autres méthodes pour prendre en compte des groupements pouvant intervenir sur des hiérarchies d'héritage ou de composition, par exemple.

Ces deux contributions ouvrent de nouvelles perspectives en matière de regroupement d'objets, mais elles ne sont pas encore abouties et n'ont fait l'objet d'aucune implémentation.

3.4. Synthèse et étude comparative

Dans [Bullat 95] et [Bullat 96] est proposée toute une série de critères permettant de dresser une analyse critique comparative de stratégies de groupement d'objets.

- *Mode de spécification des groupements :* Ce critère indique de quelle manière les consignes de regroupement sont spécifiées. Il influe directement sur la transparence de la stratégie utilisée. Le mode de spécification est dit *dicté* lorsque les actions de regroupement sont indiquées au système par l'utilisateur, *explicite* lorsque que ces spécifications sont codées dans les programmes d'application, *déclaré* lorsqu'elles sont définies dans le schéma de la base ou séparément (dans un schéma de placement) et finalement *interne* lorsque le système dispose de moyens pour déterminer seul quels sont les groupements judicieux.
- *Intervenant extérieur :* Certains modes de spécification des regroupements supposent l'intervention d'un acteur externe au système. Il peut s'agir de l'*utilisateur*, du *développeur d'applications* ou de l'administrateur de la base de données.
- *Mode d'activation des regroupements :* Ce critère indique si le SGBD effectue de lui-même un travail de regroupement systématique (mode d'activation *automatique*) ou s'il est nécessaire de déclencher le groupement (mode d'activation *manuel*).
- *Critères de regroupement :* Les critères servant de base pour effectuer les groupements peuvent être de type *structurels* (regroupement basé sur les relations structurelles entre objets : composition, héritage, référence…), *logiques* (versions, équivalence entre objets…), *de localité* (un objet est regroupé à proximité d'un autre objet, ou dans la page ou le segment indiqués) ou *applicatifs* (statistiques d'exploitation, fréquences d'accès estimées…).
- *Alternativité :* Une stratégie est qualifiée d'*alternative* si elle propose le choix entre plusieurs critères de regroupement.
- *Stratégie multi-critères ou non :* Une stratégie est dite *multi-critères* si elle intègre plusieurs critères à la fois pour établir le regroupement des objets sur le disque.
- *Dynamicité :* Ce critère indique l'automatisation du déclenchement du travail de réorganisation des groupements suite à tout événement pouvant le remettre en cause. Avec une stratégie *dynamique*, le placement des objets est réévalué

immédiatement et modifié si nécessaire. Une stratégie *statique* ne laisse aucune réorganisation à l'initiative du système.

- *Déterminisme :* Ce critère caractérise la façon de résoudre les conflits occasionnés en cas de partage d'objets. Un placement est qualifié de *déterministe* s'il existe une règle apte à lever l'ambiguïté du placement des objets partagés, d'*aléatoire* dans le cas contraire.
- *Région de regroupement :* La taille de la région de regroupement exerce une influence sur les performances potentielles des stratégies de regroupement. La région peut être la *page* disque ou le *segment* (ensemble de pages contiguës).
- *Présence d'un outil d'évaluation :* Un outil d'évaluation permet à l'administrateur de la base de contrôler la pertinence des regroupements existants afin de pouvoir prendre des mesures en vue de les améliorer, si nécessaire.
- *Présence d'un outil de réorganisation :* Un outil de réorganisation permet à l'administrateur de la base de données de réorganiser l'espace de stockage lorsque le placement physique des objets devient inopportun.
- *Paramétrabilité :* Ce critère indique s'il est possible d'agir sur les spécifications de regroupement.

La Table 1.2 récapitule les caractéristiques des stratégies de regroupement que nous venons de présenter, ainsi que celles des prototypes ENCORE [Hornick et Zdonik 87] et EXODUS [Carey et al. 88], que nous n'avons fait que mentionner, en fonction de ces critères.

3.5. Le clustering dans des domaines connexes

L'extraction de données dans les bases de données (*data mining*) peut être définie comme l'exploration et l'analyse de grandes quantités de données afin de découvrir des formes et des règles significatives en utilisant des moyens automatiques ou semi-automatiques. Le but de l'extraction de données est le plus souvent de permettre à une entreprise d'améliorer ses fonctions de soutien au marketing, aux ventes et au service client à travers une meilleure compréhension de ses clients, mais les techniques de *data mining* peuvent également s'appliquer dans des domaines allant de l'action juridique à la radioastronomie, en passant par la médecine et la commande de processus industriels.

	Mode de spécification	Intervenant extérieur	Mode d'activation	Type du / des critère(s)	Alternativité	Multi-critères	Dynamicité	Détermi-nisme	Taille région	Outil d'évaluation	Outil de réorganisation	Paramétrabilité	Implantation
ORION	Dicté	Utilisateur	Manuel	Structurel	Non	Non	Statique	—	Segment	Non	Non	Non	Oui
Cactis	Interne	Aucun	Manuel	Applicatif	Oui	Non	Statique	Choix	Page	Non	Oui	Non	Oui
O_2	Déclaré	Administrateur	Automatique	Structurel	Oui	Non	Statique	Choix	Page	Proposé	Non	Non	Oui
GemStone	Explicite	Développeur	Manuel	De localité	Non	Non	Statique	—	Segment	Non	Non	Non	Oui
ObjectStore	Explicite	Développeur	Manuel	De localité	Non	Non	Statique	—	Segment	Non	Non	Non	Oui
ONTOS	Explicite	Développeur	Manuel	De localité / Structurel	Oui	Non	Statique	—	Variable	Non	Non	Non	Oui
ENCORE	Déclaré	Administrateur	Automatique	Structurel / Applicatif	Oui	Non	Statique	Copie	Segment	Oui	Non	Non	Oui
EXODUS	Dicté	Administrateur	Manuel	De localité	Non	Non	Statique	—	?	Non	Non	Non	Oui
McIver & King	Interne	Aucun	Automatique	Applicatif	Oui	Non	Dynamique	—	Page	Non	Oui	Non	Non
Chang & Katz	Interne	Aucun	Automatique	Structurel / Applicatif	Oui	Non	Dynamique	—	Page	Oui	—	Options	Non
Gay & Gruenwald	Interne	Aucun	Automatique	Applicatif	Oui	Oui	Dynamique	—	Segment	Oui	—	Oui	Non
DSTC	Interne	Aucun	Automatique	Tout type	Oui	Non	Dynamique	Choix	Segment	Non	—	Oui	Oui
Cheng & Hurson	Interne	Aucun	Automatique	Structurel / Logique	Oui	Oui	Dynamique	—	?	Non	—	Non	Non
Tsangaris / Gerlhof	Dicté	Administrateur	Manuel	Structurel	Non	Non	Statique	—	Page	Non	—	Non	Non
EOS	Déclaré	Administrateur	Automatique	Structurel	Oui	Non	Dynamique	Choix	Segment	Non	—	Non	Non

Table 1.2 : Tableau comparatif des stratégies de regroupement étudiées

Dans le domaine de l'extraction de données, une des tâches est l'analyse des *clusters* [Berry et Linoff 97], c'est-à-dire la segmentation d'une population hétérogène en un certain nombre de sous-groupes plus homogènes, ou *clusters*. Les enregistrements sont en fait regroupés en fonction d'une similitude mutuelle. L'analyse des *clusters* est souvent préalable à une autre forme d'exploitation des données. Par exemple, pour une étude de segmentation de marché, au lieu de chercher directement une règle comme « quel est le type de promotion auquel les clients répondent le mieux ? », il est possible de diviser la base de données des clients en *clusters* de personnes ayant les mêmes habitudes de consommation avant de poser la question pour chaque *cluster*.

Le problème du regroupement de données en *data mining* peut se formuler comme suit [Zhang et al. 96]. Étant donné un gros ensemble de points de données multidimensionnels, l'espace des données n'est pas occupé uniformément. L'objectif du groupement est d'identifier les régions presque vides des régions peuplées et de découvrir les schémas de distribution globaux dans l'ensemble de données. De plus, les clusters identifiés peuvent être visualisés plus efficacement que l'ensemble de données complet original. L'algorithme BIRCH [Zhang et al. 96] remplit ces objectifs grâce à une approche par construction (cf. Section 3.1).

Le domaine du traitement d'images rencontre la même problématique qu'en *data mining*, par exemple, pour segmenter des images (l'algorithme BIRCH [Zhang et al. 96] a d'ailleurs été appliqué, entre autres, au filtrage d'images). La segmentation d'image consiste, à partir d'une image, à découvrir quelles sont les principales entités qui la composent en se basant sur leur densité lumineuse. Dans ce cadre, étant donné un ensemble de M points de données $X = \{x_1, x_2, \ldots, x_M\}$, l'objectif du groupement [Wu et Leahy 93] est de partitionner l'ensemble de données en K sous-ensembles non vides tels que les données similaires sont groupées ensemble et que les données groupées dans des sous-ensembles (ou *clusters*) différents ne soient pas similaires. [Wu et Leahy 93] propose une approche issue de la théorie des graphes pour résoudre le problème.

D'autres domaines encore, certains très éloignés de l'informatique, font usage de *clusters*, comme par exemple la « taxonomie numérique », dont l'objectif est de classifier les espèces vivantes en examinant les données concernant un ensemble d'organismes et en regroupant ceux qui sont similaires ensemble [Wiggerts 97].

Chapitre 2
Évaluation des performances des SGBDOO

L'objectif de ce chapitre est tout d'abord d'établir l'intérêt de disposer de moyens d'évaluer les performances pour les SGBDOO. Nous présentons successivement les trois grands types d'approches qui permettent de répondre à cette attente : l'analyse mathématique appliquée aux algorithmes qui régissent les SGBDOO, les banc d'essais, ainsi que l'approche simulation, qui est parfois employée pour estimer *a priori* le comportement et les performances des systèmes. Nous nous livrons pour terminer à une analyse critique de chacune de ces trois approches.

1. De la nécessité d'évaluer les performances

Quelque temps après l'émergence des premiers SGBDOO est apparu le besoin d'outils de mesure de performance pour les gestionnaires d'objets persistants, besoin exprimé par des appels répétés lors de conférences [Atkinson et Maier 90]. L'idée sous-jacente à ces appels était que l'évaluation de performance est un composant essentiel dans le développement de gestionnaires d'objets persistants bien conçus et efficaces. De telles mesures sont indispensables, d'une part, pour valider ou réfuter les suppositions des concepteurs quant au comportement réel des SGBDOO et de la charge de travail qu'ils supportent et, d'autre part, pour disposer d'éléments de comparaison en ce qui concerne l'efficacité réelle de différentes technologies. Il existe donc deux raisons bien distinctes pour mesurer les performances d'un SGBDOO :

- établir quel est le comportement d'un produit afin qu'il puisse être comparé à d'autres par des utilisateurs potentiels ;
- mieux comprendre les mécanismes qui régissent le système afin d'éventuellement améliorer sa conception.

Pour cela, [Atkinson et al. 92] propose quatre critères permettant de juger de la qualité d'une méthode d'évaluation de performance.

- *Pertinence :* Une mesure est dite pertinente si elle concerne des aspects de la performance qui s'adressent au plus grand nombre d'utilisateurs potentiels et qui

sont déterminés en appliquant une charge donnée au système. Par exemple, la mesure de performance par excellence est le temps de réponse. Les charges typiques sont un mélange d'opérations sur des structures de données complexes et de grande taille, dont l'objectif est d'être représentatives de charges réelles.

- *Fiabilité :* Pour être fiable, une mesure doit être telle qu'elle puisse être répétée avec le même résultat par un tiers indépendant. La mesure doit donc être spécifiée sans ambiguïté afin que ses paramètres ne puissent pas être interprétés de façon différente dans différentes conditions.
- *Faisabilité :* Une mesure ne doit pas requérir de ressources trop importantes, sans quoi elle ne sera pas utilisée. Elle peut par contre être répétée afin d'obtenir des résultats plus sûrs, tant que le temps requis par les tests reste raisonnable.
- *Indépendance :* Une mesure doit pouvoir être réutilisable pour évaluer les performances de plusieurs SGBDOO, et pouvoir être activée par tous les langages de bases de données possibles. La mesure doit donc être spécifiée de façon indépendante de tout langage existant afin de ne pas exploiter ses éventuelles spécificités.

Il est important de noter que ces quatre critères sont en conflit mutuel. En effet, la taille et la complexité d'une charge pertinente entre en conflit avec sa faisabilité. De même, la précision requise par la fiabilité peut entrer en conflit avec les impératifs d'indépendance. Il est donc nécessaire de trouver des compromis.

Par ailleurs, plusieurs auteurs [Atkinson et al. 92] [Banerjee et Gardner 95] [Kempe et al. 95] insistent sur le fait que les fonctionnalités offertes par un SGBDOO sont au moins aussi importantes que ses performances brutes et que cet aspect est peu pris en compte par les outils d'évaluation de performance existants.

2. Analyse mathématique

Deux types de méthodes analytiques peuvent être utilisés pour évaluer la qualité d'un regroupement d'objets.

- Les *modèles de coût* sont employés pour apprécier *a priori* la pertinence d'un groupement dans le cadre du choix de ce groupement ou pour déterminer le coût d'accès à des objets regroupés, dans le cas de l'optimisation de requête.
- Les *études de complexité* sont employées pour calculer la complexité d'un algorithme de groupement *a posteriori*, ce qui permet en général de le comparer à d'autres algorithmes.

Dans le domaine des bases de données relationnelles, les travaux d'analyse mathématique sont nombreux et validés. Ils sont moins abondants en ce qui concerne les

bases de données orientées objet. Nous présentons dans cette section deux exemples de modèles de coût et un exemple représentatif d'étude de complexité, afin d'illustrer l'usage de ces méthodes mathématiques.

2.1. Modèles de coût

2.1.1. Modèle d'évaluation des stratégies de groupement dans O_2

Le groupement des objets dans le système O_2 [Deux et al. 91] est déterminé par des arbres de placements déclarés par l'administrateur de la base. Ces arbres spécifient les classes dont les instances doivent être groupées ensemble sur le disque (cf. Chapitre 1, Section 3.2.3).

Pour aider l'administrateur de la base de données dans sa tâche, un modèle de coût a été développé afin de déterminer l'efficacité de divers arbres de regroupement possibles [Benzaken 90b]. Ce modèle de coût se base sur les hypothèses simplificatrices suivantes.

- L'espace disque est infini. En d'autres termes, pour tout arbre de placement $PT(c)$ de la classe c, toutes les instances de c peuvent être stockées sur disque selon cet arbre de placement.
- Aucun index n'est pris en considération. Autrement dit, lorsqu'une méthode doit accéder à un composant o' d'un objet o, elle doit accéder à tous les composants du chemin $o, o_1, \ldots, o_i, \ldots, o'$.
- Les paramètres éventuels des méthodes ne sont pas pris en compte. L'ensemble des méthodes associées à une classe est donc considéré comme représentatif de l'utilisation de cette classe.

Les notations suivantes permettent de définir une fonction de coût pour l'arbre de placement $PT(c)$ de c. Soient :

- $m_1, m_2, \ldots, m_l$ les méthodes associées à c ;
- $p_1, p_2, \ldots, p_l$ le nombre de fois où elles sont invoquées, respectivement (leur fréquence) ;
- $t_1, \ldots, t_n, \ldots, t_p$ les types qui composent c ;
- $coût(i,j)$ le coût associé à la méthode m_i et au type t_j ;
- $coût(i,j) = \alpha$ si, pour la méthode m_i, le composant $o(t_j)$ a été chargé en mémoire et utilisé. α est un profit, donc $\alpha > 0$;
- $coût(i,j) = \beta$ si, pour la méthode m_i, le composant $o(t_j)$ a été chargé en mémoire, mais pas utilisé. Dans ce cas, il y a gaspillage d'espace mémoire. β est une perte, donc $\beta < 0$;

- *coût*(*i*,*j*) = γ si, pour la méthode m_i, le composant $o(t_j)$ n'a pas été chargé en mémoire et n'a pas été nécessité. γ est un profit, donc $\gamma > 0$;
- *coût*(*i*,*j*) = δ si, pour la méthode m_i, le composant $o(t_j)$ n'a pas été chargé en mémoire, mais était nécessité. Dans ce cas, il y a faute d'objet. δ est une perte, donc $\delta < 0$.

Soit *PT*(*c*) = (*V*, *E*) l'arbre de placement associé à *c*. *V* est l'ensemble des nœuds de l'arbre, *E* l'ensemble des arcs. Soit la matrice :

$$M_{l,1} = (p_1, p_2, \ldots, p_l)\ (coût(i,j)).$$

- Si $t_j \in V$ et que m_i utilise t_j, alors *coût*(*i*,*j*) = α.
- Si $t_j \in V$ et que m_i n'utilise pas t_j, alors *coût*(*i*,*j*) = β.
- Si $t_j \notin V$ et que m_i n'utilise pas t_j, alors *coût*(*i*,*j*) = γ.
- Si $t_j \notin V$ et que m_i utilise t_j, alors *coût*(*i*,*j*) = δ.

La fonction de coût *F*(*PT*(*c*)) associée à *PT*(*c*) est obtenue en additionnant les coefficients de la matrice $M_{l,1}$. Elle reflète les fautes d'objets et les surcharges en mémoire primaire.

$$F(PT(c)) = \sum_{i=1}^{i=l} \sum_{j=1}^{j=n} coût(i,j)\, p_i$$

$$F(PT(c)) = \alpha \sum_{i=1}^{i=l} p_i Nbr(j; coût(i,j) = \alpha) + \beta \sum_{i=1}^{i=l} p_i Nbr(j; coût(i,j) = \beta) +$$

$$\gamma \sum_{i=1}^{i=l} p_i Nbr(j; coût(i,j) = \gamma) + \delta \sum_{i=1}^{i=l} p_i Nbr(j; coût(i,j) = \delta)$$

Nbr(*j*; *coût*(*i*,*j*) = $\aleph$) correspond au nombre de fois où *coût*(*i*,*j*) est égal à $\aleph$. L'objectif est de trouver un arbre de placement qui maximise la fonction de coût *F*(*PT*(*c*)). Soient :

- $USED = \sum_{i=1}^{i=l} p_i Nbr(j; coût(i,j) = \alpha)$ le nombre d'objets utilisés par toutes les méthodes de *c* ;
- $WASTE = \sum_{i=1}^{i=l} p_i Nbr(j; coût(i,j) = \beta)$ le nombre d'objets non nécessités chargés en mémoire ;
- $UN\_USED = \sum_{i=1}^{i=l} p_i Nbr(j; coût(i,j) = \gamma)$ le nombre d'objets qui n'ont été ni chargés en mémoire ni utilisés ;
- $DEFAULT = \sum_{i=1}^{i=l} p_i Nbr(j; coût(i,j) = \delta)$ le nombre de fautes d'objet.

Supposons qu'en utilisant cet arbre de placement, *k* méthodes parmi les *l* provoquent une faute d'objet. Soient p_{i1}, …, p_{ik} leurs fréquences associées. Si *k* méthodes parmi *l* provoquent une faute d'objet, cela signifie que l'arbre associé n'est pas une couverture minimale. Il est intéressant de savoir si l'arbre de couverture minimal *PT*°(*c*) (qui élimine les fautes d'objet) est meilleur qu'un arbre qui n'est pas une couverture minimale. Les coûts de *PT*(*c*) et *PT*°(*c*) sont donc comparés. Les objets *UN_USED* sont chargés et associés à la perte des objets *WASTE*. Les objets *DEFAULT* sont chargés et associés au profit des objets *USED*. La fonction de coût associée à *PT*°(*c*) est donc :

$$F(PT°(c)) = \alpha(USED + DEFAULT) + \beta(WASTE + UN\_USED).$$

Pour déterminer lequel de *PT*(*c*) ou de *PT*°(*c*) est le meilleur candidat, la différence entre leurs fonctions de coût associées est calculée :

$$F(PT(c)) - F(PT°(c)) = (\delta - \alpha)DEFAULT + (\gamma - \beta)WASTE.$$

PT°(*c*) est le meilleur candidat si et seulement si :

$$(\delta - \alpha)DEFAULT + (\gamma - \beta)WASTE \leq 0$$

$$\frac{DEFAULT}{WASTE} \geq \frac{(\gamma - \beta)}{(\delta - \alpha)}.$$

DEFAULT et *WASTE* dépendent des fréquences d'appel aux méthodes associées à *c*. Cette inéquation indique que si les fautes d'objet correspondent aux méthodes avec une basse fréquence, il est plus avantageux d'utiliser un arbre de placement qui n'est pas une couverture minimale et qui permettra un gain d'espace mémoire. Les coefficients α, β, γ, δ dépendent des caractéristiques du système. Ils restent à fixer.

Cette fonction de coût ne prend pas en compte l'utilisation des ensembles (*set*). Il est supposé que, soit tous les éléments sont chargés en mémoire en un unique accès disque (si l'arbre de placement le permet), soit aucun d'entre eux n'est chargé. Dans tous les cas, un seul objet est pris en compte pour le calcul de la fonction, ce qui n'est pas conforme à la réalité. Pour pallier ce problème, [Benzaken 90b] raffine encore la fonction de coût élémentaire *coût*(*i*,*j*).

2.1.2. Modèle de coût générique pour les bases de données regroupées

Dans le cadre d'une étude portant sur l'optimisation de requêtes, [Gardarin et al. 95] a proposé un modèle de coût générique pour les SGBDOO. Ce modèle de coût prend explicitement en compte le groupement d'objets. L'objectif est d'estimer le nombre d'entrées-sorties pour accéder à une collection d'objets groupés. Ce modèle de coût a été validé par des expériences menées avec le système O_2 [Deux et al. 91].

2.1.2.1. Paramètres

Le modèle de coût utilise des statistiques concernant les composants de la base de données, afin d'estimer le coût d'un plan d'exécution. Ces statistiques renferment des informations concernant les collections d'objets au sein de la base. Elles sont définies comme suit.

- $\|C\|$: cardinalité de la collection *C*.
- $\|C\|_{Cli}$: cardinalité de la collection *C* dans le *cluster i*.
- $|C|_{Cli}$: pages de la collection *C* appartenant au *cluster i*.
- S_c : taille moyenne des objets dans la collection *C*.
- $D_{c1,\,c2}$: nombre de références distinctes d'un objet *C1* vers des objets *C2*.
- $X_{c1,\,c2}$: nombre d'objets *C1* ayant une référence à NULL vers des objets *C2*.
- $Z_{c1,\,c2}$: nombre moyen de références distinctes vers des objets *C2* pour les objets *C1* ayant au moins une référence non NULL.

$$Z_{c1,\,c2} = \frac{D_{C1,\,C2} \cdot \|C_1\|}{\|C_1\| - X_{C1,\,C2}}$$

- S_p : taille des pages disque.

2.1.2.2. Estimation de nombre de pages occupées par une collection groupée

Nous présentons ici la méthode employée par [Gardarin et al. 95] pour estimer le nombre de pages occupées par chaque partition d'une collection *A* lorsque celle-ci est groupée avec d'autres collections.

Dans un système de stockage où il n'y a pas de groupement d'objets, il y a $\left\lfloor \frac{S_p}{S_A} \right\rfloor$ objets dans une page et le nombre total de pages occupées par la collection est :

$$|A| = \frac{\|A\|}{\left\lfloor \frac{S_p}{S_A} \right\rfloor}.$$

Cependant, dans un système où les objets de différentes collections peuvent être stockés ensemble, le nombre de pages occupées ne peut pas être calculé de cette façon, car une collection peut se retrouver disséminée sur plusieurs partitions. Une autre méthode est

proposée pour estimer le nombre total de pages occupées par une collection après regroupement.

Hypothèses

- Les objets ne sont pas dupliqués.
- La cardinalité d'une collection ne varie pas après regroupement.
- Supposons que la collection *B* est groupée avec la collection *A*. La taille moyenne des objets de la collection *A* (respectivement *B*) S_A (respectivement S_B) est inférieure à la taille d'une page disque.

Le nombre de pages occupées par la collection *A* est estimé en premier lieu car elle est considérée comme la racine de l'arbre de placement (cf. Chapitre 1, Section 3.2.3). Après regroupement, la collection *A* est placée dans deux partitions physiques : une partition qui contient aussi des objets de *B*, $Cl_{A \to B}$ et une partition Cl_A qui ne contient que des objets de *A*.

- Pour Cl_A, le nombre de pages de la collection A est $\|A\|_{Cl_A} = X_{A,B}$. Alors :

$$|A|_{Cl_A} = \frac{X_{A,B}}{\left\lfloor \frac{S_p}{S_A} \right\rfloor}.$$

- Pour $Cl_{A \to B}$, le nombre d'objets racines dans le *cluster* est $\|A\| - X_{A,B}$. La taille du *cluster* est $S_{Cl_{A \to B}} = S_A + (Z_{A,B} \cdot S_B)$. Alors :

$$|A|_{Cl_{A \to B}} = \begin{cases} \dfrac{\|A\| - X_{A,B}}{\left\lfloor \dfrac{S_p}{S_{Cl_{A \to B}}} \right\rfloor} \text{ si } S_{Cl_{A \to B}} < S_p \\ \|A\| - X_{A,B} \text{ si } S_{Cl_{A \to B}} \geq S_p \end{cases}.$$

Le nombre total de pages est : $|A| = |A|_{Cl_A} + |A|_{Cl_{A \to B}}$.

Dans un second temps, le nombre de pages occupées par la collection *B* est estimé. Les deux partitions où se trouvent des objets de *B* sont $Cl_{A \to B}$ et Cl_B.

- Pour Cl_B, le nombre d'objets de *B* qui ne sont pas référencés par des objets de *A* est $\|B\| - (\|A\| \cdot D_{A,B})$. Alors :

$$|B|_{Cl_B} = \frac{\|B\| - (\|A\| \cdot D_{A,B})}{\left\lfloor \frac{S_p}{S_B} \right\rfloor}.$$

- Pour $Cl_{A \to B}$, la taille et le nombre d'objets par *cluster* restent identiques, mais le nombre de *clusters* à accéder est dorénavant $X'_{Cl_{A \to B}} = Min(Z_{A,B} \cdot X_{A,B}, X_{A,B})$. Alors :

$$|B|_{Cl_{A \to B}} = \begin{cases} |A|_{Cl_{A \to B}} \text{ si } S_{Cl_{A \to B}} < S_p \\ \|A\| - X_{A,B} \text{ si } Z_{A,B} \cdot S_B \le S_p \\ (\|A\| - X_{A,B}) \cdot \left\lfloor \dfrac{Z_{A,B} \cdot S_B}{S_p} \right\rfloor \text{si } Z_{A,B} \cdot S_B > S_p \end{cases} .$$

Le nombre total de pages est donc : $|B| = |B|_{Cl_B} + |B|_{Cl_{A \to B}}$.

La même méthode de calcul est utilisée pour les techniques de groupements conjonctives, disjonctives ou des combinaisons des deux. Ainsi, dans le cas d'un groupement plus élaboré, la complexité des formules augmente avec la forme de l'arbre de placement.

2.2. Exemple d'étude de complexité

Afin de comparer les performances de deux techniques de groupement : celle employée dans le système Cactis [Hudson et King 89] et la méthode proposée par Chang et Katz [Chang et Katz 89] (cf. Chapitre 1, Section 3) ; Sophie Chabridon a effectué au sein de l'Université d'Oklahoma une étude de complexité portant d'une part sur l'espace mémoire nécessaire à chacune des techniques pour gérer leurs statistiques et, d'autre part, sur la complexité en temps d'exécution des algorithmes de groupement [Chabridon et al. 92] [Chabridon et al. 93].

2.2.1. Espace mémoire requis

L'algorithme de groupement de Cactis fait usage de deux types de compteurs. La taille de ces compteurs est supposée de deux octets. Le premier compteur enregistre, pour chaque objet, le nombre d'accès à cet objet. Si N est le nombre d'objets dans la base, la taille requise pour ces compteurs est $2N$ octets. Le second type de compteurs est chargé de mémoriser le nombre de fois où chaque référence inter-objets est utilisée. Le nombre maximum de références possibles est N^2. Cependant, il est très probable que le nombre de références R soit très inférieur à N^2, chaque objet de la base n'étant normalement pas relié à tous les autres. La taille requise pour ces compteurs est $2R$ octets. Au total, l'espace mémoire requis est donc de $2 (N + R)$ octets.

L'algorithme de Chang et Katz fait usage d'un compteur d'un octet pour chaque attribut de chaque objet de la base. Soient nA le nombre moyen d'attributs par objet et N le

nombre total d'objets dans la base. Au total, l'espace mémoire requis est donc de $nA * N$ octets.

Avec une petite base de 100 objets (N = 100) et une moyenne de 20 attributs par objet (nA = 20), l'algorithme de Chang et Katz nécessite nA x N = 20 x 100 = 2000 octets. L'algorithme de Cactis nécessite dans le pire des cas, si $R=N^2$, 2 (N + R) = 2 (100 + 10000) = 20200 octets, soit dix fois plus que l'algorithme de Chang et Katz. [Chabridon et al. 93] estime cependant que dans le cas le plus favorable, l'algorithme de Cactis n'utilise pas plus d'espace mémoire que l'algorithme de Chang et Katz (environ 2 Ko).

2.2.2. Temps d'exécution

Dans l'algorithme de regroupement de Cactis (cf. Chapitre 1, Section 3.2.2), l'objet le plus référencé dans la base de données doit tout d'abord être détecté. Si la base contient N objets, il faut donc lire les N compteurs associés à l'accès aux objets individuels, ce qui nécessite O(N) opérations. Un autre procédé pourrait être de trier les objets par ordre de nombre de références décroissant, ce qui prendrait avec un algorithme de tri efficace O(Nlog$N$) opérations. Ensuite, pour l'objet sélectionné, la référence la plus utilisée doit être détectée. Si aucune structure de donnée appropriée n'est utilisée, toutes les références doivent être examinées (O(R) opérations). Finalement, dans le pire des cas, la boucle principale de l'algorithme parcourt tous les objets de la base, ce qui donne une complexité maximale en O(N.(N+R)). Si le nombre R de relations est assimilé à son maximum N^2, la complexité de l'algorithme devient en O(N^3), ce qui le rend très gourmand en temps.

Comme nous avons seulement énoncé les principes de l'algorithme de Chang et Katz (qui est compliqué) au Chapitre 1, nous livrons seulement les conclusions de [Chabridon et al. 93] quant à sa complexité. Soient P_{inh} la probabilité qu'un attribut soit hérité par héritage d'instance à instance (cf. Chapitre 1, Section 3.3.1.2), nA le nombre moyen d'attributs par objet, P_{cand} la probabilité qu'une page disque soit sélectionnée comme page d'accueil pour des objets groupés par l'algorithme et P le nombre de pages occupées par la base de données. Alors la complexité de l'algorithme de Chang et Katz est O($P.P_{cand}.nA.P_{inh}$).

Pour réorganiser la petite base de données citée en exemple dans la Section 2.2.1, l'algorithme de regroupement de Cactis requiert 11000 opérations au minimum et 1010000 opérations au maximum (si le nombre de relations inter-objets est très élevé). Par contre, l'algorithme proposé par Chang et Katz ne nécessite lui que 2000 opérations au maximum.

2.2.3. Éléments qualitatifs

Finalement, il faut ajouter que [Chabridon et al. 93] aborde, parallèlement aux études de complexité, des critères de comparaison à caractère qualitatif, comme par exemple, la taille de la région de regroupement (page ou segment), la nature de la stratégie (statique ou

dynamique) et la nécessité ou non que l'utilisateur donne des indications pour déterminer les groupements. Ceci répond au souci de ne pas prendre en compte que des éléments de performance pure lors du choix d'un algorithme ou d'un autre.

2.2.4. Autres études de complexité

Les auteurs qui proposent des algorithmes de groupement d'objets basés sur des heuristiques existantes de la théorie des graphes [Tsangaris et Naughton 91] [Gerlhof et al. 93] [Wierzyk 96] (cf. Chapitre 1, Section 3.3.3.1) comparent leurs propositions par la complexité. Comme leurs algorithmes se basent sur des heuristiques déjà connues, dont la complexité est clairement établie, la complexité des nouveaux algorithmes se déduit directement. À titre indicatif, la proposition de [Tsangaris et Naughton 91] est de complexité d'exécution $O(N^{2,4})$, celle de [Gerlhof et al. 93] en $O(R\log R)$ et celle de [Wierzyk 96] en $O(N)$, avec N le nombre d'objets dans la base de données et R le nombre de relations inter-objets (le nombre d'arcs dans le graphe d'objets).

2.3. Synthèse

L'intérêt de ces approches est qu'elles induisent une meilleure connaissance du problème étudié, grâce à son analyse détaillée. Cela permet de déterminer immédiatement les paramètres les plus influents. De plus, ces techniques d'analyse mathématique, notamment les modèles de coût, sont faciles à mettre en œuvre. Les calculs qu'elles induisent sont rapides et reproductibles, ce qui satisfait le critère de fiabilité.

Néanmoins, ces approches présentent également un certain nombre d'inconvénients.

- Des hypothèses simplificatrices sont effectuées dans le but de rendre l'analyse mathématique possible, ce qui a pour inconvénient de donner un modèle de comportement (qui est déjà une abstraction) qui diverge plus ou moins fortement par rapport à la réalité, surtout si les simplifications introduites sont importantes. La pertinence des modèles mathématiques en souffre.
- Ces analyses mathématiques utilisent des paramètres dont la valeur n'est pas aisée à estimer dans un SGBDOO réel (statistiques non fournies en standard, coûts d'opérations difficiles à apprécier…), ce qui peut rendre les formules obtenues impossibles à calculer numériquement. C'est cette fois le critère de faisabilité qui est mis en cause.
- Les trois analyses mathématiques que nous présentons sont très focalisées sur un type de problème (le groupement d'objets, en l'occurrence). Elles sont de surcroît fortement dépendantes du système qu'elles modélisent (les arbres de placement dans O_2, en ce qui concerne nos exemples de modèles de coût). Ces études sont donc nettement ciblées, ce qui va à l'encontre du critère d'indépendance.

3. Bancs d'essais

3.1. Qu'est-ce qu'un banc d'essais ?

Utiliser un banc d'essais (*benchmark*, dans la terminologie anglo-saxonne) consiste à effectuer une série de tests sur un SGBD existant dans le but d'estimer ses performances dans des conditions données. Les bancs d'essais sont généralement utilisés pour comparer les performances globales des SGBD, mais aussi pour illustrer les avantages d'un système ou d'un autre dans une situation donnée, ou pour déterminer une configuration matérielle optimale (taille du cache mémoire, nombre de disques, etc.) pour un SGBD et/ou une application donnés.

Typiquement, un banc d'essais est composé de deux éléments principaux :

- une base de données (schéma conceptuel et méthode de génération) ;
- une charge, c'est-à-dire un ensemble d'opérations à effectuer sur la base de données (par exemple, différentes sortes de requêtes) et un protocole détaillant de déroulement de l'exécution de ces opérations.

[Gray 93] définit quatre critères primordiaux permettant de spécifier un bon banc d'essais. Un banc d'essais doit être *pertinent*, *portable*, *simple* et *adaptable* (*scalable* en anglais). Les trois premiers critères sont très similaires, respectivement, aux qualités de pertinence, d'indépendance et de faisabilité évoquées dans la Section 1. L'adaptabilité, par contre, qui consiste pour un banc d'essais à pouvoir s'adapter à de petits et de gros systèmes informatiques, voire à de nouvelles architectures (parallèles, par exemple), est en contradiction avec la notion de fiabilité. Nous l'utiliserons néanmoins pour juger des qualités des bancs d'essais que nous présentons, car elle leur est véritablement adaptée.

Dans le monde des SGBD relationnels, le *Transaction Processing Council* (TPC) joue un rôle prépondérant en matière de bancs d'essais. Cet institut à but non lucratif, fondé en 1988, a pour mission de définir des bancs d'essais standards, de vérifier leur application correcte par les compagnies qui souhaitent voir leur produit testé et de publier régulièrement les résultats de ces tests de performance. Le TPC a spécifié quatre bancs d'essais principalement basés sur des applications commerciales et bancaires (opérations de débits/crédits sur des comptes). Les derniers en date sont appelés TPC-C [TPC 96] et TPC-D [TPC 95].

En revanche, il n'existe pas de banc d'essais standard pour les SGBDOO, même si les plus souvent cités et utilisés, OO1 [Cattell 91], HyperModel [Anderson et al. 90] et OO7 [Carey et al. 93], font office de standard de fait. Ces bancs d'essais sont plutôt ciblés sur les applications d'ingénierie (CAO, PAO, AGL…), mais restent assez généraux dans l'esprit. C'est pourquoi, parallèlement, ont été développés plusieurs bancs d'essais dédiés à l'étude de domaines particuliers, comme le regroupement d'objets [Bancilhon et al. 92] [Gerlhof et al. 96] [Bullat 96], les bases de données actives [Geppert et al. 94]

[Zimmermann et Buchmann 95], les systèmes relationnels-objets [Carey et al. 97], la gestion de charge (*workflow management*) [Bonner et al. 95], les applications de CAO [Kempe et al. 95] ou les AGL [Emmerich et Kampmann 93], l'étude des vues dans un contexte orienté objet [Kuno et Rundensteiner 95].

Notre objectif n'est pas de présenter ici tous ces bancs d'essais. Nous nous contentons des plus courants, à savoir OO1, HyperModel et OO7, de quelques dérivés, ainsi que du banc d'essais Justitia [Schreiber 94], qui est intéressant pour son approche générique multi-utilisateurs. Pour plus de précisions, le lecteur intéressé pourra se référer à l'état de l'art établi par [Halloran et Roth 93], qui fait mention des bancs d'essais les plus utilisés pour les SGBD relationnels et orientés objet, à la bibliographie annotée de [Chaudhri 95], qui est, elle, spécifique aux bancs d'essais pour SGBDOO, ou à la FAQ (*Frequently Asked Questions*) du *newsgroup* comp.benchmark [Sill 95].

3.2. OO1

OO1 (*Object Operations 1*) est né au début des années 90, alors qu'il n'existait pas de banc d'essais approprié pour évaluer les performances des applications d'ingénierie (CAO et AGL, principalement), qui diffèrent grandement des applications de bases de données traditionnelles, pour lesquelles les bancs d'essais du TPC sont adaptés [Cattell 91]. OO1, qui est également appelé « the Cattell benchmark », a été conçu sur la base d'entretiens avec des concepteurs d'applications CAO et d'AGL et de retours concernant un banc d'essais antérieur : le banc d'essais Sun [Rubenstein et al. 87]. Malgré son orientation vers les applications d'ingénierie, OO1 a été simplifié par rapport au banc d'essais Sun afin d'être plus générique [Cattell et Skeen 92]. OO1 a d'ailleurs été employé pour évaluer les performances de nombreux SGBD relationnels et orientés objet.

3.2.1. Base d'objets d'OO1

La base de données d'OO1 est présentée dans la Figure 2.1 par un diagramme statique de structure UML. Elle est basée sur deux classes : composant (*Part*) et connexion (*Connection*). Les composants sont les éléments d'une hiérarchie composite dans laquelle chaque composant est relié à trois autres, grâce à un objet connexion. Chaque connexion référence deux composants : le composant source (*From*) et le composant destination (*To*). La valeur des attributs *X*, *Y* et *Length* est aléatoirement distribuée dans l'intervalle [0, 99999]. Celle des attributs *Type* est sélectionnée aléatoirement parmi les chaînes de caractères {« part-type0 »…« part-type9 »}. La valeur de la date de construction (*Build*) est aléatoirement distribuée dans un intervalle de dix ans.

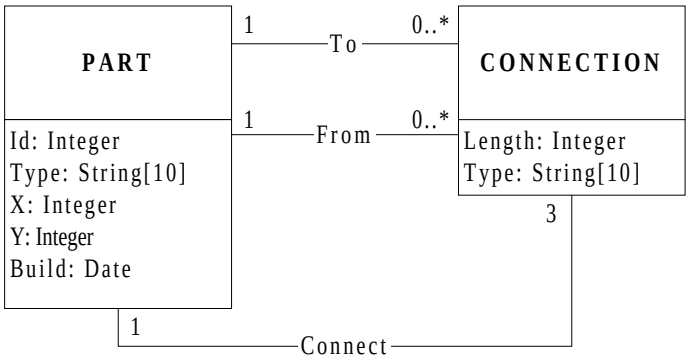

Figure 2.1 : Schéma de la base de données d'OO1

La base est générée de la façon suivante :

1) tous les composants sont créés et stockés dans un dictionnaire ;
2) pour chaque composant, trois autres composants sont sélectionnés au hasard et reliés au premier composant grâce à un objet connexion.

Une certaine localité de référence, indiquant que des objets sont souvent reliés à d'autres objets relativement proches, est assurée grâce à la définition d'une zone de référence *RefZone*. En effet, le composant d'indentifiant (*Id*) *i* est lié à des composants dont les identifiants appartiennent à l'intervalle [*i* - *RefZone*, *i* + *RefZone*]. La probabilité que les liens soient déterminés de cette façon est de 0,9. Si ce n'est pas le cas, les composants liés sont choisis totalement au hasard.

Une base OO1 typique est constituée de 20000 composants (et donc de 60000 connexions). Sa taille est d'environ 4 Mo. La base peut cependant être étendue si besoin est à une taille dite « grande » (200000 composants, 40 Mo), voire « très grande » (2000000 composants, 400 Mo).

3.2.2. Charge d'OO1

OO1 comprend trois types d'opérations. Chacun d'eux est exécuté dix fois par un utilisateur unique. Le temps de réponse est mesuré à chaque exécution. La première exécution est considérée comme effectuée « à froid », et les autres « à chaud ».

- Recherche (*Lookup*) : Accès à 1000 composants sélectionnés au hasard.
- Parcours (*Traversal*) : Sélection aléatoire d'un composant racine, puis exploration de l'arbre des composants correspondant (en profondeur d'abord) à travers les références *Connect* et *To*, jusqu'à une profondeur de sept (ce qui donne un total de 3280 composants accédés, avec des doubles possibles). En inversant les références destination (*To*) et source (*From*), un parcours à rebours (*reverse traversal*) peut également être exécuté.

- Insertion (*Insert*) : Ajout de 100 composants dans la base, ainsi que des connexions associées. Validation (*commit*) des modifications.

3.2.3. Banc d'essais dérivé d'OO1 : DSTC-CluB

DSTC-CluB (*DSTC Clustering Benchmark*) a uniquement retenu d'OO1 les aspects qui convenaient à l'évaluation de la stratégie de regroupement d'objets DSTC [Bullat et Schneider 96]. Ces aspects concernent essentiellement le schéma conceptuel et les opérations de parcours (*traversal*).

[Bullat 96] introduit en supplément le concept d'*utilisation partagée* qui permet de modéliser de façon fine le problème du partage des objets entre plusieurs *types d'utilisation*. Ceci permet d'évaluer les gains réalisés pour les types de parcours les plus fréquents, ainsi que les pertes engendrées dans les contextes opposés, moins fréquents. Un type d'utilisation représente une façon de parcourir un ensemble d'objets donné, c'est-à-dire l'exécution d'une opération de parcours. L'idée est de parcourir la base de données x fois selon un type d'utilisation donné, à partir d'un composant racine r_i, puis y fois selon un autre type d'utilisation, à partir d'un composant racine r_j voisin de r_i, et ainsi de suite. L'objectif est que :

- certains liens soient parcourus plus souvent que d'autres ;
- plusieurs objets soient partagés dans leur utilisation (qu'ils soient accédés tantôt par un chemin au sein d'un type d'utilisation donné, tantôt par un autre chemin au sein d'un autre type d'utilisation) ;
- parmi ces objets partagés, certains possèdent un chemin d'accès privilégié (un type d'utilisation plus souvent employé que les autres).

La Figure 2.2 illustre une telle situation de partage dans la façon d'utiliser les objets. Plus les racines sont choisies proches les unes des autres, plus la probabilité que les parcours accèdent à des objets communs est élevée. De même, plus la zone de référence utilisée pour la construction de la base (cf. Section 3.2.1) est petite, plus le parcours se resserre sur une zone géographique de la base de données et plus la probabilité d'avoir des objets partagés est élevée. Chaque type d'utilisation est donc caractérisé par l'identificateur du composant racine de navigation, la profondeur du parcours et le nombre de fois où le parcours est exécuté.

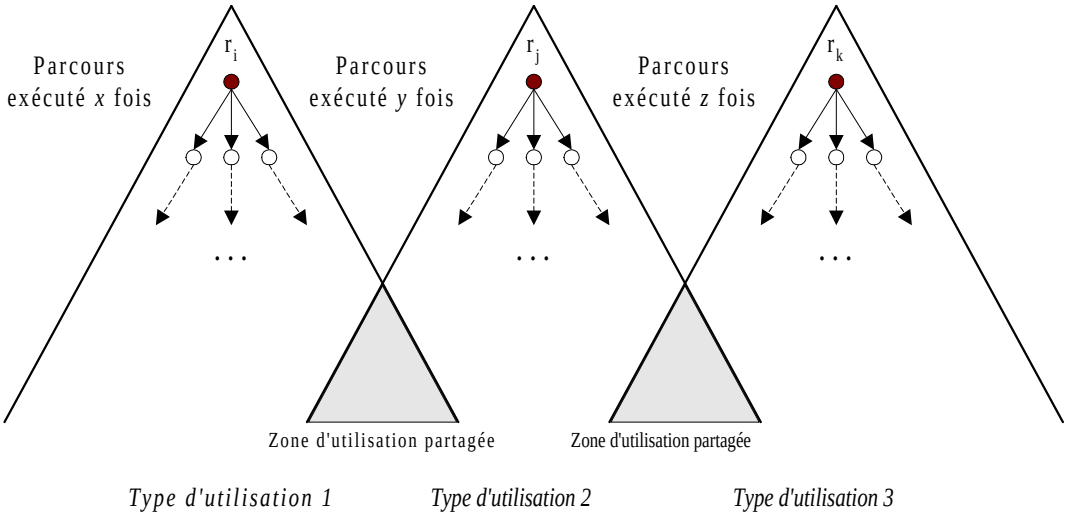

Figure 2.2 : Types d'utilisation avec objets partagés dans DSTC-CluB

3.2.4. Critique d'OO1

OO1 est un banc d'essais simple et, par conséquent, très facile à implémenter. Il remplit parfaitement le critère de faisabilité. Cela l'a rendu très populaire auprès des fabricants de SGBD, qui l'ont intronisé standard de fait pendant très longtemps. OO1 a été utilisé pour évaluer les performances d'une large gamme de systèmes, y compris des SGBDOO, des SGBD relationnels et d'autres systèmes comme INDEX, de Sun (qui est basé sur l'utilisation de B-arbres). OO1 satisfait donc les critères de faisabilité et de portabilité. Les trois tailles de base proposées par OO1 sont, elles, trop limitées pour satisfaire pleinement le critère d'adaptabilité.

Par ailleurs, le modèle de données d'OO1 est trop élémentaire pour mesurer les parcours élaborés qui sont communs à de nombreuses applications orientées objet, dont les applications d'ingénierie. De plus, OO1 ne propose que des opérations simples de navigation et de mise à jour et sa notion d'objet complexe est très limitée (une seule hiérarchie composite), ce qui contrevient fortement au critère de pertinence.

3.3. HyperModel

Le banc d'essais HyperModel (aussi appelé le banc d'essais Tektronix dans la littérature) [Anderson et al. 90] [Berre et Anderson 91] offre une base de données plus complexe que celle d'OO1. De plus, il est reconnu pour la richesse des tests qu'il propose. En effet, ses auteurs considèrent que pour être représentatif d'une grande variété d'applications d'ingénierie, un banc d'essais doit mesurer l'effet d'un nombre suffisant d'opérations exécutées sur une base de données suffisamment complexe.

3.3.1. Base d'objets d'HyperModel

Le banc d'essais HyperModel est basé sur un modèle hypertexte étendu. Le modèle hypertexte est une structure générique de graphe constituée de nœuds (*Nodes*) et de liens. Son schéma est présenté en Figure 2.3 sous la forme d'un diagramme statique de structure UML. La caractéristique principale de cette base de données est le nombre de relations variées existant entre les classes :

- *spécialisation :* les attributs d'un objet *Node* peuvent être hérités d'un autre objet *Node* ;
- *agrégation :* une instance de la classe *Node* peut être composée d'une ou de plusieurs autres instances ;
- *association :* deux objets *Node* peuvent être reliés par un lien orienté.

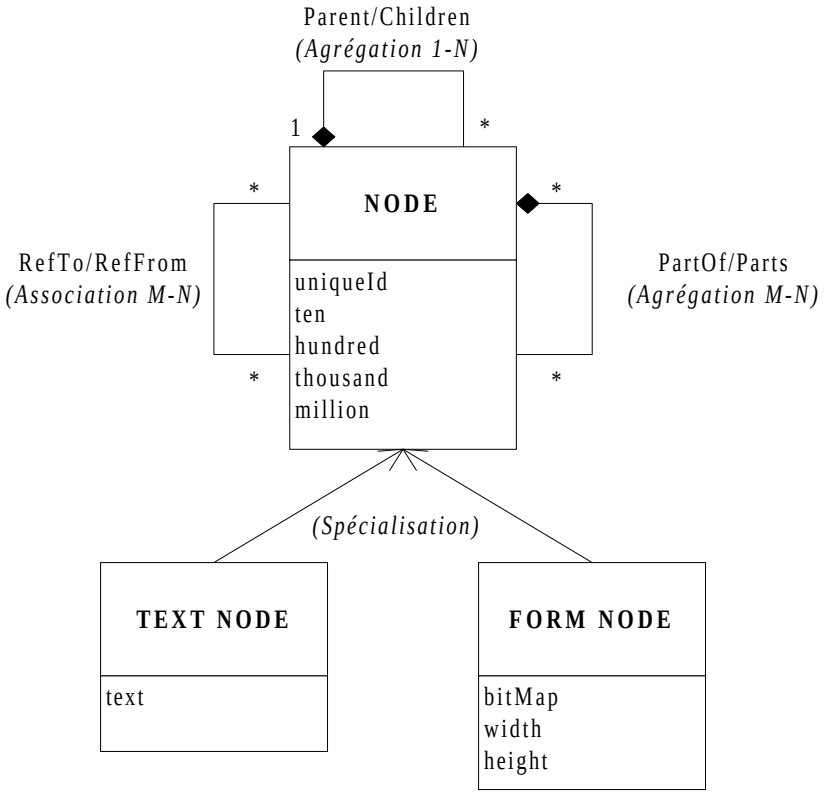

Figure 2.3 : Schéma conceptuel de la base d'objets d'HyperModel

Chaque nœud du graphe constituant la base de données test est relié à cinq autres nœuds suivant la relation parent/enfant (*Parent/Children*). Un document contient normalement sept générations de nœuds, soit au total 19531 nœuds (Figure 2.4). La relation de composition (*PartOf/Part*) est créée en reliant chaque nœud au niveau *k* de la hiérarchie parent/enfant à cinq nœuds sélectionnés aléatoirement (suivant une loi de

distribution uniforme) au niveau *k+1*. La relation d'association (*RefTo/RefFrom*) est créée en visitant chaque nœud et en créant une référence vers un autre nœud aléatoirement sélectionné parmi tous ceux de la base.

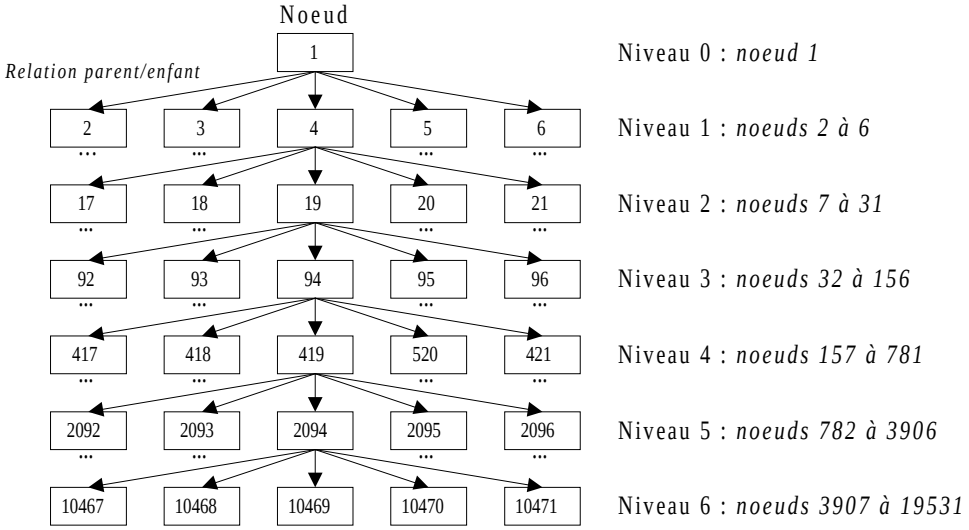

Figure 2.4 : Partie de la hiérarchie parent/enfant dans HyperModel

Chaque attribut *text* est constitué d'au plus cent mots d'au plus dix caractères. Un attribut image (*bitMap*) a une taille maximum de 400 x 400 pixels. Le plus bas niveau de la hiérarchie parent/enfant (génération n° 7) est constitué de 125 *FormNodes* et 15500 *TextNodes*, qui héritent de la classe *Node*.

3.3.2. La charge d'HyperModel

Le banc d'essais en lui-même consiste en vingt opérations. Pour mesurer le temps de réponse nécessaire à chaque opération, le protocole suivant doit être suivi.

1) Initialisation (*setup*) : 50 entrées sont préparées pour l'opération, par tirage de nombres aléatoires ; ce temps d'initialisation n'est pas pris en compte dans le temps de réponse.
2) Exécution à froid (*cold run*) : L'opération est exécutée 50 fois, en utilisant les 50 entrées précalculées à la phase d'initialisation. Si l'opération procède à des mises à jour, elle n'est validée qu'une seule fois à la fin des 50 exécutions.
3) Exécution à chaud (*warm run*) : L'opération est répétée 50 fois, en utilisant les mêmes entrées, dans le but de tester les effets d'éventuels caches. L'opération est validée une fois en fin en cas de mise à jour.

Les vingt opérations de base du banc d'essais sont subdivisées en sept types.

- Recherche par nom (*Name Lookup*) : recherche d'un nœud sélectionné au hasard.

- Recherche par valeur (*Range Lookup*) : recherche des nœuds satisfaisant un prédicat basé sur la valeur d'un attribut.
- Recherche par groupe (*Group Lookup*) : parcours d'une relation (héritage, agrégation ou association) sur un niveau à partir d'un nœud de départ sélectionné aléatoirement.
- Recherche par référence (*Reference Lookup*) : recherche par groupe en suivant les relations dans l'autre sens (par exemple, d'un nœud composant vers un nœud composite).
- Balayage séquentiel (*Sequential Scan*) : parcours de tous les nœuds.
- Fermeture (*Closure Traversal*) : recherche par groupe sur plus d'un niveau, jusqu'à une profondeur prédéfinie.
- Mise à jour (*Editing*) : mise à jour d'un nœud.

3.3.3. Banc d'essais dérivé d'HyperModel : CluB-0

Lors du développement du SGBDOO O_2 [Bancilhon et al. 88] s'est posé le problème de l'effet réel sur les performances du système du choix des arbres de placement pour grouper les objets sur disque. Pour étudier ce problème, CluB-0, un banc d'essais orienté regroupement, a été conçu [Bancilhon et al. 92].

CluB-0 est basé sur le banc d'essais HyperModel, mais son graphe de composition est légèrement différent car les concepteurs d'O_2 voulaient avoir le contrôle total d'un petit nombre de paramètres. De plus, de nombreuses opérations d'HyperModel, comme les recherches par valeur, ne pouvaient bénéficier de regroupement d'objets. Elles ont donc été écartées. En effet, seules les opérations du type fermeture sont intéressantes du point de vue du groupement d'objets.

La structure du graphe de composition de CluB-0 est la suivante. Une seule classe *N* est conservée. Son graphe d'héritage est présenté en Figure 2.5. La base de données est composée d'objets de classe *N*, les nœuds, qui forment un arbre (selon la relation *T*) entrelacé avec un graphe (selon la relation *G*). Chaque nœud de l'arbre *T* situé à la profondeur *i* est connecté à exactement *f* nœuds distincts aléatoirement sélectionnés parmi les nœuds du niveau *i*+1. Au niveau 0, les deux types de liens sont les mêmes. À tous les autres niveaux, les objets peuvent être partagés au sein du graphe engendré par la relation *G*. Chaque nœud a comme attribut un identificateur unique et une chaîne de caractères de taille variable, dont la fonction est de faire varier la taille des objets pour diverses expériences concernant les caches ou la taille des pages disque.

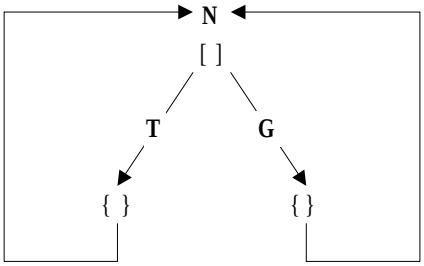

Figure 2.5 : Graphe d'héritage de la classe *N* dans CluB-0 [Bancilhon et al. 92]

Les études menées par les concepteurs d'O_2 ont consisté à générer plusieurs bases de données en faisant varier les paramètres (en particulier *f*) et à tester les effets de différents arbres de placement. Les résultats complets sont décrits dans [Harrus et al. 90].

3.3.4. Critique d'HyperModel

Le banc d'essais HyperModel possède à la fois un schéma plus riche et une gamme plus importante d'opérations qu'OO1. Cela le rend potentiellement plus pertinent qu'OO1 pour mesurer les performances des applications d'ingénierie utilisant des bases de données orientées objet (critère de pertinence). Pourtant, la notion d'objet complexe est encore peu exploitée dans HyperModel. Par ailleurs, la complexité de ce banc d'essais le rend également plus difficile à implémenter qu'OO1. HyperModel est d'autant plus difficile à implémenter que ses spécifications ne sont pas aussi claires et complètes que celles d'OO1. Il contrevient en cela au critère de faisabilité. HyperModel n'a d'ailleurs jamais rencontré un grand succès auprès des fabriquants de SGBDOO. Il a cependant été utilisé pour évaluer les performances des systèmes VBase et GemStone, ce qui joue en faveur de sa portabilité. Par contre, ses capacités en terme d'adaptabilité ne sont pas clairement exprimées dans les publications décrivant ce banc d'essais.

3.4. OO7

OO7 est plus récent qu'OO1 et HyperModel et, par conséquent, il s'appuie sur les structures décrites aux Sections 3.2 et 3.3 pour proposer un banc d'essais plus complet et pour simuler l'exécution de transactions variées sur une base de données diversifiée. Il a été conçu pour évaluer les performances des techniques et des algorithmes d'implantation des SGBDOO d'une manière plus générique que ses aînés et pour remédier à leurs carences, notamment en terme de complexité des objets manipulés et d'accès associatifs à ces objets [Carey et al. 93].

3.4.1. Base d'objet d'OO7

La base de données d'OO7 est basée sur un modèle conceptuel proche de celui du banc d'essais HyperModel, bien qu'il contienne un plus grand nombre de classes. Ce schéma conceptuel est présenté dans la Figure 2.6 comme un diagramme de classes UML. Quatre grands types de relations sont également supportées : héritage (IS-A), 1-1, 1-N, M-N.

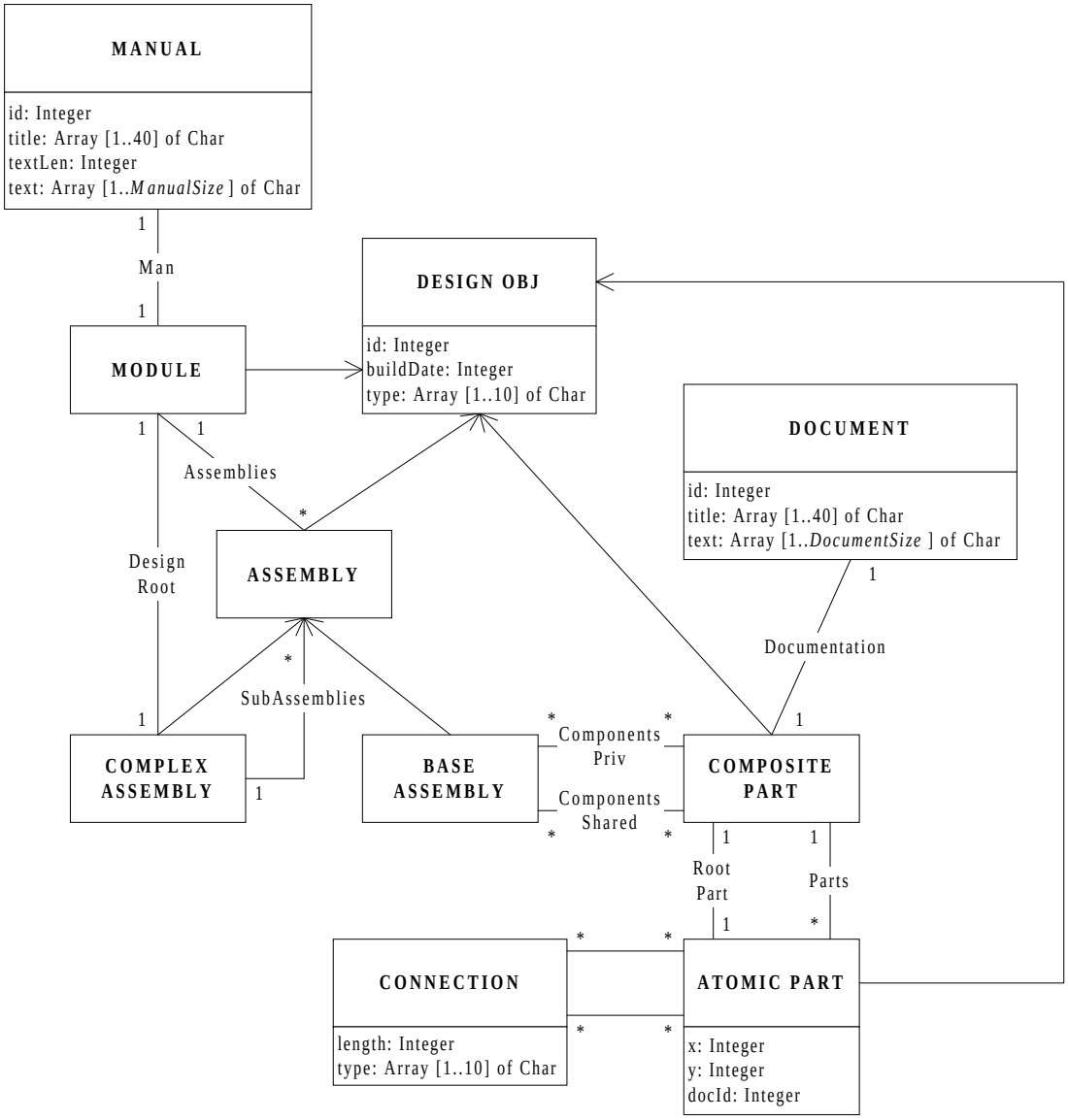

Figure 2.6 : Schéma conceptuel de la base d'objets d'OO7

Les objets composites (*composite parts*) du schéma forment ce que les auteurs appellent la « design library » au sein de la base de données OO7. La « design library » est composée de *NumCompPerModule* objets composites. Un objet document de taille *DocumentSize* est associé à chaque objet composite. De surcroît, chaque objet composite possède un graphe associé de *NumAtomicPerComp* objets atomiques (*atomic parts*).

Chacun de ces objets atomiques est lui-même connecté grâce à une association bidirectionnelle à *NumConnPerAtomic* autres objets atomiques.

Une hiérarchie d'assemblage (*assembly hierarchy*) correspond à un niveau supérieur d'objets complexes. Le premier niveau d'une hiérarchie d'assemblage consiste en des objets assemblages de base (*base assemblies*), qui sont chacun associés à *NumCompPerAssm* objets composites « partagés » et à *NumCompPerAssm* objets composites « non partagés ». Les niveaux supérieurs de la hiérarchie d'assemblage sont fait d'assemblages complexes (*complex assemblies*) qui sont liés par des associations bidirectionnelles avec *NumAssmPerAssm* sous-assemblages (*subassemblies*). Les sous-assemblages peuvent être soit des assemblages de base, soit des assemblages complexes. Il y a *NumAssmLevels* niveaux dans la hiérarchie d'assemblage. Chaque hiérarchie d'assemblage est appelée un module. Il y a *NumModules* modules. À chaque module est associé un manuel (*manual*) de taille *ManualSize*.

Il existe trois tailles possibles pour la base de données OO7 : petite, moyenne et grande. La valeur des paramètres d'OO7 pour chacune de ces tailles est fournie en Table 2.1.

Paramètre	Petite base	Base moyenne	Grande base
NumAtomicPerComp	20	200	200
NumConnPerAtomic	3, 6, 9	3, 6, 9	3, 6, 9
DocumentSize	2000 octets	20000 octets	20000 octets
ManualSize	100 Ko	1 Mo	1 Mo
NumCompPerModule	500	500	500
NumAssmPerAssm	3	3	3
NumAssmLevels	7	7	7
NumCompPerAssm	3	3	3
NumModules	1	1	10

Table 2.1 : Paramètres de la base de données OO7 [Carey et al. 93]

3.4.2. Charge d'OO7

L'éventail de transactions proposé par OO7 est également proche de celui du banc d'essais HyperModel. Trois groupes principaux d'opérations peuvent être identifiés.

- Les parcours (*traversals*) explorent le graphe d'objets en utilisant divers critères. Ces parcours sont très proches de ceux d'OO1 (par exemple, les parcours de hiérarchie d'assemblage ou les parcours avec mise à jour d'attribut). Il existe sept opérations différentes numérotées T1, T2, T3, T6, T8, T9 et TCU (pour *Cached Update*).
- Les requêtes (*queries*) recherchent des objets choisis en fonction de divers critères (par exemple, recherche aléatoire d'objet atomique, recherche par valeur comme dans HyperModel, jointures…). Il existe sept sortes de requêtes numérotées de Q1 à Q6 et Q8.

- Les opérations de modification structurelle (*structural modification operations*) gèrent l'insertion et la suppression d'objets en deux opérations distinctes.

3.4.3. Banc d'essais dérivé d'OO7 : BEAST

BEAST (*BEnchmark for Active database SysTems*) a été conçu pour tester les fonctionnalités actives des SGBD : détection des événements, gestion des règles (recherche des règles, surtout) et exécution des règles. Il se concentre plus particulièrement sur les performances des SGBDOO actifs [Geppert et al. 94].

BEAST est basé sur OO7. Il utilise son schéma et la base de données correspondante (c'est-à-dire, les programmes de génération et de peuplement de la base). Ce réemploi de la base d'objets d'OO7 est destiné à permettre à la fois l'évaluation des performances actives du système (grâce à BEAST) et de ses performances passives (grâce à OO7 standard).

Les opérations de BEAST sont donc uniquement destinées à évaluer les composants actifs. Elles sont divisées en trois grandes catégories :

- tests de détection d'événements (*Tests for Event Detection*). Ils sont eux-mêmes subdivisés en deux types de tests, les tests de détection des événements primitifs (numérotés ED-01 à ED-05) et les tests de détection des événements composites (numérotés ED-06 à ED-11) ;
- tests de gestion des règles (*Tests for Rule Management*), numérotés RM-1 et RM-2 ;
- tests d'exécution des règles (*Tests for Rule Execution*), numérotés de RE-01 à RE-05.

BEAST présente également des paramètres dont la valeur varie en fonction de la taille de la base de règles (Table 2.2). Ce sont :

- le nombre d'événements définis,
- le degré d'interconnexion des définitions d'événements (*DOI*),
- le nombre de règles définies.

Paramètre	**Taille de la base de règles**		
	Petite	*Moyenne*	*Grande*
Nombre d'événements	50	250	500
DOI	0	5	10
Nombre de règles	50	250	500

Table 2.2 : Valeur des paramètres de BEAST [Geppert et al. 94]

À notre connaissance, BEAST n'a été utilisé que pour tester les performances d'un SGBD actif dénommé SAMOS [Gatziu et al. 91], développé comme BEAST à l'Université de Zurich.

3.4.4. Critique d'OO7

OO7 tente de corriger les faiblesses des bancs d'essais OO1 et HyperModel. L'objectif est atteint grâce à un schéma riche et un ensemble d'opérations complet. Le critère de pertinence est pleinement satisfait. Cependant, si OO7 est un bon banc d'essais en ce qui concerne les applications d'ingénierie (CAO, AGL), il s'adapte moins bien à d'autres domaines comme les finances, les télécommunications et le multimédia [Tiwary et al. 95]. Comme son schéma est statique, il ne peut pas être adapté à d'autres environnements, ce qui le handicape au niveau de l'adaptabilité. [Tiwary et al. 95] affirme également que la structure d'OO7 élimine les effets de certaines techniques d'optimisation comme le groupement physique ou le préchargement des objets. Par ailleurs, la structure et les opérations d'OO7 ne sont pas triviales, ce qui rend le banc d'essais difficile à appréhender, adapter, ou tout simplement implémenter, ce qui contrevient au critère de faisabilité. Néanmoins, les implémentations d'OO7 sont disponibles par FTP anonyme[*]. Pour terminer, OO7 a été utilisé (principalement par la communauté scientifique) pour évaluer les performances de plusieurs systèmes (Exodus, Objectivity/DB, ONTOS et ObjectStore, par [Carey et al. 93] ; Texas et Opal par [Tiwary et al. 95] ; Monet par [Van den Berg et Van der Hoeven 96]), ce qui démontre sa portabilité.

3.5. Justitia

Justitia a été conçu pour permettre la comparaison de SGBDOO en vue de leur utilisation pour une application spécifique. Cependant, il peut également être employé pour évaluer l'impact de techniques d'implémentation. Il s'appuie sur la constatation [Schreiber 94] [Schreiber 95] que les bancs d'essais antérieurs ne présentent pas de fonctionnalité multi-utilisateurs satisfaisante (ce qu'il est important de tester dans un contexte client-serveur) et ne savent pas tester les capacités d'un SGBDOO à réorganiser sa base de données. Justitia s'attache à combler ces lacunes.

3.5.1. Base d'objets de Justitia

La structure de la base de données de Justitia est un arbre de relations 1-N bidirectionnelles entrelacé avec un graphe d'héritage. Son schéma est présenté dans la

[*] ftp://ftp.cs.wisc.edu/OO7

Figure 2.7 sous forme de diagramme de classes UML. Les classes de base qui le composent sont inspirées des structures de données des applications d'ingénierie :

- *objets de taille dynamique* (DO) : ils contiennent un champ de données non structuré BLOB (*Binary Large OBject*) dont la taille moyenne est un paramètre du banc d'essais (Table 2.3). La taille réelle de chaque instance varie aléatoirement, ce qui immunise le banc d'essais contre des optimisations de SGBDOO en fonction d'une taille d'objet donnée ;
- *objets de taille statique* (SO) : ils représentent un objet typique d'une application donnée et contiennent un champ de données de type tableau d'octets de taille fixe. La taille de ces objets est connue du SGBD, qui est autorisé à utiliser cette information pour effectuer des optimisations ;
- *objets conteneurs* (CO) : ce sont des nœuds qui pointent chacun vers un anneau de références vers des *objets primitifs* (PO), qui représentent des objets complexes internes. Chaque objet PO pointe sur un ensemble (défini par l'utilisateur) d'autres PO appartenant au même CO.

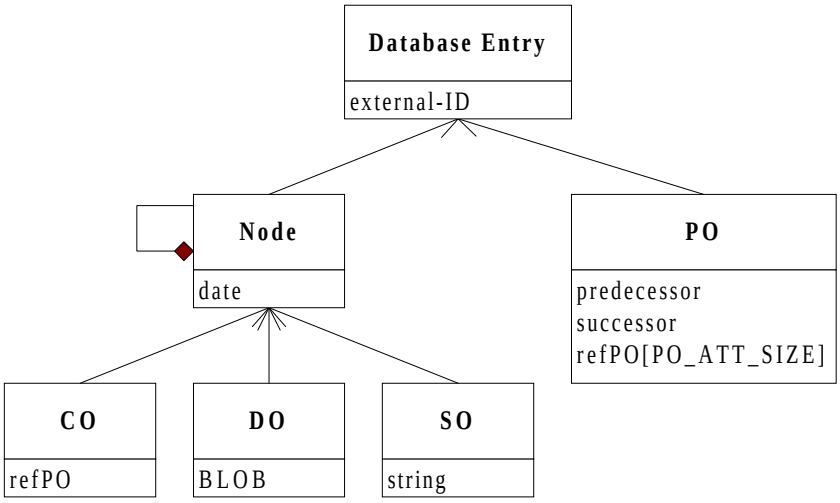

Figure 2.7 : Schéma de la base d'objets de Justitia

Les classes CO, DO et SO héritent de la classe *Node*. Les instances de la classe *Node* représentent en fait les composants d'une hiérarchie d'agrégation. Chaque objet *Node* possède un attribut *date* qui correspond à la date de sa génération ou de sa dernière mise à jour. À l'opposé, les objets primitifs PO ne participent à aucune hiérarchie. Ils ne servent que de structure de base pour les objets complexes.

Les classes *Node* et PO héritent de la classe *Database Entry*, qui n'est définie que par un unique attribut identificateur *external-ID*. Ces identificateurs sont logiques et contrôlés par l'utilisateur. Ils sont générés automatiquement pendant la création de la base.

De plus, pour toutes ces classes, la distinction est faite entre la partie indépendante de tout SGBDOO et la partie dépendant du système. Ainsi, lorsque le banc d'essais est porté

sur plusieurs SGBDOO, seule la partie dépendant du système nécessite d'être changée. La partie indépendante est fournie sous forme de librairies de classes C++, qui contiennent également des primitives de mesure de performance génériques.

Finalement, chaque base de donnée est constituée d'un ensemble paramétrable de sections dont les nœuds sont logiquement reliés. Ces sections sont nécessaires pour l'organisation de mises à jour concurrentes de la base (cf. Section 3.5.3).

3.5.2. Charge de Justitia

Toutes les opérations du banc d'essais sont basées sur quelques transactions de base, qui peuvent être combinées par l'utilisateur pour former des opérations composites.

3.5.2.1. Opérations de base

Les opérations de base utilisent les méthodes d'accès élémentaires des classes du banc d'essais et modélisent des accès simples.

- *Lecture simple :* Accès à un ensemble d'objets, lecture de l'attribut *date* de chacun des objets.
- *Mise à jour simple :* Accès à un ensemble d'objets, mise à jour de l'attribut *date* de chacun des objets.
- *Lecture étendue :* Accès à un ensemble d'objets, lecture de l'attribut *date* de chacun des objets et exécution d'une opération variant selon la classe. CO : lecture du premier pointeur du tableau de pointeurs de chaque PO. DO/SO : détermination de la fréquence d'un caractère donné dans les chaînes de caractères.
- *Mise à jour étendue :* Accès à un ensemble d'objets, mise à jour de l'attribut *date* de chacun des objets et exécution d'une opération variant selon la classe. CO : insertion ou suppression d'un PO. DO : réinitialisation de l'attribut BLOB dont la longueur prend une nouvelle valeur aléatoire. SO : remplacement de chaque caractère de l'attribut par un caractère aléatoire.

3.5.2.2. Opérations composites

Ces opérations sont définies par l'utilisateur du banc d'essais au moyen d'une série de paramètres (Table 2.3). Leur but est de mesurer l'efficacité du mode multi-utilisateurs. Nous donnons quelques exemples d'opérations composites.

- *Multi-ultilisateurs, multi-sections :* exécution de toutes les opérations de base en parallèle, chaque processus travaillant sur une section distincte de la base de données.

- *Multi-utilisateurs, une section :* exécution de toutes les opérations de base en parallèle, chaque processus travaillant sur la même section de la base de données.
- *Réorganisation :* exécution de toutes les opérations de base en mode mono-utilisateur, manipulation de la structure physique de la base et répétition des opérations de mesure.

3.5.3. Aspect multi-utilisateurs de Justitia

Justitia utilise un processus de synchronisation (processus maître) et plusieurs processus exécutant des opérations et effectuant des mesures de performance (processus esclaves). Le processus maître coordonne et contrôle toutes les activités des processus esclaves et est chargé de synthétiser l'évaluation de performance finale. Les processus esclaves peuvent être assimilés à des utilisateurs individuels. Ils peuvent être individualisés grâce à une série de paramètres (Table 2.3). Les processus esclaves passent tout d'abord par une phase d'initialisation (connexion à la base, préparation des références). Ils attendent ensuite l'autorisation du processus maître avant d'exécuter leurs opérations. Le temps nécessité pour l'exécution d'une opération donnée est calculé par moyenne pour compenser la faible précision des horloges système pour des temps très brefs.

Nom du paramètre	Description
SO_ATT_SIZE	Taille de la chaîne de caractère attribut de SO
PO_ATT_SIZE	Nombre de liens d'un PO vers d'autres PO
CO_ATT_SIZE	Nombre moyen de PO appartenant à un CO
CO_ATT_DEVI	Variation du nombre de PO appartenant à un CO
DO_ATT_SIZE	Taille moyenne d'un attribut BLOB de PO
DO_ATT_DEVI	Variation de taille pour les attributs BLOB de PO
SECTION	Nombre de sections
MAXWIDTH	Largeur des sections
MAXLEVEL	Profondeur des sections
TYPE_ASSOC	Assignation des types de nœuds aux sections
ACCESS_MODE	Type d'opération
TASKS	Nombre de processus esclaves concurrents
WORK_SECT	Assignation des processus aux sections
REPETITIONS	Nombre de répétitions des exécutions à chaud
BLOW_UP_SIZE	Facteur de destruction pour les tests de réorganisation

Table 2.3 : Paramètres de Justitia

3.5.4. Critique de Justitia

Le point fort de Justitia est sans conteste sa prise en compte implicite d'utilisateurs multiples, ce qui n'avait jamais été vraiment fait (une version multi-utilisateurs d'OO7 était prévue, mais elle n'a pas été finalisée à notre connaissance). Cependant, la prise en compte du contexte multi-utilisateurs complique beaucoup la mise en œuvre du banc d'essais, qui n'apparaît pas triviale. De plus, les spécifications publiées de Justitia manquent de précision pour que le travail des auteurs puisse être reproduit aisément, ce qui

contrevient au critère de faisabilité. Justitia présente également d'autres défauts. Il se veut générique, mais fait encore appel pour la génération de son schéma à des structures typiques des applications d'ingénierie. Le schéma de la base d'objets de Justitia est d'ailleurs plus limité que ceux d'HyperModel ou d'OO7. Malgré la diversité des types d'objets, les relations inter-objets sont, elles, très restreintes. Le graphe d'héritage est substantiel, mais les autres types de références se limitent à une hiérarchie de composition. Cela contrevient au critère de pertinence. Par contre, Justitia est complètement paramétrable et satisfait le critère d'adaptabilité. Il a également été implémenté par ses auteurs sur divers SGBDOO du commerce : Itasca, Objectivity/DB, ObjectStore, ONTOS et VERSANT, ce qui témoigne d'une bonne portabilité.

3.6. Synthèse

Nous récapitulons dans la Table 2.4 les points forts et les points faibles que nous attribuons aux bancs d'essais que nous venons de présenter, en nous basant sur les quatre critères définis par [Gray 93] pour la conception d'un « bon » banc d'essais (pertinence, faisabilité, adaptabilité, portabilité).

	Pertinence	***Faisabilité***	***Adaptabilité***	***Portabilité***
OO1	**– –**	**++**	**–**	**++**
HyperModel	**+**	**–**	**– –**	**+**
OO7	**++**	**–**	**–**	**+**
Justitia	**–**	**– –**	**+**	**+**

Point fort : + *Point très fort : ++* *Point faible : –* *Point très faible : – –*

Table 2.4 : Tableau comparatif des bancs d'essais étudiés

De plus, nous énumérons ci-dessous des points qui ne sont pas couverts par ces critères et pour lesquels les bancs d'essais étudiés présentent des faiblesses. Une partie de notre travail consiste à proposer des solutions à ces problèmes en concevant un nouveau banc d'essais adapté à nos besoins (cf. Chapitre 3).

- Tous les bancs d'essais que nous avons présentés aspirent à une certaine généricité, mais tous mettent en œuvre des schémas de bases de données inspirés des structures en usage dans les logiciels d'ingénierie (CAO, PAO, AGL…) ce qui les rend finalement plus adaptés à l'étude des performances de ces systèmes en particulier. Leur pertinence en souffre dans les domaines qui ne relèvent pas de l'ingénierie.
- L'adaptation de ces bancs d'essais pour un objectif donné demande des efforts, c'est-à-dire la conception d'un banc d'essais dérivé permettant de prendre en compte des éléments spécifiques (cf. les cas de DSTC-CluB, CluB-0 et BEAST). Il n'est pas possible de configurer ces bancs d'essais, hormis Justitia, pour une tâche précise. Leur adaptabilité n'est donc pas optimale.

- Seul Justitia propose un environnement multi-utilisateurs, alors que dans un contexte client-serveur (qui est désormais le standard pour les SGBDOO), il n'est pas seulement important de connaître la capacité d'un système à absorber la charge, mais aussi la façon dont se déroule le contrôle de concurrence (et ses incidences sur les performances). C'est encore le critère de pertinence qui n'est pas entièrement satisfait.
- Finalement, aucun de ces bancs d'essais n'intègre l'aspect qualitatif des systèmes étudiés, c'est-à-dire les fonctionnalités offertes (possibilités de *backup* et de reprise après panne, compactage de la base de données en ligne, regroupement d'objets, sécurité…), qui peuvent aussi bien déterminer le choix de décideurs en faveur d'un SGBDOO plutôt qu'en faveur d'un autre.

Pour terminer, la spécification d'un bon banc d'essais n'est pas le seul problème que les concepteurs de ces outils rencontrent [Carey et al. 94b]. De nombreux fabriquants de SGBDOO incluent dans les licences de leur produits des clauses qui interdisent la publication de tests de performance sans leur accord explicite. Il est de fait illégal d'acheter un produit, de tester ses performances avec un banc d'essais et de publier les résultats. Les concepteurs d'OO7 ont d'ailleurs dû renoncer à publier les résultats qu'ils avaient obtenus concernant le système ObjectStore, sous la menace d'une action en justice de la part d'Object Design, Inc. [Carey et al. 93]

4. Simulation des SGBDOO

L'utilisation de la simulation aléatoire à événements discrets est peu répandue dans le domaine des bases de données. La littérature traitant de modèles de simulation de SGBDOO est particulièrement réduite. En général, le système étudié est disponible et l'approche par banc d'essais est naturellement préférée à l'approche simulation. Néanmoins, certains concepteurs de techniques d'optimisation (groupement d'objet, gestion de cache mémoire…) ont recours à la simulation afin de valider leur proposition. Nous présentons dans cette section quelques-uns de ces modèles de simulation, qui nous intéressent au premier chef.

Par ailleurs, l'approche simulation est plus volontiers utilisée pour évaluer les performances des systèmes répartis, auxquels les bancs d'essais sont mal adaptés. C'est dans ce contexte que le plus grand nombre de contributions est disponible. Cependant, les problèmes traités s'éloignent de nos préoccupations : dégroupement d'objets (*declustering*) plutôt que groupement, réplication des objets, problèmes de communication… Ces études fournissent toutefois quelques éléments pouvant nous être utiles.

Finalement, nous nous intéressons également dans cette section à des études de modélisation et de simulation des disques magnétiques, dont l'accès est pris en charge par la partie basse de tout SGBD.

4.1. Modèles de simulation pour l'évaluation de techniques de regroupement d'objets

4.1.1. Travaux de Chang et Katz

Ellis Chang et Randy Katz, de l'Université de Californie, Berkeley, ont utilisé une approche par simulation pour valider leurs propositions concernant des techniques de groupement d'objets et de gestion de cache mémoire [Chang et Katz 89] dans un contexte CAO (cf. Chapitre 1, Section 3.3.1.2). Leur objectif était de découvrir comment différents algorithmes d'optimisation influencent les performances lorsque les caractéristiques de l'application qui accède aux données varient et quelle est la relation entre le groupement des objets et des paramètres comme le taux de lecture/écriture des objets.

Le modèle de simulation proposé [Chang 89] [Chang et Katz 89] est composé de plusieurs stations de travail et serveurs de fichiers. Un fichier des activités des utilisateurs incluant des fonctions d'accès aux données (ouverture, fermeture, lecture, écriture), des modes d'accès aux données (base de données ou fichiers) et des volumes de données a été construit à partir de renseignements concernant l'utilisation réelle du gestionnaire d'objets CAO OCT, en usage à Berkeley. Les données collectées concernant OCT couvrent plus de 400 heures d'utilisation.

Le modèle de simulation en lui-même a été conçu avec le logiciel PAWS (*Performance Analysis Workbench System*), qui supporte un grand nombre de fonctionnalités de haut niveau, comme des primitives de gestion de file d'attente selon plusieurs politiques, des statistiques détaillées en sortie, etc. Il consiste en six modules en interaction (Figure 2.8).

- *Stations de Travail :* Ensemble de stations de travail modélisant les utilisateurs travaillant de manière interactive et les temps de latence induits.
- *Définition de la Charge :* Module chargé de définir les caractéristiques de la charge appliquée au système.
- *Gestionnaire de Regroupement :* Module chargé de mettre en œuvre diverses politiques de regroupement des objets.
- *Gestionnaire de Cache Mémoire :* Module chargé de mettre en œuvre diverses politiques de remplacement de pages dans le cache ;
- *Gestionnaire de Transactions :* Module chargé d'exécuter les transactions.
- *Sous-système des E/S :* Ensemble de composants décrivant la configuration des entrées-sorties.

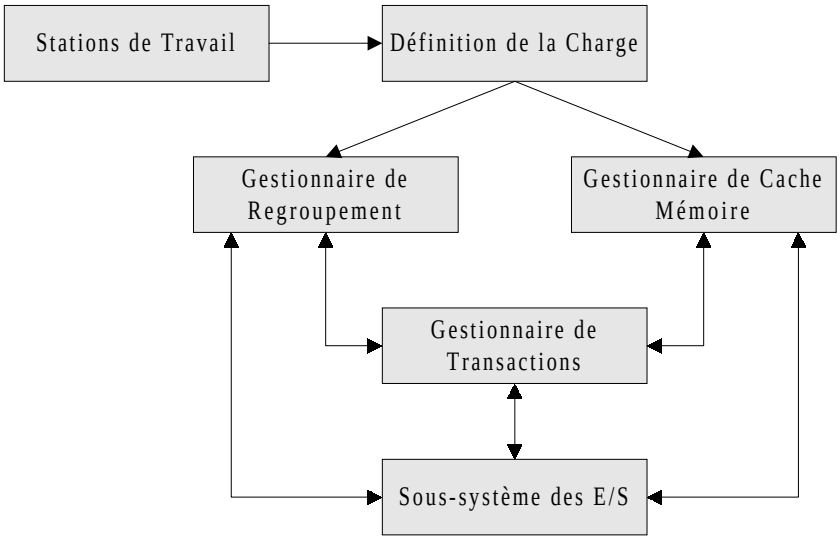

Figure 2.8 : Architecture du modèle de simulation de Chang et Katz

Dans ce modèle, une transaction est initiée dans le module des *Stations de Travail* avant d'être soumise, après un temps de latence prédéfini, au module de *Définition de la Charge*. Chaque transaction est associée à une *catégorie* (fixe) et une *phase* (qui varie au cours de l'exécution de la requête). Si la requête est une opération de lecture, le *Gestionnaire de Cache Mémoire* vérifie si les informations nécessaires se trouvent dans le cache et requiert une entrée-sortie au *Sous-système des E/S* si ce n'est pas le cas. L'exécution courante des requêtes (et de leur *phase*) est gérée par le *Gestionnaire de Transactions*. Le *Gestionnaire de Groupement* fonctionne en parallèle et administre le groupement physique des objets sur le disque, en fonction de l'usage qui est fait de la base.

L'exécution des simulations peut varier selon un certain nombre de paramètres (Table 2.5), divisés en paramètres statiques qui sont fixés pour toutes les simulations et en paramètres de contrôle qui peuvent varier selon les expériences.

Paramètres statiques	**Valeur par défaut**
Taille de la base de données	300 Mo
Taille des pages disque	4 Ko
Nombre de sessions actives	10
Temps de latence	4 s
Paramètres de contrôle	**Valeurs**
Densité de structure	low-3, med-5, high-10
Taux de lecture/écriture	5, 10, 100 %
Politique de regroupement	No_Cluster, Cluster_within_Buffer, 2_IO_limit, 10_IO_limit, No_limit
Politique de fractionnement de page	No_Split, Linear_Split, NP_Split
Politique d'intervention des utilisateurs	No_hint, User-hint
Politique de remplacement de pages dans le cache	LRU, Context_sensitive, Random
Taille du cache mémoire	100, 1000, 10000 pages
Politique de préchargement	No-prefetch, Prefetch_within_buffer_pool, Prefetch_within_Database

Table 2.5 : Paramètres de simulation [Chang 89]

La densité de structure est une caractéristique de la charge. Par exemple, « low-3 » signifie que chaque requête retourne au plus trois composants d'un objet composite. La politique de fractionnement de page entre en jeu dans la politique de groupement d'objets proposée par Chang et Katz (cf. Chapitre 1, Section 3.3.1.2).

4.1.2. Travaux de Tsangaris et Naughton

Emmanuel-Manolis Tsangaris et Jeffrey Naughton ont également utilisé une approche « simulation » pour comparer les algorithmes de partitionnement de graphes appliqués au groupement d'objets proposés dans [Tsangaris et Naughton 91] avec des algorithmes antérieurs utilisés en théorie des graphes (cf. Chapitre 1, Section 3.3.3.1) [Tsangaris et Naughton 92]. Il est important de noter que cette approche est totalement déterministe et, bien que qualifiée de simulation, a peu de choses en commun avec la simulation aléatoire à événements discrets.

L'outil employé est CLAB (*CLustering LAboratory*), un système expérimental spécialement conçu pour cette étude [Tsangaris 92]. CLAB est constitué d'un ensemble d'outils Unix programmés en C++ et pouvant être assemblés selon diverses configurations. Ces programmes communiquent entre eux au moyen de fichiers textes Unix ou de mécanismes de *pipe*. Il est ainsi facile d'inspecter et de modifier les données produites par ces outils.

- *Gen* est une application orientée objet synthétique qui implémente le banc d'essais CluB-0 (cf. Section 3.3.3) et génère un graphe d'objets et des traces d'accès associées à ce graphe.
- *Smooth* est un générateur de modèles d'accès. À partir d'une trace d'accès et d'un graphe d'objets (optionnel), *smooth* effectue une analyse statistique de la trace et peut générer un graphe représentant les modèles d'accès IID et SMC présentés à la Section 3.3.3.
- *Cl* est un outil de regroupement d'objets dans des pages disque à partir d'un modèle d'accès donné. *Cl* implémente plusieurs algorithmes de partitionnement de graphe et produit un objet *cluster* (assignation des objets à des pages).
- *Sim* est un simulateur de disque et un moniteur de performances. *Sim* prend en compte plusieurs stratégies de remplacement de pages en mémoire cache, dont LRU. Il utilise en entrée une trace d'accès et un *cluster* (optionnel). Il génère un flux de fautes de page pour une taille de cache mémoire et une politique de remplacement de pages données, ainsi que des statistiques sur l'utilisation du cache mémoire.
- *Stawk* ressemble à l'outil Unix *awk*. Il fournit une interface relationnelle pour tous les objets produits par les précédents programmes. *Stawk* implémente les fonctions statistiques les plus courantes (moyenne et écart type). Il est utilisé pour traiter les

statistiques brutes fournies par *Sim* et générer des courbes qui peuvent être visualisées avec les outils standards d'Unix, comme *xgraph* ou *gnuplot*.

La Figure 2.9 représente une configuration typique de CLAB pour tester les performances d'un cache client utilisant les *clusters* générés par les algorithmes étudiés dans [Tsangaris et Naughton 91].

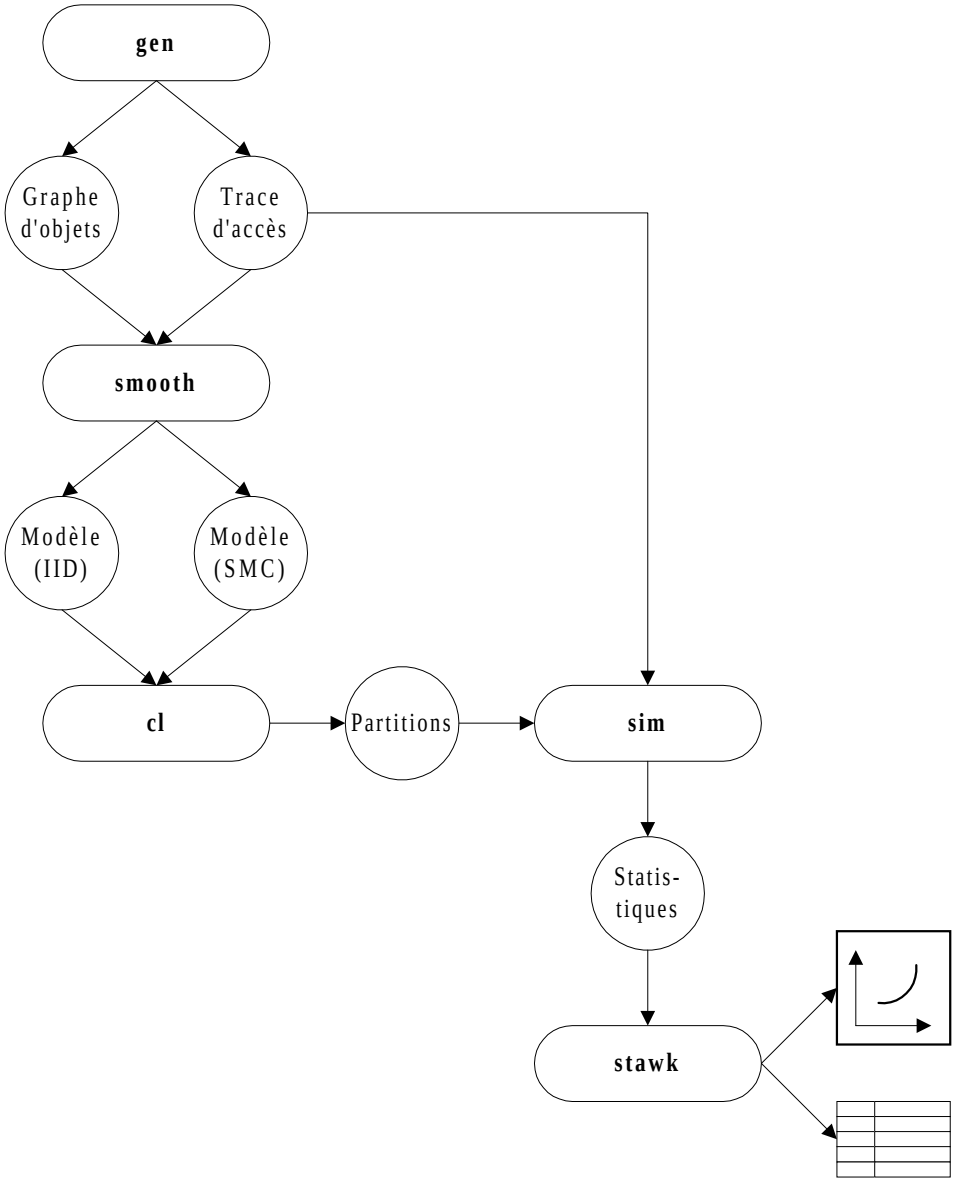

Figure 2.9 : Configuration typique de CLAB

4.1.3. Travaux de Darmont, Gay et Gruenwald

Ces travaux, effectués au sein de l'Université d'Oklahoma, étendent l'étude présentée dans [Chabridon et al. 93] concernant les performances des méthodes de regroupements

utilisées dans les systèmes ORION et Cactis et la proposition de Chang et Katz (cf. Section 2.2). Ils utilisent pour cela une approche par simulation aléatoire à événements discrets. L'architecture du modèle de simulation est inspirée de celle proposée par Chang (cf. Section 4.1.1) [Darmont et Gruenwald 96b]. Sa structure est présentée dans la Figure 2.10.

- *Utilisateur :* Après un temps de latence prédéfini, l'utilisateur génère les transactions et les soumet au Gestionnaire de Transactions. Chaque type de transaction a une probabilité d'occurrence.
- *Gestionnaire de Transactions :* Le Gestionnaire de Transactions détermine les objets individuels nécessaires aux transactions (qui dépendent du type de transaction) et exécute les opérations prévues avec ces derniers. Pour une transaction normale, les objets sont requis au Gestionnaire de Mémoire Cache. Dans le cas d'une création d'objet ou d'une demande de regroupement, la main est passée au Gestionnaire de Regroupement.
- *Gestionnaire de Mémoire Cache :* Le Gestionnaire de Mémoire Cache vérifie si un objet est présent en mémoire primaire. Si ce n'est pas le cas, la page contenant l'objet est requise du Sous-système des Entrées-Sorties. Le Gestionnaire de Mémoire Cache met également en œuvre une politique de remplacement de pages dans le cache (FIFO).
- *Gestionnaire de Regroupement :* Le Gestionnaire de Regroupement est activé différemment selon l'algorithme de groupement testé (ORION, Cactis ou Chang et Katz). Il s'occupe de réorganiser la base de données en mémoire secondaire.
- *Sous-système des E/S :* Le Sous-système des Entrées-Sorties gère les accès physiques à la mémoire secondaire.

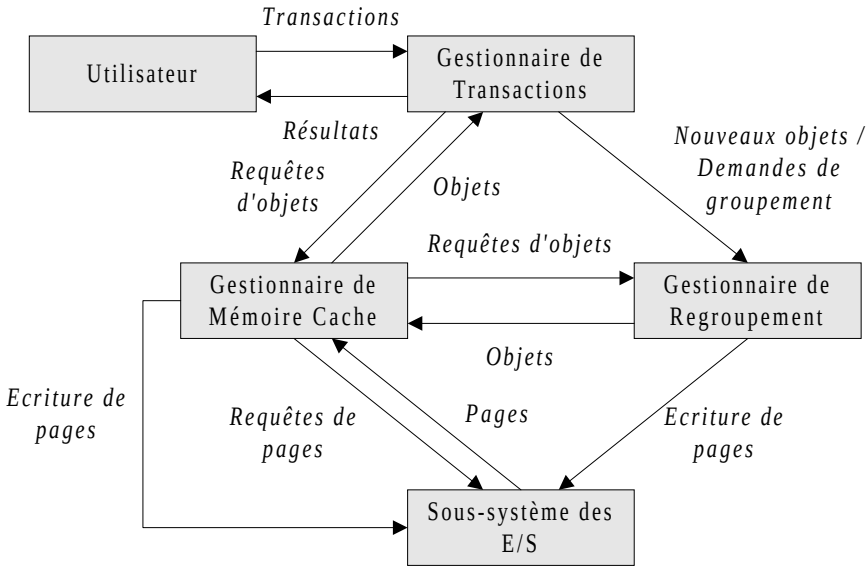

Figure 2.10 : Structure du modèle de simulation de Darmont et Gruenwald

Les différentes transactions exécutées par ce modèle de simulation sont celles du banc d'essais HyperModel, présenté à la Section 3.3.2. Elles affectent une base d'objets aléatoirement générée en deux étapes : premièrement, des hiérarchies de classe sont créées (héritage, agrégation, liens d'équivalence, versions), puis des instances de ces classes sont générées. Les deux algorithmes de génération de la base sont donnés par la Figure 2.11 et la Figure 2.12.

```
    Étant donné un certain nombre de classes, une hiérarchie d'héritage incluant
des versions est tout d'abord créée (1), puis la hiérarchie d'agrégation y est
greffée ainsi que les liens d'équivalence (2).

(1)  Ajout d'une nouvelle classe.
          Ajout d'un nombre aléatoire de versions de cette classe (descendants).
          Si la nouvelle classe a une superclasse (sachant la probabilité
          d'avoir une superclasse) alors :
               Sélection aléatoire d'une superclasse parmi les classes
                    existantes.
               Héritage des attributs et des méthodes de la superclasse.
               Pour chaque version supplémentaire de la classe :
                    Sélectionner aléatoirement une superclasse parmi les classes
                         descendant de la superclasse de la classe initiale.
                    Héritage des attributs et des méthodes de la superclasse.
               Fin pour
          Fin si
          Ajout d'attributs et de méthodes supplémentaires à toutes les
               versions (la taille des attributs et des méthodes est assignée de
               façon aléatoire).
          Calcul de la taille des instances de la classe.

(2)  Pour toutes les classes :
          Si la classe est un composant d'une autre classe (sachant la
          probabilité d'être un composant) alors :
               Sélection aléatoire d'une classe composée de la classe courante.
          Fin si
          Si la classe a une classe équivalente (sachant la probabilité d'avoir
          une classe équivalente) alors :
               Sélection aléatoire de la classe équivalente.
          Fin si
     Fin pour
```

Figure 2.11 : Génération du treillis de classes [Darmont et Gruenwald 96b]

Le comportement de ce modèle de simulation peut être réglé par un ensemble complet de paramètres, divisés en paramètres statiques qui ne varient pas au cours des expériences de simulation et en paramètres dynamiques (Table 2.6 et Table 2.7).

Ce modèle de simulation a par la suite été réemployé pour comparer les performances de la technique de regroupement proposée par Jean-Yves Gay (cf. Chapitre 1 Section 3.3.1.2) avec celles d'ORION et de Cactis [Gay 96] [Gay et Gruenwald 97].

```
Pour chaque nouvel objet :
     Sélection aléatoire d'une classe.
     Si la classe du nouvel objet est un composant d'une autre classe alors :
          Sélection aléatoire d'une instance de cette classe (s'il en existe)
               pour être composée de l'objet courant.
     Fin si
     Si la classe du nouvel objet est une version alors :
          Sélection aléatoire d'un objet ancêtre parmi les instances de la
               classe ancêtre de la classe du nouvel objet.
          Si l'algorithme de Chang et Katz est utilisé alors :
               Héritage d'instance à instance des valeurs des attributs communs.
          Fin si
     Fin si
     Si la classe du nouvel objet a une classe équivalente alors :
          Sélection aléatoire d'un objet équivalent parmi les instances de la
               classe équivalente.
     Fin si
Fin pour
```

Figure 2.12 : Génération des instances [Darmont et Gruenwald 96b]

Paramètre	Désignation	Valeur
RCC	Temps nécessaire au contrôle de concurrence	0,5 ms
IMLVL	Niveau de multiprogrammation	10
IWDSIZE	Taille du mot mémoire	4 octets
ICPU	Puissance du CPU	2 Mips
RMACC	Temps d'accès au mot mémoire	0,0001 ms
RMTEST	Temps de comparaison entre deux mots mémoire	0,0007 ms
IPGSIZE	Taille des pages disque	2048 octets
RSEEK	Temps de recherche disque moyen	28 ms
RLATENCY	Temps de latence disque moyen	8,33 ms
RTRANSFER	Temps de transfert disque moyen	1,28 ms

Table 2.6 : Paramètres de simulation statiques [Darmont et Gruenwald 96b]

4.2. Simulation des SGBDOO répartis et parallèles

4.2.1. Méthodologies de modélisation

Partant de la constatation que les méthodologies de prédiction des performances des SGBD répartis ne sont pas très adaptées à ce domaine et qu'elles engendrent des modèles trop coûteux, tant au niveau de la définition du modèle que de son analyse (en raison de la complexité structurelle des systèmes répartis et de leur taille), [Iaezolla et Mirandola 95] présente et compare deux méthodologies de modélisation en terme de coût d'exécution et de précision des résultats.

Paramètre	Désignation	Valeur par défaut	Variation
RAVGTHINK	Temps de latence utilisateur	4 s	0,1-10 s
NCL	Nombre de classes	20	10-30
IAVGVER	Nombre moyen de versions par classe	3	1-5
RPSUPER	Probabilité pour une classe d'avoir une superclasse	0.9	0-1
RPCOMP	Probabilité pour une classe d'être composant d'une autre classe	0.5	0-1
RPEQUI	Probabilité pour une classe d'avoir une classe équivalente	0.1	0-1
INOBJ	Nombre d'objets initial	400	100-1000
IAVGASIZE	Taille moyenne des attributs	1 mot	1-3 mots
IAVGNATTR	Nombre moyen d'attributs par classe	10	5-20
IBUFF	Taille du cache mémoire	10 pages	10-100 pages
IMD	Profondeur maximum des fermetures	5	3-10
ISEGSIZE	Taille de segment par défaut (ORION)	5	3-10
ITHRESHOLD	Seuil de mise à jour (C&K)	25	0-255
ISCALEF	Facteur d'échelle (C&K)	0.5	0-1
ISPLIT	Politique de fractionnement de page	ON	ON/OFF
PT1-PT12	Probabilités des opérations de lecture (op. n° 1 à 12)	0.065	0-1
PT13	Probabilité de mise à jour	0.1695 (Cactis) 0.169 (ORION) 0.17 (C&K)	0-1
PT14	Probabilité de création d'objet	0.05	0-1
PT15	Probabilité de regroupement	0.0005 (Cactis) 0.001 (ORION) 0 (C&K)	0-1
SIMTIME	Temps de simulation	10800 s	3600-86400 s

Table 2.7 : Paramètres de simulation dynamiques [Darmont et Gruenwald 96b]

La première de ces méthodologies, baptisée *independent modelling approach* (IMA), considère séparément le modèle de la partie logicielle du SGBD et le modèle de sa partie « machinerie » (la machine abstraite). Ces deux modèles sont également distingués de leurs implémentations respectives. De cette façon, les aspects concernant la gestion des données peuvent être analysés séparément de ceux concernant la gestion des ressources du système. Il est de surcroît possible de modéliser de façon analytique certaines parties du système (logiciel) et ainsi d'appliquer des méthodes combinant simulation et moyens analytiques pour l'évaluation de performances. La combinaison de ces deux types de méthodes a pour but de compenser les limitations de chacune des approches prise séparément.

La seconde méthodologie, *pure simulation* (PSA), est plus classique et son application amène à la conception d'un seul modèle qui gère à la fois les aspects logiciel et système. Cette approche aboutit à une évaluation des performances par simulation.

La comparaison des deux approches montre que celles-ci sont équivalentes au niveau des résultats, malgré des approximations dans le modèle analytique d'IMA. En revanche, IMA permet d'abaisser les coûts de définition des modèles, qui sont bien moins complexes que le modèle unique de PSA.

4.2.2. Modélisation de bases d'objets et de transactions

Afin de tester l'efficacité d'un protocole de stockage pour SGBDOO répartis, [He et al. 93] recourent à un modèle de simulation. La base de données utilisée dans ce modèle forme un graphe orienté acyclique (*Directed Acyclic Graph* ou DAG). Il est en effet considéré que dans les SGBDOO réels, les séquences d'accès aux objets suivent ce genre de schémas. Lorsqu'un objet particulier est référencé, la probabilité qu'un objet qui lui est lié soit référencé à son tour est élevée, ce qui génère un DAG. Cette approche n'est pas forcément réaliste car des cycles dans le graphe des références inter-objets sont parfaitement envisageables.

Cette base d'objets a été utilisée dans des expériences de simulation d'un environnement réparti comportant cinq sites. L'objectif était de comparer plusieurs algorithmes de répartition des objets sur ces sites distants, les performances du système étant mesurées par le trafic réseau.

La simulation est également employée pour tester le comportement de SGBD parallèles. [Bates et al. 95] propose un modèle pour la génération et l'exécution des transactions dans un système parallèle qui pourrait s'appliquer directement pour un SGBD non parallèle (Figure 2.13). Il présente d'ailleurs des similarités avec le modèle de SGBDOO de Chang et Katz (cf. Section 4.1.1).

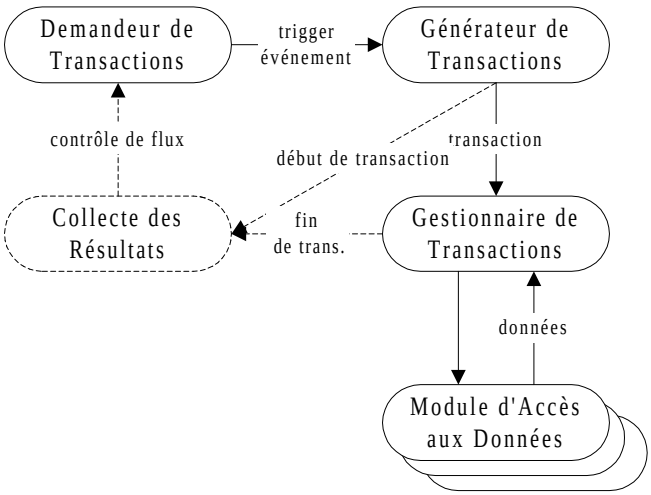

Figure 2.13 : Génération et exécution de transactions dans un modèle de SGBD parallèle

Le *Générateur de Transactions* crée la charge traitée par la simulation. Cette charge est structurée comme celles typiquement produites par des applications bancaires. Le *Demandeur de Transactions* permet de varier le nombre de transactions exécutées de façon concurrente, afin de voir comment le système gère toutes ses ressources pour exécuter une seule transaction complexe ou de nombreuses transactions concurrentes. Le *Gestionnaire de Transactions* et le *Module d'Accès aux Données* administrent l'exécution des

transactions. Un module de *Collecte des Résultats* de simulation, externe au système, est également représenté sur la Figure 2.13.

Ce modèle de simulation a été employé pour déterminer la configuration matérielle optimale (nombre de disques, stratégies d'indexation et de gestion de cache…) d'une unité de stockage en mémoire dénommée DAC (*Data Access Component*) au sein d'une nouvelle architecture de SGBD parallèle. Il a également servi à examiner l'effet d'accès concurrents au même disque ou à la même partition de disque au sein de ce système parallèle.

4.3. Modélisation et simulation des disques magnétiques

Les performances des disques magnétiques se sont beaucoup améliorées ces dernières années, permettant désormais des temps d'accès en deçà de dix milli-secondes par entrée-sortie. Cependant, ces accès demeurent bien plus lents que les accès en mémoire centrale (il existe encore un facteur de l'ordre de 10^5 entre les temps d'accès en mémoire primaire et en mémoire secondaire). L'accès à des données stockées sur disque reste donc un véritable goulot d'étranglement pour les applications.

Les principaux recours actuels pour les constructeurs se situent dans l'amélioration de la vitesse de rotation des disques, de la vitesse d'accès aux bus et dans l'implantation de caches au sein même des unités de disque [Karedla et al. 94]. Comme la conception de prototypes est délicate et peut s'avérer coûteuse, la simulation est couramment employée pour évaluer *a priori* l'impact sur les performances d'une technologie en cours de développement. Nous nous contentons d'énoncer ici quelques notions de base, qui nous seront suffisantes par la suite. Le lecteur intéressé pourra se référer à [Knadler 92], [Ruemmler et Wilkes 94] ou [Ganger et Worthington 98] pour plus de détails.

Les unités de disques magnétiques traditionnelles sont composées d'un contrôleur de disque, d'une pile de disques superposés (ou volume) et d'un bras assurant le déplacement simultané d'un ensemble de têtes de lecture/écriture [Ruemmler et Wilkes 94]. Les disques d'un même volume tournent simultanément à vitesse constante. Une seule tête peut être active à la fois.

Soit v la vitesse de rotation et s le nombre de secteurs par piste. Le temps T d'accès à un groupe de secteurs consécutifs peut se décomposer comme suit [Cheng et Hurson 91] [Knadler 92] [Ruemmler et Wilkes 94] :

$$T = \Delta R + \Delta L + \Delta T \ .$$

- ΔR est le *temps de recherche*, c'est-à-dire le temps de déplacement du bras de disque pour que la tête se positionne sur le bon cylindre.
- ΔL est le *temps de latence*, c'est-à-dire le temps de rotation nécessaire pour amener le secteur désiré devant la tête de lecture/écriture.

- ΔT est le *temps de transfert*, c'est-à-dire le temps d'exécution de la lecture ou de l'écriture physique du nombre *s'* de secteurs considérés.

Le temps de transfert dépend directement du nombre de secteurs consécutifs lus en rapport du nombre total de secteurs par piste et de la vitesse de rotation du disque.

$$\Delta T = v \cdot \frac{s'}{s}$$

Par exemple, il faut 0,75 ms pour accéder à 3 secteurs sur une piste de 32 secteurs avec un disque tournant à 80 tours par seconde.

La Figure 2.14 illustre l'influence de la contiguïté sur les temps d'accès disque. Soit δt le temps de transfert unitaire pour un secteur. On note T_{cont} le temps d'accès à *s'* secteurs contigus et T_{disp} le temps d'accès à *s'* secteurs dispersés.

$$T_{comp} = \Delta R + \Delta L + s' \cdot \delta t \qquad T_{disp} = s' \cdot (\Delta R + \Delta L + \delta t)$$

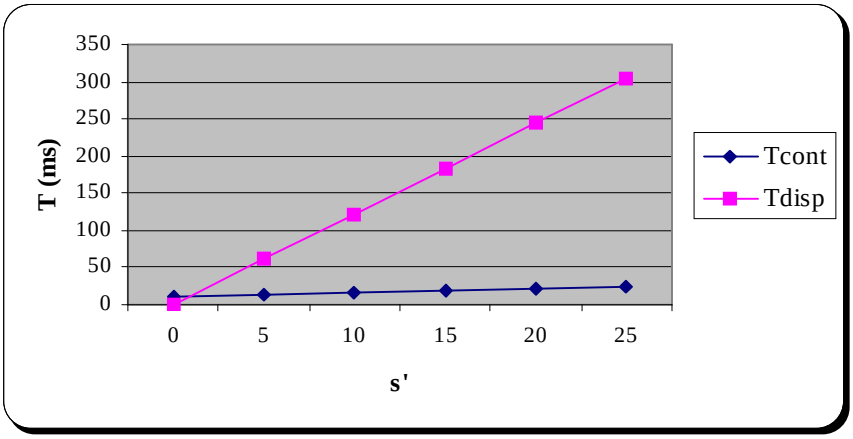

Figure 2.14 : Influence de la contiguïté sur les temps d'accès disque

La Figure 2.14 montre qu'il est beaucoup moins coûteux d'accéder à *s'* secteurs consécutifs plutôt qu'à *s'* secteurs dispersés sur le disque. Les chiffres indiqués sont basés sur les estimations moyennes suivantes : $\Delta R = 7{,}4$ ms, $\Delta L = 4{,}3$ ms et $\delta t = 0{,}5$ ms.

4.4. Synthèse

4.4.1. Besoins pour effectuer des simulations de SGBDOO

Les approches par simulation présentées à la Section 4.1 présentent trois points communs. Ce sont en fait les composants nécessaires à une étude de simulation pour évaluer les performances d'un SGBDOO.

- Un modèle de SGBDOO est indispensable pour servir de cadre aux simulations. Il doit être spécifié sous la forme d'une architecture fonctionnelle (cf. Figure 2.8, Figure 2.11 ou Figure 2.13).
- Une base d'objets doit être spécifiée. Deux approches sont possibles : soit la base de données est calquée sur une base existante utilisée dans le domaine étudié, soit il s'agit d'un modèle de base de données généré *ex nihilo* grâce à des règles précises de création.
- Des opérations à effectuer sur cette base d'objets doivent être définies. Ici encore, soit ces transactions peuvent être déduites d'une trace d'utilisation d'une application réelle, soit elles sont issues de modèles de transaction spécifiés par les concepteurs du modèle de simulation.

Il est important de noter que ces deux derniers points rejoignent la problématique de conception d'un banc d'essais. [Tsangaris et Naughton 91] utilise d'ailleurs la base de données du banc d'essais CluB-0, alors que [Darmont et Gruenwald 96b] utilise les opérations du banc d'essais HyperModel. Dans un souci de cohérence, il nous apparaît intéressant de systématiser cette démarche et d'intégrer un banc d'essais complet (base d'objets *et* opérations associées) dans un modèle de simulation de SGBDOO.

4.4.2. Critique des travaux effectués dans le domaine de la simulation

Nous reprenons dans cette section les critères énoncés à la Section 1 et qualifiant une bonne mesure de performance. Nous évaluons les études de simulation que nous venons de présenter en fonction de ces critères.

- *Pertinence :* Les modèles de simulation présentés sont focalisés sur un seul type de problème (le groupement d'objets, par exemple). De plus, ils n'exploitent à chaque fois qu'un seul type de SGBDOO, dont les caractéristiques ne sont pas toujours clairement spécifiées, alors que diverses architectures influençant les performances sont possibles (serveur d'objets, serveur de pages, etc. — cf. Chapitre 1 Section 1.2). La pertinence de ces modèles n'est donc pas excellente. Nous préconisons une approche plus générique, qui permettrait de modéliser le comportement de différents systèmes, d'y implanter des bases d'objets diverses et d'exécuter sur elles des transactions variées.
- *Fiabilité :* La spécification des modèles de simulation que nous avons étudiés varie en précision, si bien qu'il n'est pas toujours aisé de reproduire ces modèles à partir des publications disponibles. Ceci contrevient au critère de fiabilité. Il nous apparaît donc utile de proposer une méthodologie de modélisation qui permette, pas à pas, d'analyser un système, de spécifier un modèle de simulation pour ce système et pratiquement d'automatiser sa traduction dans un langage de programmation ou de simulation.

- *Faisabilité :* Les modèles de simulation étant des abstractions et des simplifications de la réalité, ils sont beaucoup plus faciles à concevoir que des prototypes. Par exemple, des problèmes purement techniques peuvent être considérés comme résolus ou occultés dans un modèle de simulation. Le critère de faisabilité est donc satisfait.
- *Indépendance :* Les modèles de simulation que nous venons d'évoquer ont été décrits dans des langages de simulation (comme PAWS) ou des langages de programmation (comme C++). Dans tous les cas, un autre langage aurait facilement pu être employé, le passage des spécifications du modèle à son implémentation étant quasi-automatique. Les modèles de simulation garantissent donc une parfaite indépendance.

Chapitre 3
Le banc d'essais OCB

Ce chapitre présente le premier outil d'évaluation des performances des SGBDOO que nous avons élaboré : le banc d'essais OCB (*Object Clustering Benchmark*) [Darmont et al. 98]. Comme son nom l'indique, OCB a dans un premier temps été conçu pour estimer plus spécifiquement l'impact de différentes politiques de regroupement sur les performances. Il trouve sa motivation dans les lacunes présentées par les bancs d'essais usuels dans ce domaine (cf. Chapitre 2, Section 3.6). OCB a par la suite été étendu de façon à être pleinement générique.

Nous présentons ici plus en détail les motivations qui nous ont poussé à concevoir un nouveau banc d'essais, plutôt que d'en utiliser un déjà existant où d'en adapter un à nos besoins. Nous détaillons également les spécifications d'OCB, tant au niveau de la base d'objets utilisée que de la charge qui lui est appliquée. Finalement, nous dressons une comparaison entre notre proposition et les bancs d'essais antérieurs.

1. Motivations pour un nouveau banc d'essais

1.1. OCB : un banc d'essais orienté regroupement

Le besoin d'un banc d'essais orienté regroupement s'est fait sentir en premier lieu parce que les bancs d'essais existants ne sont pas adaptés pour évaluer les performances de la majorité des algorithmes de regroupement d'objets. Ceci est particulièrement sensible pour les méthodes de groupement dites sémantiques, qui groupent les objets présentant certains types de liens logiques entre eux, plutôt que de les grouper en se basant sur des critères plus structurels (groupement par classe ou par hiérarchie composite, par exemple).

Les bancs d'essais généraux sont utiles pour évaluer les performances globales d'un SGBDOO, mais ne modélisent pas de façon inhérente une application spécifique, même si la plupart d'entre eux sont orientés vers les applications d'ingénierie (CAO, PAO, AGL…). Ils conviennent donc mal à l'évaluation des performances de politiques de groupement. En effet, certaines des opérations qu'ils proposent ne peuvent tout simplement bénéficier d'aucun regroupement (balayage de toute la base de données, accès aléatoires…) [Bancilhon et al. 92] [Bullat 96]. De plus, le regroupement d'objets est en

général très dépendant des données elles-mêmes, ce qui n'est pas pris en compte par les bancs d'essais synthétiques, qui présentent tous un schéma de base de données assez simple. Les bancs d'essais usuels ne prennent également pas en compte les opérations de réorganisation physique de la base de données, telles que celles qui sont mises en œuvre pour le groupement d'objets [Schreiber 94] [Banerjee et Gardner 95].

OCB a donc été structuré autour d'une base de données riche, incluant de nombreuses classes différentes (et donc différentes tailles d'instances, des références inter-objets nombreuses, etc.) et des types de références multiples permettant la conception de diverses hiérarchies entrelacées. Cette base de données est totalement générique. Par contre, la charge appliquée à cette base était dans un premier temps à dessein ciblée sur le groupement d'objets, toute transaction ne pouvant bénéficier d'un bon regroupement ayant été écartée.

1.2. OCB : un banc d'essais générique

La première version d'OCB était à la fois générique en ce qui concerne la base d'objets et orientée groupement en ce qui concerne les transactions. L'étape suivante a naturellement été d'étendre cet ensemble de transactions afin qu'OCB soit complètement générique et puisse être utilisé pour tester aussi bien les performances globales des SGBDOO que les performances spécifiques de méthodes d'optimisation.

Une telle flexibilité est atteinte grâce à un ensemble complet de paramètres qui rendent notre banc d'essais très adaptatif. Plusieurs sortes de bases d'objets très différentes peuvent être modélisées avec OCB, de même que différents types d'applications (modèles de charge) utilisant ces bases de données. Comme l'application de bases de données orientée objet typique n'existe pas, ceci est une fonctionnalité importante. OCB peut effectivement être paramétré facilement pour modéliser soit une application générique, soit une application plus ciblée sur un type de base d'objets et/ou d'application données. OCB est également très adaptable. En termes de taille et de complexité, une grande gamme de bases d'objets peut être générée. Le temps d'utilisation de la base peut également être réglé pour être court ou plus étendu. Finalement, les paramètres d'OCB ont été conçus pour être faciles à fixer.

Par ailleurs, il est prévu que la prochaine version d'OCB supporte des utilisateurs multiples, très simplement, en exécutant les transactions du banc d'essais de façon concurrente au sein de processus différents. Cela permettra de tester l'efficacité du contrôle de concurrence et le comportement des systèmes face à une charge plus importante.

2. Spécifications du banc d'essais OCB

2.1. Base d'objets d'OCB

La base de données d'OCB est à la fois riche et facile à générer, aisément paramétrable, et donc hautement générique. Son schéma est présenté sous la forme d'un diagramme de classes UML dans la Figure 3.1.

La base de données est constituée de *NC* classes, toutes dérivées de la même métaclasse *CLASS*. Une classe possède un identifiant logique unique *Class_ID* et est définie par deux paramètres : *MAXNREF*, le nombre maximum de références possibles pour cette classe et *BASESIZE*, la taille de base de la classe (qui est un incrément utilisé pour calculer la taille des instances, *InstanceSize*, au cours de la génération des graphes d'héritage lors de la création de la base). La clause UML « bind » de la Figure 3.1 indique que les classes sont instanciées d'après la métaclasse en utilisant les paramètres indiqués entre < >. Il existe donc une valeur de *MAXNREF* et de *BASESIZE* par classe.

Comme différentes références peuvent pointer sur la même classe, les relations 0-N, 1-N et M-N sont implicitement modélisées. Chacune de ces références (*CRef*) possède un type mémorisé dans l'attribut *TRef*. Il existe *NREFT* types de références différents. Un type de référence peut être, par exemple, une sorte d'héritage, une relation d'agrégation ou d'association, etc.

Finalement, au sein de chaque classe est maintenu un itérateur (attribut *Iterator*) qui permet d'accéder, *via* des pointeurs, à toutes les instances de cette classe.

Les objets (instances de la classe *OBJECT*) sont caractérisés par un identifiant logique unique *OID* et leur classe (grâce au pointeur *ClassPtr*). Chaque objet possède *ATTRANGE* attributs entiers accessibles et modifiables par les transactions du banc d'essais. Une chaîne de caractères de longueur *InstanceSize* (le *Filler*) simule la taille réelle que l'on souhaite donner à l'objet lorsqu'il sera stocké sur disque.

Une fois que le schéma de la base de données est instancié, chaque objet pointe (*via* les références *ORef*) sur au plus *MAXNREF* objets choisis dans l'itérateur (*Iterator*) de la classe référencée par la classe mère de l'objet en question, *via* la relation *CRef* correspondante. Pour chaque référence directe (identifiée par un lien *ORef*) d'un objet o_i vers un objet o_j, il existe de plus une référence inverse (*BackRef*) de o_j vers o_i.

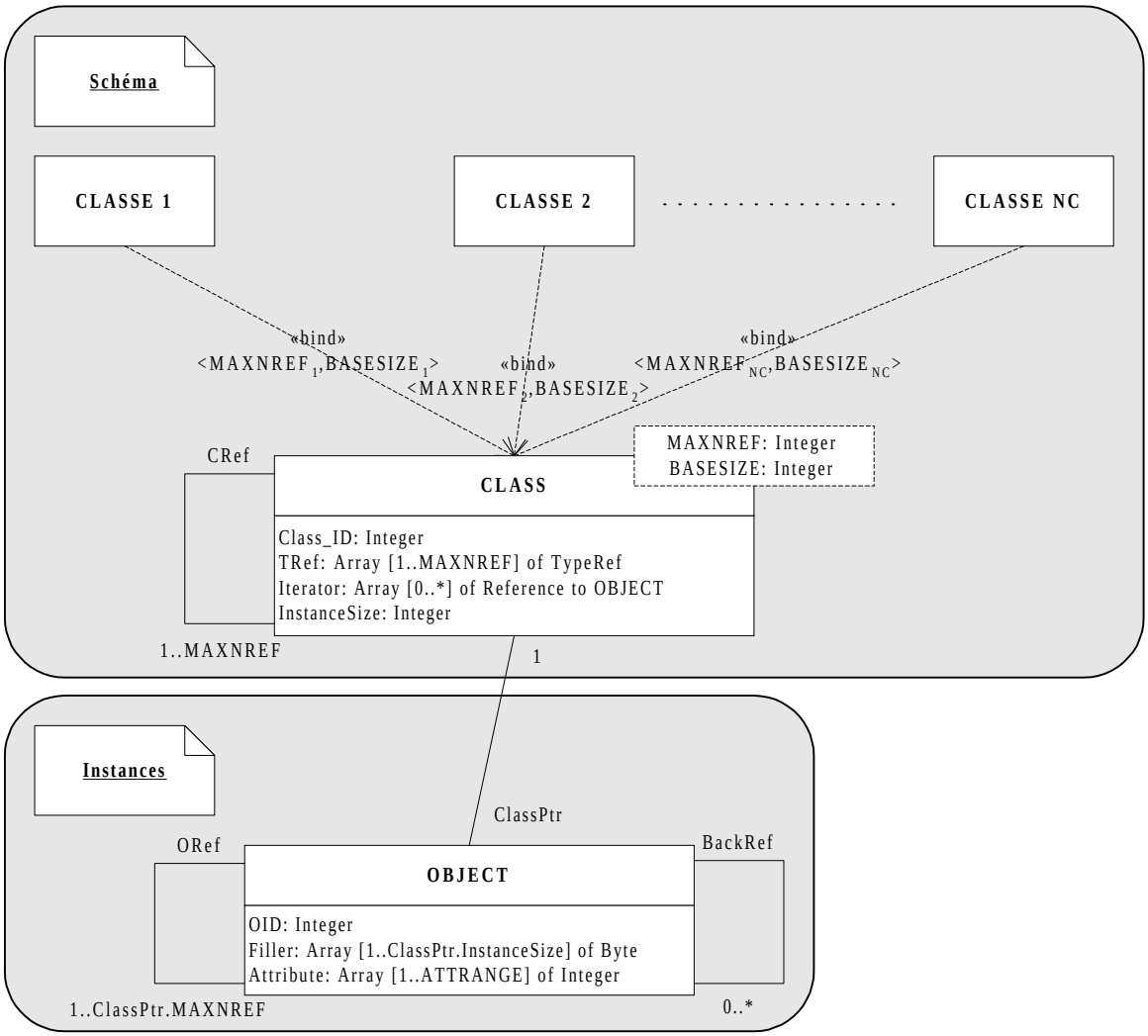

Figure 3.1 : Schéma de la base de données d'OCB

La génération de la base d'objets se déroule en trois étapes principales.

1) *Instanciation de la métaclasse* CLASS *en* NC *classes :* Dans un premier temps, les classes sont créées indépendamment les unes des autres, sans aucune référence inter-classes. Ensuite, chaque classe est reliée à *MAXNREF* autres classes. Le type de référence (*TRef*) est soit sélectionné aléatoirement suivant la distribution aléatoire *DIST1*, soit fixé *a priori*. Les identificateurs des classes référencées par la classe d'identifiant (*Class_ID*) *i* appartiennent à un intervalle [*i – CLOCREF, i + CLOCREF*] qui modélise une certaine localité de référence, telle qu'elle est introduite dans OO1 [Cattell 91] (cf. Chapitre 2, Section 3.2.1), mais au niveau des classes. La sélection des classes référencées est soit aléatoire, selon la distribution aléatoire *DIST2*, soit fixée *a priori*. Les références nulles sont autorisées.

2) *Vérification de la cohérence de la base :* Les cycles et les incohérences éventuels induits par la sélection aléatoire des références sont supprimés dans les graphes qui les interdisent explicitement. Par exemple, les hiérarchies d'héritage ou de composition doivent être acycliques. Les types de référence « héritage » et

« agrégation » sont donc étiquetés de façon à ce que, lors de cette phase de vérification structurelle, toutes les références appartenant à ces types soient parcourues.

3) *Instanciation des* NC *classes en* NO *objets* OBJECT *:* Dans un premier temps, les objets sont générés indépendamment les uns des autres, sans aucune référence inter-objets. Leur classe est sélectionnée aléatoirement selon la distribution aléatoire *DIST3*. Ensuite, chaque objet est relié à autant d'autres objets que la valeur du paramètre *MAXNREF* de sa classe. Les identificateurs des objets référencés par l'objet d'indentifiant (*OID*) *i* appartiennent à un intervalle [*i* – *OLOCREF*, *i* + *OLOCREF*] qui modélise la localité de référence introduite dans le banc d'essais OO1, au niveau des instances. La sélection aléatoire des références d'objets suit la distribution aléatoire *DIST4*. Les références inverses (*BackRef*) sont instanciées lorsque les références directes sont fixées.

L'algorithme complet de génération de la base de données est fourni dans la Figure 3.2. Les nombres aléatoires employés sont générés à l'aide du générateur aléatoire de Lewis-Payne [Lewis et Payne 73], qui est l'un des meilleurs générateurs de nombres pseudoaléatoires actuellement disponibles, grâce à sa très grande période. Les paramètres qui définissent la base d'objets d'OCB sont récapitulés dans la Table 3.1.

```
// Schéma

// Classes

Pour i = 1, NC faire
     Pour j = 1, MAXNREF (i) faire
          Class (i).TRef (j) = RAND DIST1 (1, NREFT)
     Fin pour
     Class (i).InstanceSize = BASESIZE (i)
Fin Pour

// Références inter-classes

Pour i = 1, NC faire
     Pour j = 1, MAXNREF (i) faire
          infclass = i – CLOCREF
          Si infclass < 1 alors infclass = 0
          supclass = i + CLOCREF
          Si infclass > NC alors supclass = NC
          Class (i).CRef (j) = RAND DIST2 (infclass, supclass)
     Fin pour
Fin pour

// Cohérence des graphes pour les hiérarchies sans cycle

Pour i = 1, NC faire
     Pour j = 1, MAXNREF (i) faire
          Si Sans_Cycle (Class (i).TRef (j)) alors
               // Parcourir le graphe de la classe CRef (j) en suivant les
               // références de type TRef (j)
               Si Class (i) appartient au graphe ou un cycle est détecté alors
                    Class (i).CRef (j) = NIL
               Sinon
                    Si Heritage (Class (i).TRef (j)) alors
```

```
                        // Parcourir le graphe d'héritage de la classe CRef (j)
                        // et ajouter BASESIZE (i) à InstanceSize pour chaque
                        // sous-classe
                    Fin si
               Fin si
          Fin si
     Fin pour
Fin pour

// Instances

// Objets

Pour i = 1, NO faire
     Object (i).ClassPtr = RAND DIST3 (1, NC)
     Object (i).ClassPtr.Ajoute_Dans_Iterateur (Objet (i))
Fin pour

// Références inter-objets

Pour i = 1, NC faire
     Pour j = 1, Class (i).Retourne_Compte_Iterateur() faire
          Pour k = 1, MAXNREF (i) faire
               trouve = FAUX
               essais = 0
               Tant que (trouve = FALSE) et (essais < MAXRETRY) faire
                    l =
                       RAND DIST4 (1, Class (CRef(k)).Retourne_Compte_Iterateur())
                    Si Class (CRef(k)).Iterator(l) >=
                       Class (i).Iterator (j) - OREFLOC
                    Et Class (CRef(k)).Iterator(l) <=
                       Class (i).Iterator (j) + OREFLOC
                    Alors trouve = VRAI
                    Sinon essais = essais + 1
                    Fin si
               Fin tant que
               Si trouve = VRAI alors
                    Iterator (i).Object (j).ORef (k) =
                      Class (CRef(k)).Iterator (l)
                    Ajoute_BackRef (Class (CRef(k)).Iterator (l),
                      Iterator (i).Object (j))
               Sinon
                    Iterator (i).Object (j).ORef (k) = NIL
               Fin si
          Fin pour
     Fin pour
Fin pour
```

Figure 3.2 : Algorithme de génération de la base d'objets d'OCB

Paramètre	Libellé	Valeur par défaut
NC	Nombre de classes dans la base de données	50
MAXNREF (i)	Nombre maximum de références, par classe	10
BASESIZE (i)	Taille de base des instances, par classe	50 octets
NO	Nombre total d'objets	20000
NREFT	Nombre de types de références (héritage, agrégation, etc.)	4
ATTRANGE	Nombre d'attributs entiers dans un objet	1
CLOCREF	Localité de référence au niveau des classes	NC
OLOCREF	Localité de référence au niveau des instances	NO
MAXRETRY	Nombre maximum d'essais lors de la connexion des objets	3
DIST1	Distribution aléatoire des types de références	Uniforme
DIST2	Distribution aléatoire des références de classes	Uniforme
DIST3	Distribution aléatoire des objets dans les classes	Uniforme
DIST4	Distribution aléatoire des références d'objets	Uniforme

Table 3.1 : Paramètres définissant la base de données d'OCB

2.2. Charge d'OCB

2.2.1. Transactions

Le noyau de la charge d'OCB est constitué d'un ensemble d'opérations (les *parcours*) qui explorent les effets du regroupement d'objets. Afin de rendre notre banc d'essais générique, nous y avons ajouté dans un deuxième temps d'autres types d'opérations qui, elles, ne bénéficient que peu ou pas d'efforts de regroupement des objets (par exemple, les opérations de création ou de mise à jour, ou encore les recherches par valeur et les balayages séquentiels du banc d'essais HyperModel [Anderson et al. 90]). La hiérarchie des transactions que nous avons utilisées pour constituer la charge d'OCB est présentée en Figure 3.1 sous la forme d'un diagramme de classes UML.

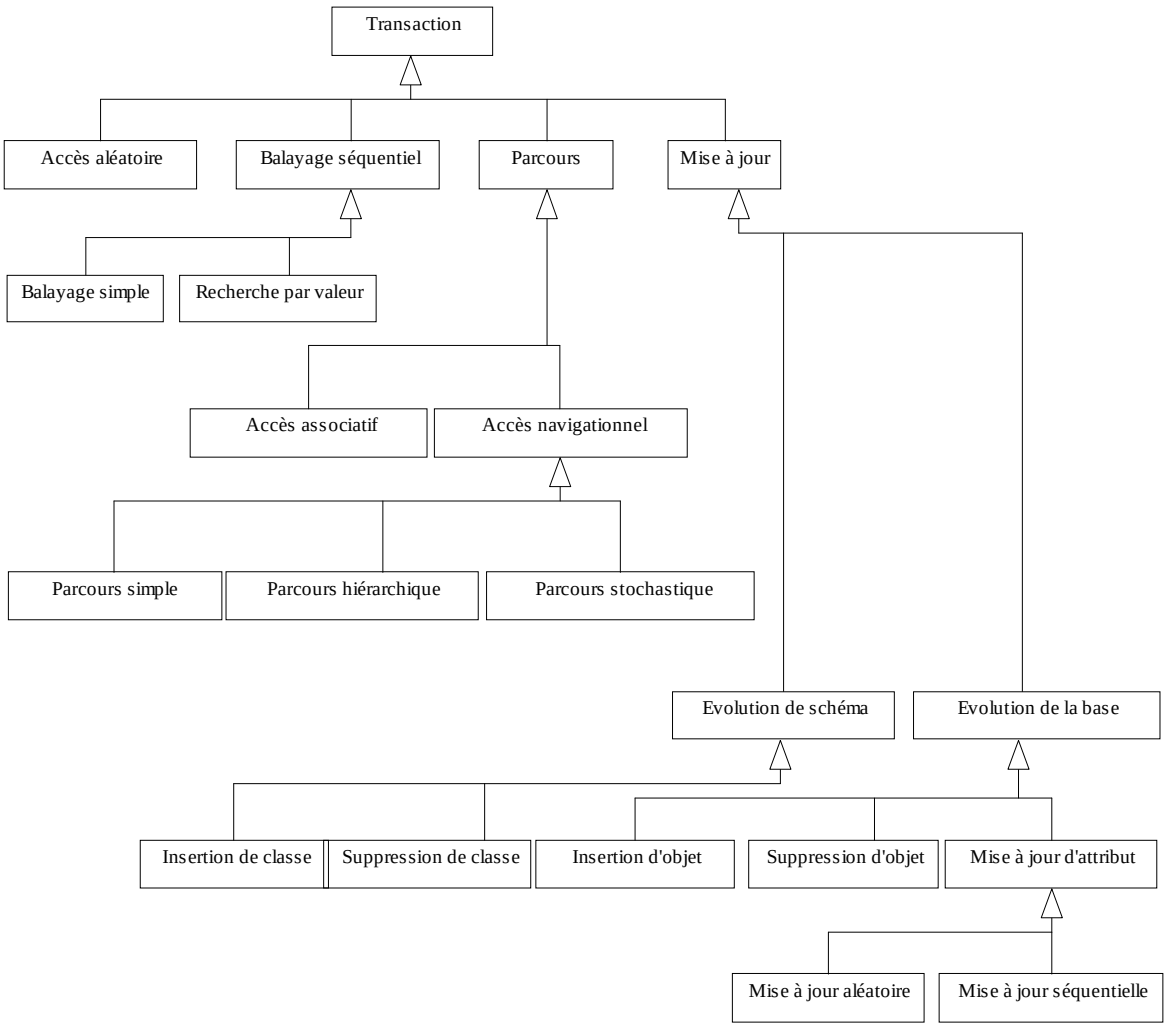

Figure 3.3 : Classes de transactions d'OCB

- *Accès aléatoire :* Accès à *NRND* objets sélectionnés au hasard selon la loi de distribution aléatoire *DIST5*.

- *Balayage séquentiel :* Lecture des instances d'une classe sélectionnée au hasard selon la loi de distribution aléatoire *DIST6* (*Balayage simple*). Une *Recherche par valeur* effectue en sus un test sur la valeur de *NTEST* attributs, pour chaque instance accédée.
- *Parcours :* Les opérations de parcours sont divisées en deux types : les *Accès associatifs* (ou *Accès ensemblistes*) et les *Accès navigationnels*, qui ont été associés empiriquement par [McIver et King 94] à des parcours en largeur d'abord et des parcours en profondeur d'abord, respectivement. Les accès navigationnels sont subdivisés en trois catégories : les *Parcours simples* (en profondeur d'abord), les *Parcours hiérarchiques* qui suivent toujours le même type de référence et, finalement, les *Parcours stochastiques* dans lesquels la prochaine référence à suivre est déterminée aléatoirement. Les parcours stochastiques approchent des chaînes de Markov, qui sont de bons modèles de requêtes pour certaines applications [Tsangaris et Naughton 92]. À chaque itération, la probabilité de suivre la référence numéro N de l'objet courant est $p(N) = 1/2^N$. Chaque type de parcours démarre à partir d'un objet racine sélectionné au hasard selon la distribution aléatoire *RAND7* et procède jusqu'à une profondeur prédéterminée dépendant du type de parcours. Tous ces parcours peuvent être inversés pour suivre les références inverses (*BackRef*) des objets et « remonter » les graphes. Le code de l'implémentation en C++ du parcours hiérarchique est fourni à titre d'exemple dans la Figure 3.4.
- *Mise à jour :* Les opérations de mise à jour sont également subdivisées en deux catégories. Les *Évolutions de schéma* gèrent l'insertion et la suppression des objets de classe *CLASS* (une insertion/suppression à la fois). Une classe à effacer est sélectionnée au hasard selon la distribution aléatoire *DIST8*. Les *Évolutions de la base* administrent l'insertion et la suppression des objets de classe *OBJECT*. Un objet à effacer est sélectionné au hasard selon la distribution aléatoire *DIST9*. Finalement, les *Mises à jour d'attribut* permettent des changements de valeur pour les attributs des objets. Ces objets sont soit choisis aléatoirement (*Mise à jour aléatoire* de *NUPDT* objets sélectionnés au hasard selon la distribution aléatoire *DISTA*), soit sont les instances d'une classe sélectionnée au hasard selon la distribution aléatoire *DISTB* (*Mise à jour séquentielle*).

2.2.2. Protocole d'exécution

L'exécution des transactions par chaque client (dans la version multi-utilisateurs à venir d'OCB) est organisée selon le protocole suivant :

1) exécution à froid de *COLDN* transactions dont le type est déterminé au hasard, suivant des probabilités d'occurrence prédéterminées. Le but de cette étape est de

remplir le cache afin d'observer le comportement réel (stationnaire) du système testé ;

2) exécution à chaud de *HOTN* transactions. Seuls les transactions exécutées à chaud sont prises en compte dans l'évaluation de performance.

Un temps de latence *THINK* peut être introduit entre chaque exécution de transaction.

De plus, l'exécution complète du banc d'essais peut être répliquée de façon à lancer le même ensemble de transactions sur différentes bases d'objets aléatoirement générées. Cette particularité permet un traitement statistique (calcul de valeurs moyennes et d'écarts types) bien plus significatif qu'une seule et unique mesure.

Les paramètres définissant la charge d'OCB sont résumés dans la Table 3.2.

2.3. Mesure des performances

Les mesures de performance prises en compte par OCB sont les suivantes.

- *Temps de réponse :* C'est le temps d'exécution moyen d'une transaction. Il doit être calculé globalement et pour chaque type de transaction individuellement. Le temps de réponse doit être mesuré côté client (dans un contexte client-serveur) à l'aide des fonctions système standards, comme les fonctions `time()` ou `getrusage()` des langages C et C++. La réplication des transactions permet de compenser l'imprécision éventuelle de ces primitives. Si le nombre de transactions sélectionné est suffisant pour qu'une exécution du banc d'essais soit assez longue, l'absence de ces fonctions système peut être compensée par un chronométrage manuel (comme cela est d'ailleurs préconisé pour le banc d'essais OO1 [Cattell 91]).
- *Débit :* Le débit du système est le nombre de transactions traitées à la seconde. De même que le temps de réponse, le débit doit être calculé globalement et pour chaque type de transaction individuellement. Le temps de traitement total doit aussi être mesuré côté client (dans un contexte client-serveur) à l'aide des fonctions système standards ou à défaut, manuellement.
- *Nombre d'objets accédés :* Il peut être intéressant de connaître le nombre d'objets accédés par telle ou telle transaction afin de se rendre compte de la portée d'une opération donnée. Il peut également être utile de savoir, globalement, quelle est la fraction de la base de données qui a été accédée durant une exécution du banc d'essais. La collecte de ces statistiques d'utilisation doit être incluse dans le code d'OCB, au niveau de l'exécution des transactions.

```
void ParcoursHierarchique (Objet *Cour, int ProfCour, int Prof, int Type2Ref) {

  int i;
  // Si la profondeur n'est pas atteinte, poursuivre le parcours
  if (ProfCour<Prof) {
    for (i=0; i<Cour->ClassPtr->MAXNREF; i++) {
      // Toutes les références sont considérées
      if ((Cour->ORef[i]!=NULL)
      && (Cour->Classe->TRef[i]==Type2Ref)) {
        // Accès à toutes les références non nulles de type Type2Ref
        ParcoursHierarchique(Cour->ORef[i], ProfCour+1, Prof, Type2Ref);
      }}}
};
```

Figure 3.4 : Code C++ du parcours hiérarchique dans OCB

Paramètre	Libellé	Valeur par défaut
NRND	Nombre d'objets accédés par les Accès aléatoires	50
NTEST	Nombre d'attributs testés dans les Recherches par valeur	1
SETDEPTH	Profondeur des accès ensemblistes	3
SIMDEPTH	Profondeur des parcours simples	3
HIEDEPTH	Profondeur des parcours hiérarchiques	5
STODEPTH	Profondeur des parcours stochastiques	50
NUPDT	Nombre d'objets mis à jour dans les Mises à jour aléatoires	50
RAND5	Distribution aléatoire des objets des Accès aléatoires	Uniforme
RAND6	Distribution aléatoire des classes des Balayages séquentiels	Uniforme
RAND7	Distribution aléatoire des objets racines de Parcours	Uniforme
RAND8	Distribution aléatoire des classes des Évolutions de schéma	Uniforme
RAND9	Distribution aléatoire des objets des Évolutions de la base	Uniforme
RANDA	Distribution aléatoire des objets des Mises à jour aléatoires	Uniforme
RANDB	Distribution aléatoire des classes des Mises à jour séquentielles	Uniforme
PRND	Probabilité d'occurrence d'un Accès aléatoire	0,1
PSCAN	Probabilité d'occurrence d'un Balayage simple	0,05
PRANGE	Probabilité d'occurrence d'une Recherche par valeur	0,05
PSET	Probabilité d'occurrence d'un accès associatif	0,2
PSIMPLE	Probabilité d'occurrence d'un parcours simple	0,2
PHIER	Probabilité d'occurrence d'un parcours hiérarchique	0,2
PSTOCH	Probabilité d'occurrence d'un parcours stochastique	0,1
PCINSERT	Probabilité d'occurrence d'une Insertion de classe	0,005
PCDEL	Probabilité d'occurrence d'une Suppression de classe	0,005
POINSERT	Probabilité d'occurrence d'une Insertion d'objet	0,02
PODEL	Probabilité d'occurrence d'une Suppression d'objet	0,02
PRNDUP	Probabilité d'occurrence d'une Mise à jour aléatoire	0,025
PSEQUP	Probabilité d'occurrence d'une Mise à jour séquentielle	0,025
COLDN	Nombre de transactions exécutées à froid	1000
HOTN	Nombre de transactions exécutées à chaud	10000
THINK	Temps de latence moyen entre deux transactions	0
CLIENTN	Nombre de clients (utilisateurs de la base)	1
RSEED	Germe du générateur aléatoire	Germe par défaut

Table 3.2 : Paramètres définissant la charge d'OCB

- *Nombre d'entrées-sorties effectuées :* Comme la plupart des techniques d'optimisation des performances pour SGBDOO agissent en réduisant le nombre des entrées-sorties, il est intéressant de les détecter lors de l'exécution d'OCB. De plus, même s'il existe en général une corrélation entre le nombre d'entrées-sorties

effectuées et le temps de réponse, elle n'est pas toujours facile à établir. Il est donc important de disposer des deux mesures. Nous distinguons les entrées-sorties nécessaires à l'exécution des transactions de la surcharge en entrées-sorties engendrée par une technique de regroupement d'objets, par exemple. Les mesures du nombre d'entrées-sorties peuvent être obtenues grâce à des fonctions système (comme la fonction `getrusage()` des langages C et C++) ou des statistiques fournies par le SGBD (O_2 fournit de telles statistiques, par exemple).

3. Comparaison d'OCB avec les bancs d'essais existants

3.1. Généricité d'OCB

Comme nous prétendons que notre banc d'essais est générique, ce dernier doit non seulement permettre de modéliser divers types de bases de données et d'applications, mais aussi être capable d'imiter le comportement des bancs d'essais existants.

L'auteur de Justitia (cf. Chapitre 2, Section 3.5) assure également que son banc d'essais possède cette faculté [Schreiber 94], moyennant un paramétrage adéquat, mais ne donne pas d'exemple concret permettant de prouver cette affirmation. Il paraît effectivement facile de retrouver la base d'objets d'OO1 [Cattell 91], mais le nombre limité de relations possibles dans Justitia (héritage et agrégation uniquement) rend difficile la reproduction des bases d'HyperModel [Anderson et al. 90] ou OO7 [Carey et al. 93]. Par exemple, la relation d'association M-N d'HyperModel (cf. Chapitre 2, Section 3.3.1) est impossible entre deux objets *Node* de Justitia, qui ne peuvent être reliés que par une relation de composition. Il en est de même pour les relations M-N « composant partagé » et « composant non partagé » d'OO7 (cf. Chapitre 2, Section 3.4.1), entre autres.

Nous soulignons donc la généricité de la base de données d'OCB en montrant comment notre banc d'essais peut être paramétré pour générer des bases d'objets très similaires à celles d'OO1, d'HyperModel et de Justitia, respectivement (Table 3.3 et Table 3.4).

En terme de charge, cependant, la démonstration est plus difficile. En effet, OO7, en particulier, propose une large gamme d'opérations complexes. Nous avons écarté dans OCB les plus complexes d'entre elles, car elles rendaient le banc d'essais plus compliqué (et donc plus difficile à implémenter) sans fournir de renseignement complémentaire facile à interpréter en ce qui concerne les performances. Cependant, le comportement d'OO1, d'HyperModel et de Justitia peuvent être facilement simulés avec OCB. La Table 3.5 montre la correspondance entre les opérations de ces bancs d'essais et celles d'OCB. Nous avons indiqué pour les opérations complexes d'OO7, les opérations d'OCB qui, combinées, pourraient leur être équivalentes. Ces dernières sont signalées par des tildes (~). La requête Q8 d'OO7 n'a pas d'équivalent.

Paramètre OCB	OO1	HyperModel
NC	2	3
MAXNREF (i)	Composants : 3 Connexions : 2	5 (*Parent/Children*) + 5 (*PartOf/Part*) + *NO* (*RefTo/RefFrom*) + 1 (Spécialisation)
BASESIZE (i)	Composants : 50 octets Connexions : 50 octets	Node : 20 octets TextNode : 1000 octets FormNode : 20008 octets
NO	20000 composants + 60000 connexions	3906 Nodes + 125 FormNodes + 15500 TextNodes
NREFT	3	4
CREFLOC	*NC*	*NC*
OREFLOC	*RefZone*	Niveau *k+1* dans la hiérarchie *Parent/Children*
DIST1	Constante (non aléatoire)	Constante (non aléatoire)
DIST2	Constante (non aléatoire)	Constante (non aléatoire)
DIST3	Constante (non aléatoire)	Constante (non aléatoire)
DIST4	Uniforme	Uniforme

Table 3.3 : Paramétrage d'OCB pour retrouver les bases d'objets d'OO1 et HyperModel

3.2. Comparaison selon les critères de [Gray 93]

En concevant le banc d'essais OCB, nous voulions principalement pallier deux inconvénients des bancs d'essais existants, qui ne satisfaisaient pas nos besoins :

1) *le manque de généricité :* les bancs d'essais existants, malgré des efforts pour tendre vers plus de généricité, restent clairement orientés dans leurs structures de données et leurs opérations vers les applications d'ingénierie (CAO, PAO, AGL…) ;

2) *l'inadéquation à l'évaluation des performances des techniques de regroupement d'objets* en raison de la structure de leur base de données, insuffisamment riche au niveau des références inter-objets, ainsi que de la présence d'opérations ne pouvant bénéficier d'un groupement quel qu'il soit.

Nous avons répondu à ces problèmes en proposant un banc d'essais totalement paramétrable, tant au niveau de la base d'objets que des transactions proposées, ce qui permet à OCB d'être générique d'une part, mais aussi de s'adapter à des besoins plus ciblés d'autre part, par exemple en définissant une base d'objets très typée ou en écartant des transactions indésirables au vu du but recherché. Les répercussions de ces choix de conception sur les critères proposés par [Gray 93] sont les suivantes.

Paramètre OCB	**OO7**	**Justitia**
NC	10	6
MAXNREF (i)	Objet conceptuel : 0 Composant atomique : 20 Connexion : 18 Objet composite : *NumAtomicPerComp* + 8 Document : 1 Manuel : 1 Assemblage : 2 Assemblage complexe : *NumAssmPerAssm* + 2 Assemblage de base : *NumComPerAssm* x 2 + 1 Module : $\sum_{i=0}^{NumAssmLevels} NumAssmPerAssm^{i}$	Database Entry : 0 Node : 2 CO : 3 PO : *PO_ATT_SIZE* + 3
BASESIZE (i)	Objet conceptuel : 18 octets Composant atomique : 12 octets Connexion : 14 octets Objet composite : 0 Document : *DocumentSize* + 44 octets Manuel : *ManualSize* + 48 octets Assemblage : 0 Assemblage complexe : 0 Assemblage de base : 0 Module : 0	Database entry : 4 octets PO : 0 Node : 4 octets CO : 0 DO : *DO_ATT_SIZE* octets SO : *SO_ATT_SIZE* octets
NO	*NumModules* modules + *NumModules* manuels + $\sum_{i=0}^{NumAssmLevels-1} NumAssmPerAssm^{i}$ assemblages compl. + $NumPerAssm^{NumAssmLevels}$ assemblages de base + *NumCompPerModule* objets composites + *NumCompPerModule* documents + *NumAtomicPerComp* . *NumCompPerModule* composants atomiques + *NumAtomicPerComp* . *NumCompPerModule* . *NumConnPerAtomic* connexions	*SECTION* . *MAXWIDTH* . *MAXLEVEL*
NREFT	12	3
CREFLOC	*NC*	*NC*
OREFLOC	*NO*	*NO*
DIST1	Constante (non aléatoire)	Constante (non aléatoire)
DIST2	Constante (non aléatoire)	Constante (non aléatoire)
DIST3	Constante (non aléatoire)	Constante (non aléatoire)
DIST4	Constante + Uniforme	Constante (non aléatoire)

Table 3.4 : Paramétrage d'OCB pour retrouver les bases d'objets d'OO7 et Justitia

- *Pertinence :* Comme nous l'avons montré dans la Section 3.1, nous avons inclus dans OCB toutes les transactions existantes dans les bancs d'essais conçus antérieurement, hormis les plus élaborées des opérations d'OO7. De surcroît, la base d'objets d'OCB est plus riche que celles des bancs d'essais existants, notamment grâce à la prise en compte de types de références variés permettant de modéliser plusieurs hiérarchies entrelacées au sein d'une même base. Le critère de pertinence est à notre avis satisfait.

Banc d'essais	Opération	Opération correspondante dans OCB
OO1	Recherche	Accès aléatoire
	Parcours	Parcours hiérarchique
	Insertion	Insertion d'objet
HyperModel	Recherche par nom	Accès aléatoire
	Recherche par valeur	Recherche par valeur
	Recherche par groupe	Accès associatif ou Parcours simple
	Recherche par référence	Accès associatif ou Parcours simple à rebours
	Balayage séquentiel	Balayage simple
	Fermeture	Parcours hiérarchique
	Mise à jour	Mise à jour aléatoire
OO7	T1	Parcours hiérarchique
	T2	~ Parcours hiérarchique avec mise à jour
	T3	~ Parcours hiérarchique avec mise à jour
	T6	Accès associatif
	T8	Recherche par valeur
	T9	Recherche par valeur
	TCU	~ Parcours hiérarchique + second Parcours hiérarchique avec mise à jour
	Q1	Accès aléatoire
	Q2	Recherche par valeur
	Q3	Recherche par valeur
	Q4	Parcours hiérarchique
	Q5	Recherche par valeur
	Q7	Balayage simple
	Q8	—
	Insertion	Insertion d'objet
	Suppression	Suppression d'objet
Justitia	Lecture simple	Accès aléatoire, Balayage simple ou Parcours
	Mise à jour simple	Mise à jour
	Lecture étendue	Accès aléatoire, Balayage simple ou Parcours
	Mise à jour étendue	Mise à jour

Table 3.5 : Correspondance des opérations entre les bancs d'essais existants et OCB

- *Faisabilité :* Nous nous sommes attachés dans ce chapitre à fournir des spécifications claires et précises pour notre banc d'essais, afin de permettre à quiconque de le comprendre, de l'implémenter facilement et de l'utiliser. Nous sommes conscients des difficultés légales pouvant être rencontrées par les concepteurs de bancs d'essais (cf. Chapitre 2, Section 3.6). Cependant, nous estimons important de contribuer à l'effort d'évaluation des performances des SGBDOO de cette manière. Nous espérons en effet qu'OCB puisse au moins être utilisé à des fins de recherche, soit pour évaluer les performances d'un prototype de SGBDOO ou d'une configuration matérielle, soit pour tester plus particulièrement les effets d'une technique d'optimisation donnée. Nous espérons pouvoir améliorer OCB grâce aux critiques de la communauté scientifique, aussi avons-nous essayé de satisfaire au mieux le critère de faisabilité.
- *Adaptabilité :* OCB est un banc d'essais très flexible, grâce à un ensemble complet de paramètres faciles à fixer. Sa base d'objets peut adopter différentes tailles et différents niveaux de complexité et ses transactions variées permettent de modéliser un grand panel d'applications. Le critère d'adaptabilité est donc satisfait.

- *Portabilité :* OCB a été utilisé pour évaluer les performances des systèmes Texas [Singhal et al. 92] et O_2 [Deux et al. 91] (cf. Chapitre 7). Ces deux implémentations ont été effectuées avec le langage C++. OCB a également été inclus dans des modèles de simulation (cf. Chapitre 4) écrits avec le logiciel QNAP2 [Simulog 95] (qui supporte un langage de programmation non orienté objet proche de Pascal) et le package de simulation DESP-C++ présenté au Chapitre 5. Dans tous ces cas, le code d'OCB est compact (il occupe par exemple moins de 1000 lignes de code QNAP2 ; les versions C++ font moins de 1500 lignes), ce qui concourt à sa portabilité.

Nous récapitulons dans la Table 3.6 les caractéristiques des bancs d'essais existants (cf. Chapitre 2, Section 3.6) en y ajoutant celles d'OCB, en fonction de ces critères.

	Pertinence	***Faisabilité***	***Adaptabilité***	***Portabilité***
OO1	**– –**	**++**	**–**	**++**
HyperModel	**+**	**–**	**– –**	**+**
OO7	**++**	**–**	**–**	**+**
Justitia	**–**	**– –**	**+**	**+**
OCB	**++**	**+**	**++**	**+**

Point fort : + *Point très fort : ++* *Point faible : –* *Point très faible : – –*

Table 3.6 : Tableau comparatif des bancs d'essais existants + OCB

Chapitre 4
Le modèle de simulation VOODB

Ce chapitre présente le second outil d'évaluation des performances des SGBDOO que nous proposons : le modèle de simulation VOODB (*Virtual Object Oriented Database*). Nous nous basons sur le postulat que l'approche simulation présente divers avantages par rapport à l'approche classique par banc d'essais (faible coût, modélisation *a priori*, souplesse…). Partant des limitations constatées des modèles de simulation de SGBDOO dédiés trouvés dans la littérature (cf. Chapitre 2, Section 4.2.2), nous préconisons la conception de modèles de simulation plus génériques et l'utilisation d'une approche plus systématique de l'analyse des systèmes à modéliser, afin d'obtenir des modèles de simulation plus fiables. Nous recommandons également l'intégration du banc d'essais OCB (cf. Chapitre 3) au sein de nos modèles de simulation afin de disposer d'un outil d'évaluation fiable et cohérent.

Nous présentons dans ce chapitre une méthodologie de modélisation complète adaptée au domaine des bases de données orientées objet, qui étend celle proposée dans [Darmont et al. 95] [Darmont et al. 97], ainsi que l'application que nous en avons faite pour produire le modèle de simulation VOODB.

1. Méthodologie de modélisation

1.1. Nécessité d'une telle approche

Les SGBDOO sont des systèmes complexes, si bien que modéliser leur fonctionnement en vue d'étudier leurs performances peut s'avérer être également une tâche complexe. Un modèle peut être défini de la façon suivante [Minsky 68] : pour un observateur A, β est un modèle du système B si A peut apprendre à partir de β quelque chose d'utile sur le fonctionnement de B. En pratique, la construction d'un modèle (l'activité de modélisation) s'effectue soit à partir d'observations du système, s'il existe (modélisation *a po steriori*), soit à partir d'une description de la manière dont il va fonctionner (cahier des charges, spécifications), s'il n'existe pas encore (modélisation *a priori*).

La construction d'un modèle d'un SGBDOO (existant ou non) n'est pas évidente. C'est pourquoi nous préconisons l'utilisation d'une méthodologie de modélisation permettant d'analyser le fonctionnement du SGBDOO étudié de façon systématique, étape par étape, et de produire des modèles fiables pour ce système. Les spécialistes en modélisation et en simulation de différents domaines (systèmes de productions, systèmes hospitaliers, gestion du trafic routier…) utilisent couramment ce genre d'approche [Balci et Nance 92] [Gourgand et Kellert 92] [Kellert et al. 97].

L'objectif de cette démarche est de guider les experts des bases de données, qui sont les plus à même de connaître le fonctionnement et la complexité des modèles, et d'assurer la liaison avec les experts de la modélisation. Elle préconise l'utilisation d'un processus de modélisation, qui est l'élément crucial de la méthodologie (cf. Section 1.2), et une vision systémique pour décomposer et structurer le système à modéliser [Fleury et al. 98].

Une méthodologie de modélisation intègre en fait une grande partie du savoir-faire des spécialistes en simulation. Elle permet de déduire facilement et sans ambiguïté les spécifications de la structure et du comportement d'un système. Elle constitue un guide tout au long du processus de modélisation afin de produire des modèles fiables et de garantir leur traduction automatique dans des langages de simulation ou de programmation. L'utilisation d'une telle méthodologie de modélisation, plutôt qu'une approche empirique, permet des gains de temps importants lors de la phase d'analyse du système.

1.2. Processus itératif de modélisation

Afin de disposer d'une méthodologie de modélisation appropriée à notre domaine, les bases de données orientées objet, nous avons adapté à nos besoins propres la démarche présentée dans [Gourgand et Kellert 92], qui est utilisée principalement pour la modélisation de systèmes de production. Cette méthodologie est basée sur une approche orientée objet permettant la construction consécutive d'un *modèle de connaissance* et de *modèles d'évaluation* pour un système donné [Sargent 79] [Nance 81] [Jackson 83] [Gourgand 84] [Sargent 91]. Ce processus est dénommé « Analyse-Spécification-Conception-Implantation » (ASCI). La phase d'analyse et de spécification concerne le modèle de connaissance et la phase de conception et d'implantation le modèle d'évaluation.

- Un *modèle de connaissance* d'un SGBDOO est une formalisation en langage naturel et/ou graphique de la structure et du fonctionnement de ce système.
- Un *modèle d'évaluation* est une traduction du modèle de connaissance dans un formalisme mathématique ou dans un langage de programmation. Ce modèle doit être directement exploitable et fournir des indications sur les performances du système modélisé.

L'exploitation de ces modèles (appelée processus de modélisation) est itérative [Walliser 77] [Gourgand 84]. Elle est divisée en quatre phases (Figure 4.1). Les deux premières consistent à recueillir et à formaliser la connaissance concernant le système. Les deux secondes concernent l'exploitation de cette connaissance.

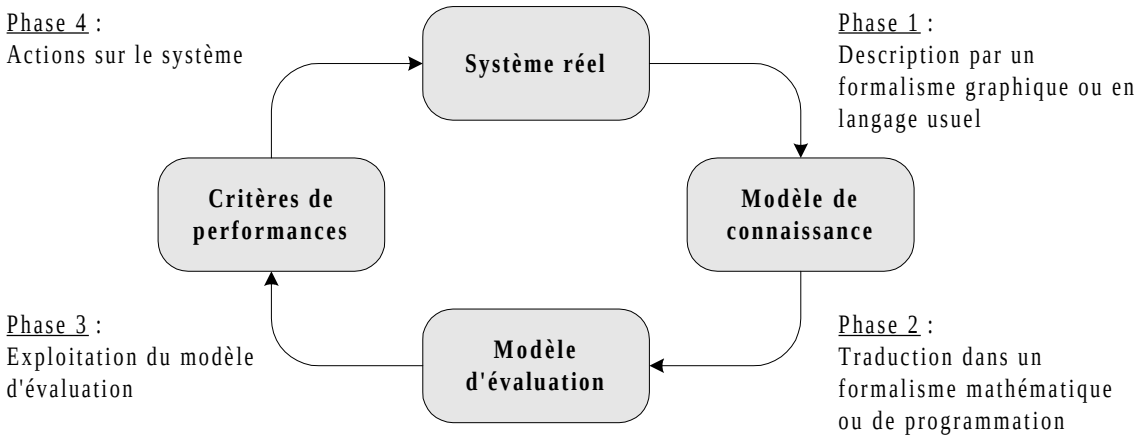

Figure 4.1 : Processus itératif de modélisation

- *Phase 1* : C'est l'analyse du système et la formalisation de la connaissance et des informations recueillies. Cette phase de spécification du système permet de construire le modèle de connaissance. Ceci représente une étape déterminante dans le processus de modélisation.
- *Phase 2* : C'est la traduction du modèle de connaissance en un modèle d'évaluation, dans un formalisme permettant son exploitation pour fournir les critères de performance.
- *Phase 3* : C'est l'exploitation du modèle d'évaluation pour fournir les critères de performance.
- *Phase 4* : C'est l'interprétation des résultats et la détermination des actions à mener sur le système pour atteindre les objectifs visés (amélioration des performances, en ce qui nous concerne).

1.3. Modèle de connaissance

Un système peut être défini comme un ensemble d'éléments en interaction dynamique, organisés en fonction d'un but [De Rosnay 75]. Tout système est caractérisé par sa structure et son fonctionnement. Sa structure est donnée par sa topologie, c'est-à-dire la description des entités qui le composent (sites, modules fonctionnels, etc.) ainsi que leur situation géographique, les interconnexions (réseaux, etc.) et les différents points dans le

système. Son fonctionnement se compose de différentes règles qu'il est indispensable de connaître, telles que des règles de supervision ou des règles de synchronisation.

Dans notre contexte, le modèle de connaissance est une formalisation dans un langage usuel ou graphique de la structure et du fonctionnement du SGBDOO modélisé, ainsi que du système informatique sur lequel il est implanté. Si le SGBD existe, sa structure est connue. Dans le cas contraire, le modèle de connaissance contient l'architecture du futur SGBD ainsi que ses spécifications (caractéristiques et règles de fonctionnement).

Nous subdivisons un système en trois sous-systèmes :

- le *sous-système physique*, qui spécifie les ressources physiques du système ;
- le *sous-système de charge*, qui définit les flux qui parcourent le SGBDOO (principalement des transactions, requêtes et sous-requêtes) ;
- le *sous-système de contrôle*, qui gère les règles de fonctionnement du système.

La démarche d'analyse d'un système en vue de sa modélisation se fait en plusieurs étapes :

- décomposition du système pour en identifier les différents niveaux ;
- décomposition du système en trois sous-systèmes (sous-système de charge, sous-système physique et sous-système de contrôle) ;
- spécification du sous-système physique ;
- spécification du sous-système de charge ;
- spécification du sous-système de contrôle ;
- spécification des communications entre les sous-systèmes (Figure 4.2). C'est le sous-système de contrôle qui administre le système, grâce à des données qu'il reçoit des deux autres sous-systèmes et à des ordres qu'il leur envoie en réaction.

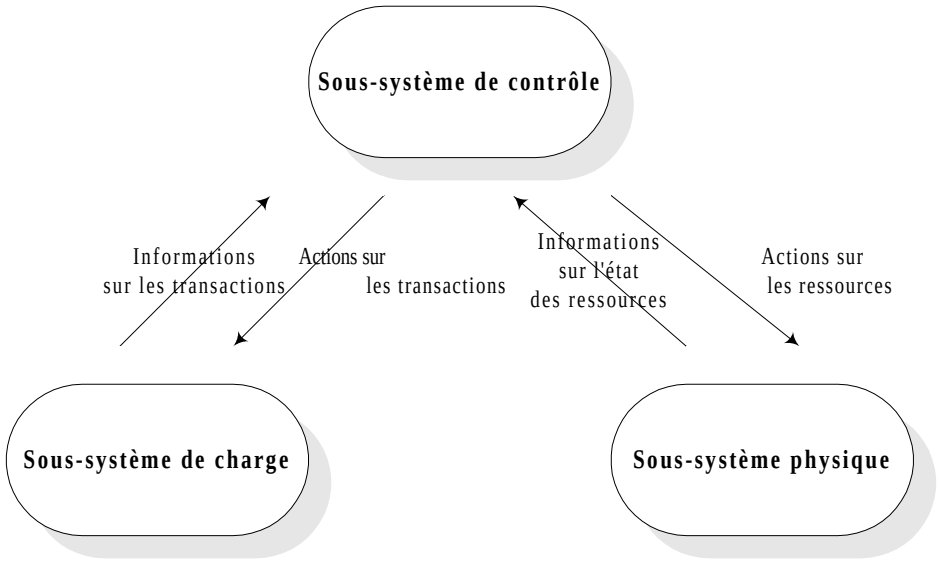

Figure 4.2 : Communications entre les trois sous-sytèmes d'un modèle

Note : Pour affiner l'analyse du système, il est nécessaire de boucler sur les étapes de spécification afin d'avoir toujours le même niveau de détail pour tous les sous-systèmes.

1.4. Modèle d'évaluation

Le modèle d'évaluation est la traduction du modèle de connaissance dans un formalisme mathématique ou dans un langage de programmation. Dans cette étude, les modèles d'évaluation sont des modèles de simulation aléatoire à événements discrets, qui sont bien adaptés à l'étude du comportement transitoire d'un SGBD (au contraire des méthodes analytiques). La simulation à événements discrets peut en effet être définie comme suit [Zeigler 76] [Leroudier 80] [Law et Kelton 91] : dans une simulation à événements discrets, les variables que nous avons besoin de connaître *à chaque instant* sont *discrètes*. Elles sont appelées *variables d'état*. L'ensemble des valeurs possibles de ces variables constitue l'*espace d'état* du système. L'espace d'état est dénombrable ou fini. D'après la définition de l'espace d'état, chaque *changement d'état* ou *événement* se produit de manière discrète dans le temps à des instants $(t_i)_{i \in \mathbf{N}}$. Ces instants (t_i) sont appelés *temps d'occurrence d'événements* ou *dates d'événements*. Les SGBDOO sont de fait des systèmes discrets : leur évolution dans le temps est accomplie de manière discrète.

Comme le processus de validation concerne la structure et le comportement du système, le modèle d'évaluation ne doit donc pas faire apparaître d'incohérence par rapport à la réalité. Le but est d'obtenir des résultats interprétables afin de connaître les performances du système et de comprendre son fonctionnement. Pour y parvenir, il faut mettre en œuvre un certain nombre d'éléments, comme le montre la Figure 4.3. Un environnement de modélisation est nécessaire. Il fournit notamment une méthode de spécification permettant d'analyser le système et d'en déduire un modèle de connaissance, un formalisme mathématique ou de programmation avec lequel le modèle de connaissance est traduit en modèle d'évaluation et des outils d'analyse permettant d'interpréter au mieux les résultats de l'exploitation du modèle d'évaluation.

1.5. Environnement de modélisation

L'objectif d'un environnement de modélisation est de fournir, dans un cadre convivial, des méthodes de spécification et d'analyse, des outils d'interprétation des résultats, ainsi que des outils permettant la mise en œuvre du processus de modélisation. La plupart des langages de simulation proposent aujourd'hui un environnement de modélisation (GPSS [Herscovitch et Schneider 65], SLAM II avec SLAMSYSTEM [Pritsker 86], QNAP2 avec MODLINE [Simulog 92], SIMAN avec ARENA [Pegden et al. 90]…), mais ce n'est en général qu'une surcouche du simulateur lui-même. [Breugnot et al. 90] et [Balci et Nance 92] proposent la composition suivante pour un environnement de modélisation complet (Figure 4.4) :

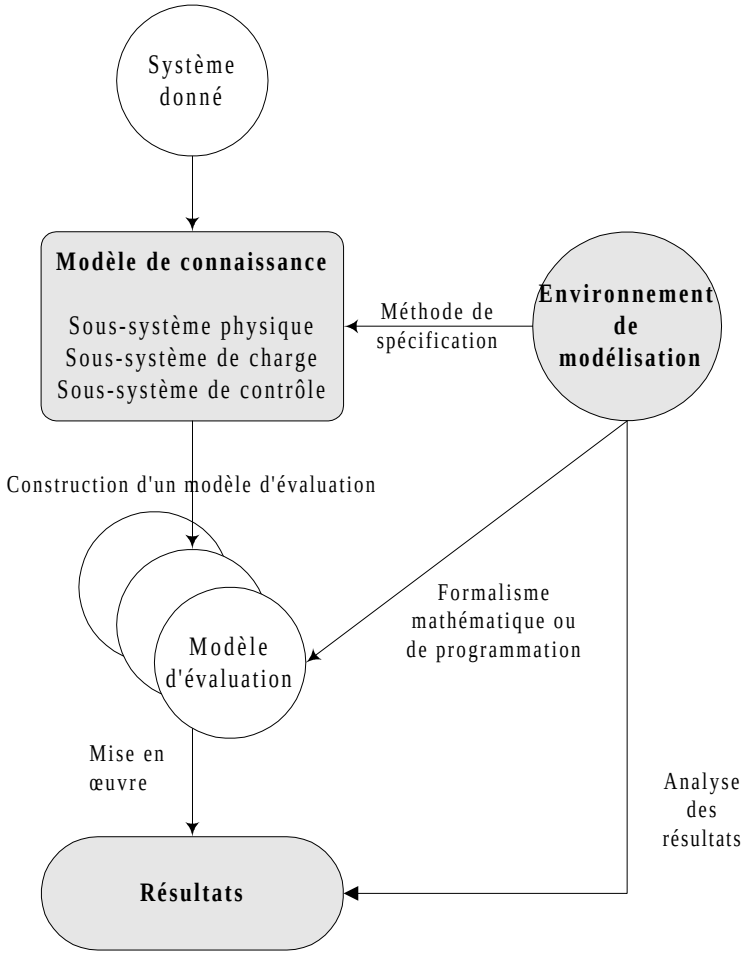

Figure 4.3 : Processus d'obtention du modèle de connaissance, du modèle d'évaluation et de la mise en œuvre

- un *logiciel d'évaluation des performances* valide : noyau de l'environnement, le logiciel d'évaluation des performances doit fournir des résultats fiables. En effet, la simulation aléatoire ne reposant sur aucune théorie mathématique, il faut que d'une part, le modèle de simulation soit correct et que, d'autre part, les résultats de simulation ne soient pas biaisés par un bogue du simulateur ;
- des *outils graphiques* permettant une utilisation conviviale de l'environnement : saisie du modèle de connaissance, spécification du fonctionnement du SGBDOO étudié, exploitation des résultats (animation graphique, par exemple), etc. ;
- des *méthodes d'analyse et de spécification* ainsi que des *outils de spécification* permettant de décrire la structure du système (système d'exploitation et SGBDOO), les flux de données qui le parcourent et les règles de gestion mises en œuvre ;
- des *outils d'aide à la décision et d'analyse des données* destinés à aider à une interprétation correcte des résultats ;

- une *base de composants logiciels* incluant divers morceaux de code standards pouvant être réutilisés dans différentes évaluations de performance ;
- une *méthodologie de modélisation du domaine*.

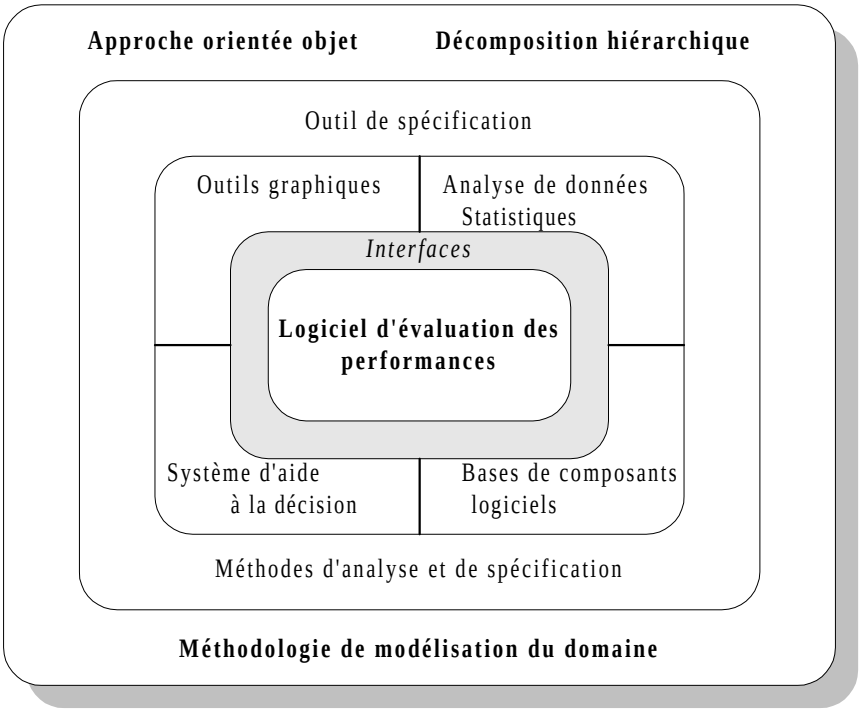

Figure 4.4 : Définition d'un environnement de modélisation

Dans un premier temps, nous nous sommes contentés d'un environnement de modélisation composé d'un minimum d'outils essentiels (Figure 4.5) :

- des outils d'évaluation des performances (QNAP2 [Simulog 95] et notre propre environnement de simulation DESP-C++) ;
- une méthode d'analyse et de spécification exploitant le formalisme UML [Rational Software et al. 97a] [Rational Software et al. 97b] ;
- une base de composants logiciels comprenant divers algorithmes de groupement d'objets et de remplacement de pages en cache mémoire ;
- un logiciel d'animation graphique, VIEWMOD [Hill et al. 93], permettant de visualiser à l'écran en « temps réel » le comportement du système modélisé et de détecter des bogues ou des particularités de fonctionnement ponctuelles indécelables à l'analyse des résultats de simulation globaux.

Nous projetons d'intégrer petit à petit la plupart des éléments d'un environnement de modélisation complet à notre propre environnement. Cependant, ce que nous perdons en complétude avec notre environnement réduit est compensé par une bonne portabilité.

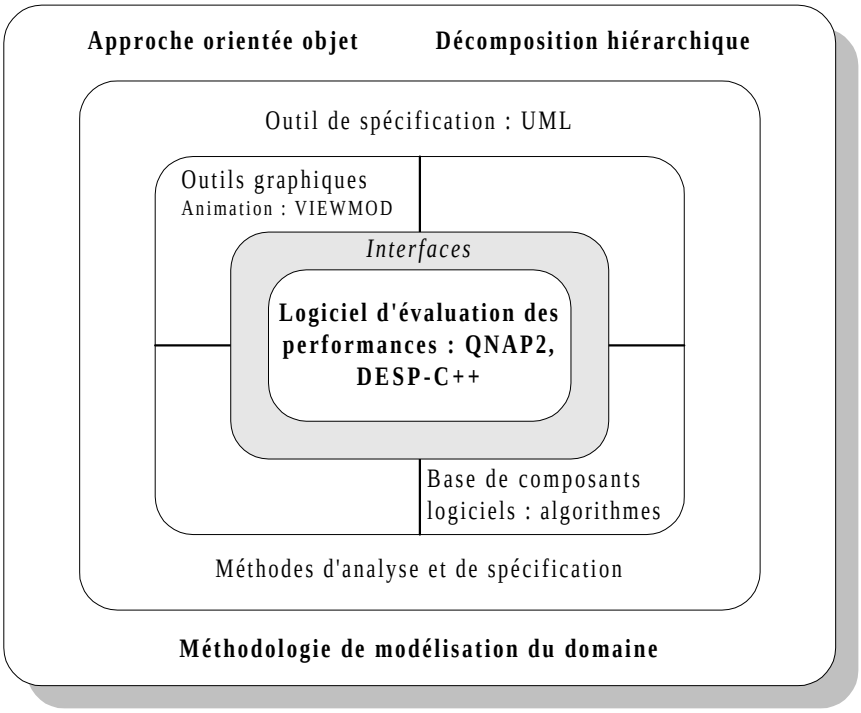

Figure 4.5 : Notre environnement de modélisation

2. Application de la méthodologie pour produire VOODB

La spécification des différents modèles présentés dans cette section a été effectuée en employant les différents formalismes d'UML (*Unified Modeling Language*) [Rational Software et al. 97a] [Rational Software et al. 97b]. Notre choix s'est porté sur UML car nous avions besoin d'un formalisme orienté objet qui assure une certaine cohérence tout au long du processus de modélisation. En outre, UML est maintenant bien connu des experts du domaine des bases de données objet, ce qui permet un dialogue aisé.

2.1. Analyse du domaine

Pour atteindre notre objectif, le domaine de notre étude, c'est-à-dire la classe de systèmes des SGBDOO, a été analysé. L'analyse du domaine identifie les objets, les méthodes et les relations fondamentales qu'il est nécessaire d'inclure dans nos modèles. C'est une sorte de modèle de connaissance pour le domaine des SGBDOO. Notre analyse du domaine est présentée dans la Figure 4.6 sous la forme d'un *diagramme de classes*. Nous y distinguons :

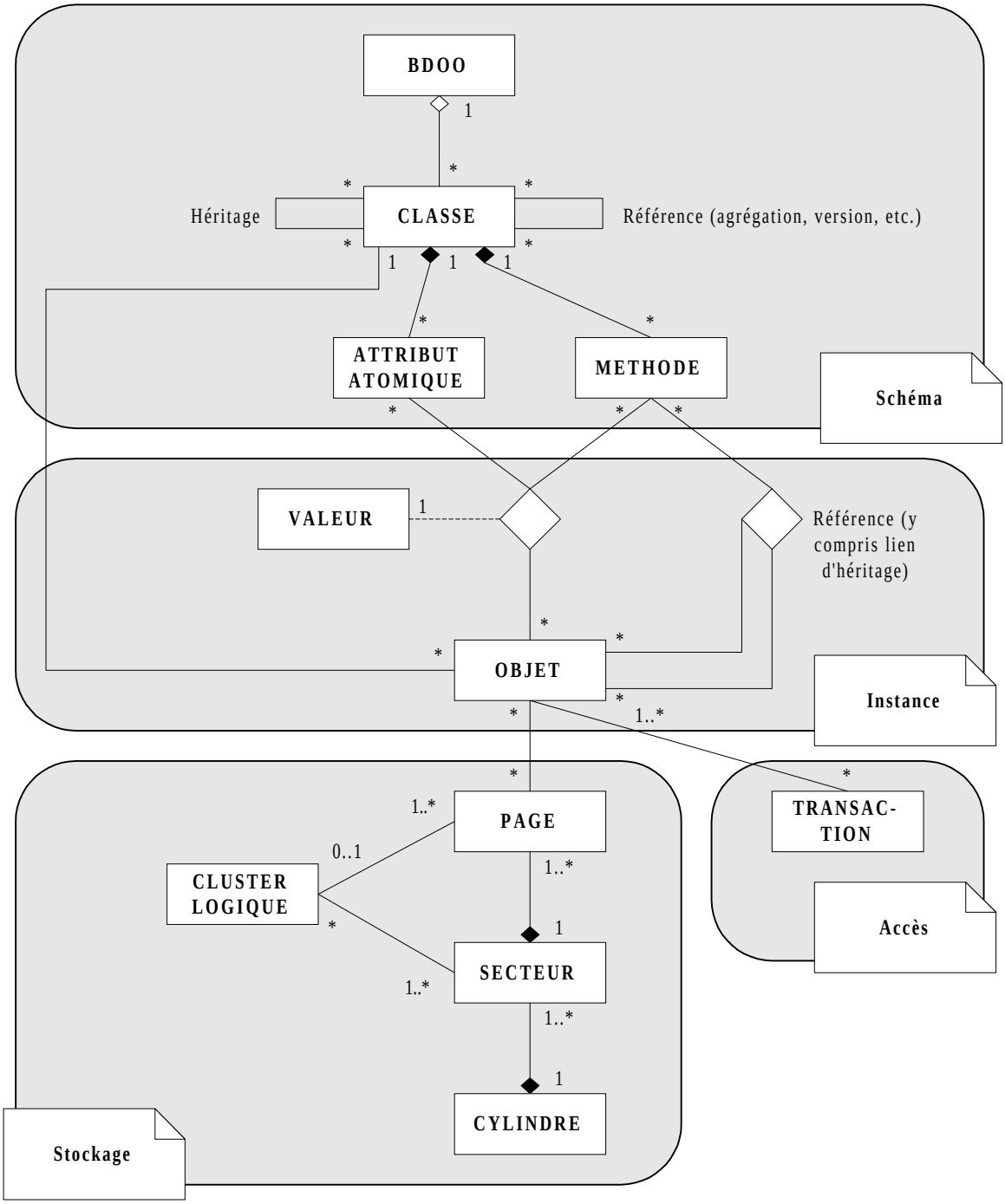

Figure 4.6 : Analyse du domaine

- le schéma d'une base de données orientée objet, constitué d'un ensemble de classes liées entre elles par divers types de références, dont la relation particulière d'héritage (IS-A). Chaque classe est associée à un ensemble d'attributs et de méthodes ;
- l'instanciation de ce schéma en objets, qui possèdent une valeur pour chaque attribut correspondant de leur classe et accèdent à ces attributs grâce aux méthodes de leur classe ;

- le stockage physique des objets de la base de données sur disque, sachant que l'unité de stockage élémentaire est la page, que chaque page appartient à un secteur situé sur un cylindre du disque (cf. Chapitre 2, Section 4.3) et que des pages peuvent constituer des *clusters* logiques ;
- l'accès aux données, c'est-à-dire l'accès aux objets de la base par des transactions.

2.2. Modèle de connaissance

Nous devons dans ce modèle décrire l'exécution des transactions au sein d'un SGBDOO. Un *diagramme d'activité* est adapté dans cette optique (Figure 4.7).

Les transactions sont générées par les *Utilisateurs*, qui les soumettent après un temps de latence prédéfini au *Gestionnaire de Transactions*. Ce dernier détermine les objets qui doivent être chargés en mémoire pour traiter la transaction courante et effectue les opérations nécessaires à ce traitement. Chacun des objets nécessaire à une transaction est requis par le *Gestionnaire de Transactions* auprès du *Gestionnaire d'Objets*, qui trouve sur quelle page disque est stocké l'objet en question. La page est ensuite requise par le *Gestionnaire d'Objets* auprès du *Gestionnaire de Cache Mémoire*, qui vérifie si la page est présente en mémoire vive. Si elle ne s'y trouve pas, une requête de page est transmise par le *Gestionnaire de Cache Mémoire* au *Sous-système des Entrées-Sorties* qui est chargé de gérer les accès disque physiques. Le *Gestionnaire de Cache Mémoire* est également chargé de mettre en œuvre une stratégie de remplacement des pages dans le cache.

Lorsqu'une opération unitaire sur un objet est terminée, le *Gestionnaire de Groupement* peut si nécessaire mettre à jour des statistiques d'utilisation de la base de données. Une analyse de ces statistiques peut déclencher un regroupement des objets, qui est alors également exécuté par le *Gestionnaire de Groupement*. Une telle réorganisation de la base de données peut également être déclenchée de façon externe par les *Utilisateurs*.

Note : Les seuls traitements qui diffèrent lorsque deux algorithmes de groupement différents sont testés sont ceux effectués par le Gestionnaire de Groupement. Les autres traitements restent les mêmes quel que soit le modèle d'évaluation.

2.2.1. Sous-système physique

Les ressources physiques qui composent le sous-système physique sont divisées en deux catégories :

- les *ressources actives* qui effectuent des traitements effectifs ;
- les *ressources passives* qui ne participent pas directement aux traitements, mais qui sont indispensables aux ressources actives pour réaliser leurs opérations.

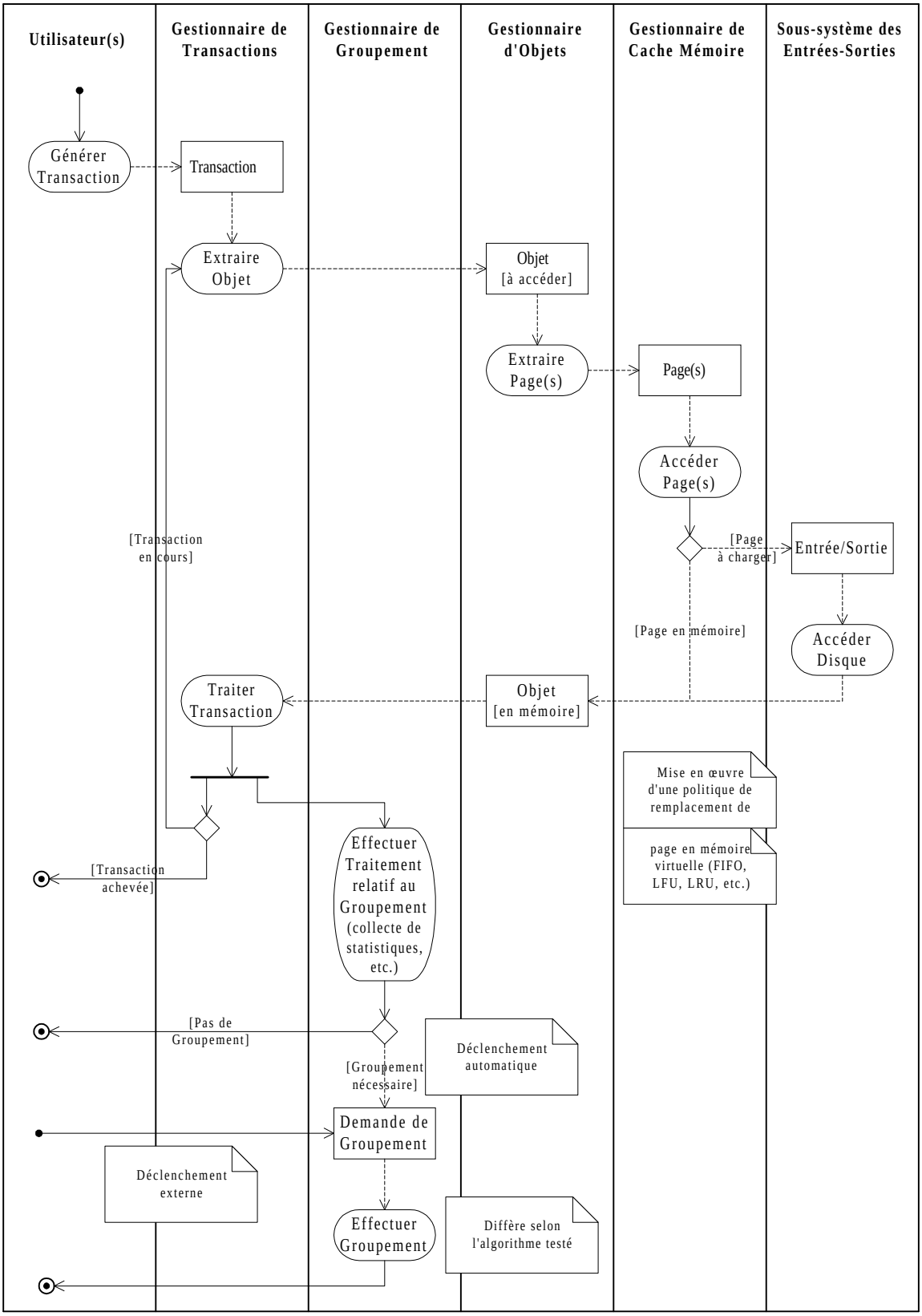

Figure 4.7 : Modèle de connaissance

Les ressources actives de VOODB sont identifiées par les lignes d'eau (*swimlanes*) du modèle de connaissance (Figure 4.7). Elles sont listées dans la Table 4.1. Les ressources passives n'apparaissent pas dans la Figure 4.7. Il est néanmoins primordial d'en établir une liste exhaustive (Table 4.2).

Code	Ressource active
RA1.i	*Utilisateurs :* génération des transactions
RA2	*Gestionnaire de Transactions :* exécution des transactions
RA3	*Gestionnaire de Groupement :* implémentation des algorithmes de groupement d'objets dont les performances sont à évaluer
RA4	*Gestionnaire d'Objets :* accès aux objets
RA5	*Gestionnaire de Cache Mémoire :* gestion du cache en mémoire primaire, mise en œuvre d'une politique de remplacement des pages et d'une politique de préchargement
RA6	*Sous-système des Entrées-Sorties :* accès à une page isolée ou à plusieurs pages contiguës, sur disque

Table 4.1 : Ressources actives de VOODB

Code	Ressource passive
RP1.0	*Processeur et mémoire primaire* dans une architecture centralisée ou *processeur et mémoire primaire du serveur* dans une architecture client-serveur
RP1.i (i>0)	*Processeur et mémoire primaire des clients* dans une architecture client-serveur
PR2	*Contrôleur de disque et mémoire secondaire* du serveur
PR3	*Base de données :* son accès concurrent est géré par un gestionnaire d'ordonnancement, qui applique une politique d'ordonnancement des transactions en fonction du niveau de multiprogrammation. La base de données implémentée dans VOODB est celle du banc d'essais OCB (cf. Chapitre 3, Section 2.1).

Table 4.2 : Ressources passives de VOODB

2.2.2. Sous-système de charge

Le sous-système de charge décrit les flux qui parcourent le système. Ils sont identifiés par les *objets* du modèle de connaissance. Dans la Figure 4.7, leur symbole est un rectangle aux coins droits (▭). Ce sont :

- des flux de transactions (des utilisateurs vers le système),
- des flux de demandes de groupement (issus des utilisateurs ou générés automatiquement),
- des flux d'objets auxquels accéder,
- des flux de pages à lire en mémoire secondaire,
- des flux d'entrées-sorties physiques à effectuer.

Il est nécessaire de décrire plus avant les transactions qui sont exécutées par VOODB. Leur description se situe à deux niveaux : d'une part, elle spécifie quelles sont les différentes classes de transactions, et en second lieu, quelles sont les étapes du traitement de ces transactions.

Nous utilisons dans VOODB les transactions types de la première version du banc d'essais OCB (cf. Chapitre 3, Section 2.2.1), c'est-à-dire les *parcours*, qui seuls sont susceptibles de bénéficier d'un bon regroupement des objets. La hiérarchie formée par ces transactions est rappelée dans la Figure 4.8, sous la forme d'un *diagramme de classes*. Les principales étapes de leur traitement sont énumérées ci-dessous (cf. Chapitre 1, Section 1.4.3) :

- sélection d'un objet auquel accéder ;
- localisation de l'objet sur disque (la page dans laquelle il est stocké) :
  - si son OID est physique, l'accès est immédiat,
  - si son OID est logique, il faut passer par une table d'objets qui associe à chaque OID son adresse en mémoire secondaire ;
- chargement de la page en mémoire primaire (si nécessaire) ;
- autorisation de futures localisations de l'objet en mémoire primaire :
  - grâce à une table des descripteurs associant à chaque OID une adresse en mémoire primaire,
  - en faisant appel à une technique de *swizzling*,
  - en utilisant une adresse uniforme (adresse en mémoire virtuelle) ;
- acquisition des verrous nécessaires (contrôle de concurrence) ;
- appel aux méthodes de l'objet ;
- restitution des verrous.

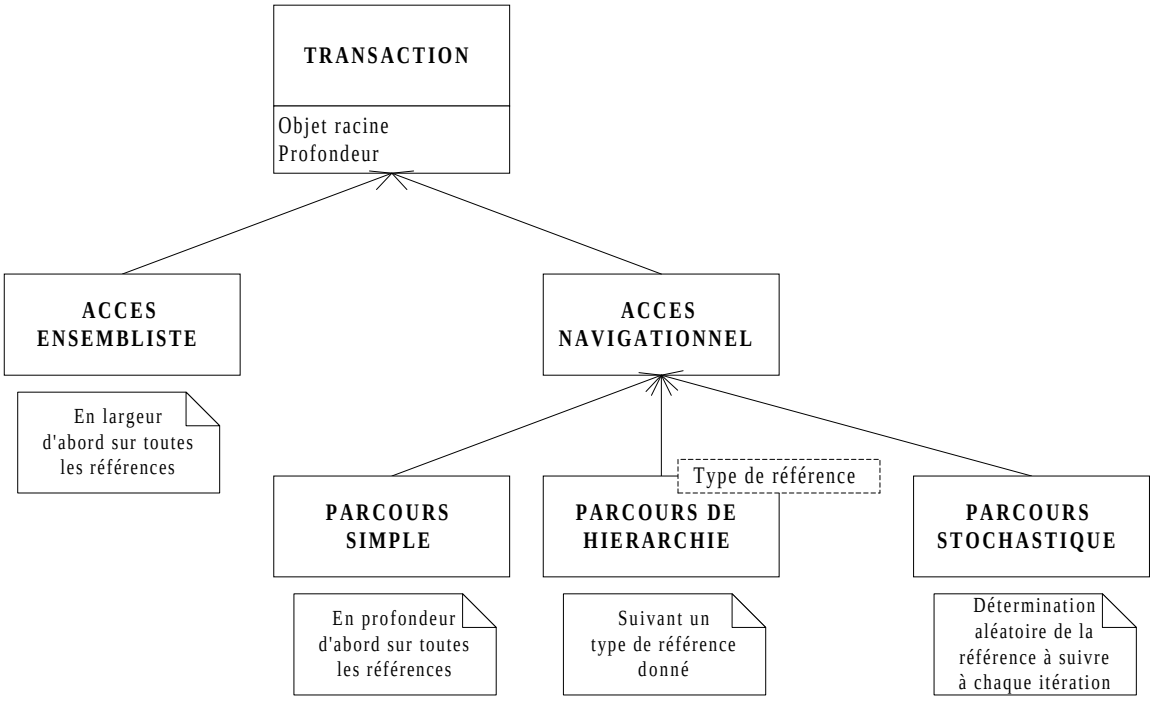

Figure 4.8 : Types de transactions utilisées dans VOODB

2.2.3. Sous-système de contrôle

Le sous-système de contrôle spécifie les règles de décision et de traitement dans VOODB. Chacune des *règles de décision* énoncées ci-dessous (Table 4.3) correspond à une *activité* du modèle de connaissance. Dans la Figure 4.7, leur symbole est un rectangle aux coins arrondis (⬭). Chaque règle de décision est également une méthode d'un objet identifié dans l'analyse du domaine (Figure 4.6) et est déclenchée par une ressource active du sous-système physique (Table 4.1).

Code	Libellé Règle	Méthode de l'objet	Déclenchée par
R1	Générer Transaction	TRANSACTION	RA1.i : Utilisateur n° i
R2	Extraire Objet	TRANSACTION	RA2 : Gestionnaire de Transactions
R3	Extraire Page(s)	OBJET	RA4 : Gestionnaire d'Objets
R4	Accéder Page(s)	PAGE	RA5 : Gestionnaire de Cache Mémoire
R5	Accéder Disque	PAGE	RA6 : Sous-système des Entrées-Sorties
R6	Traiter Transaction	VALEUR	RA2 : Gestionnaire de Transactions
R7	Effectuer Trait. Rel. Groupement	BDOO	RA3 : Gestionnaire de Groupement
R8	Effectuer Groupement	BDOO	RA3 : Gestionnaire de Groupement

Table 4.3 : Liste des règles de décision

2.3. Modèle d'évaluation

2.3.1. Architecture

Notre modèle d'évaluation générique est représenté dans la Figure 4.9 par un *diagramme d'implémentation*. Chacun des composants énumérés ci-dessous correspond à une ligne d'eau (*swimlane*) du modèle de connaissance (Figure 4.7), c'est-à-dire à une ressource active. Les *interfaces* de ces composants correspondent aux flux identifiés dans le sous-système de charge.

- *Utilisateur(s) :* Après un temps de latence prédéfini, un Utilisateur soumet une transaction au Gestionnaire de Transactions. Il peut également déclencher un regroupement des objets de la base de manière externe.
- *Gestionnaire de Transactions :* Le Gestionnaire de Transactions extrait des transactions les objets auxquels accéder et effectue le traitement des transactions.
- *Gestionnaire d'Objets :* Le Gestionnaire d'Objets extrait la page disque à laquelle l'objet appartient, puis en fait la requête au Gestionnaire de Cache Mémoire.
- *Gestionnaire de Cache Mémoire :* Le Gestionnaire de Cache Mémoire vérifie si une page se trouve en mémoire vive et en fait la requête au Sous-système des Entrées-Sorties si ce n'est pas le cas. Il met également en œuvre une stratégie de remplacement de page et une stratégie de préchargement des pages.

- *Gestionnaire de Groupement :* Le Gestionnaire de Groupement est chargé de réorganiser la base de données sur disque afin d'améliorer les performances. Il est également chargé des traitements relatifs au groupement, mais dépendant de l'exécution de transactions, comme la collecte de statistiques. Il peut ainsi provoquer automatiquement un regroupement des objets.
- *Sous-système des Entrée-Sorties :* Ce module gère les accès disque à des pages contiguës ou non.

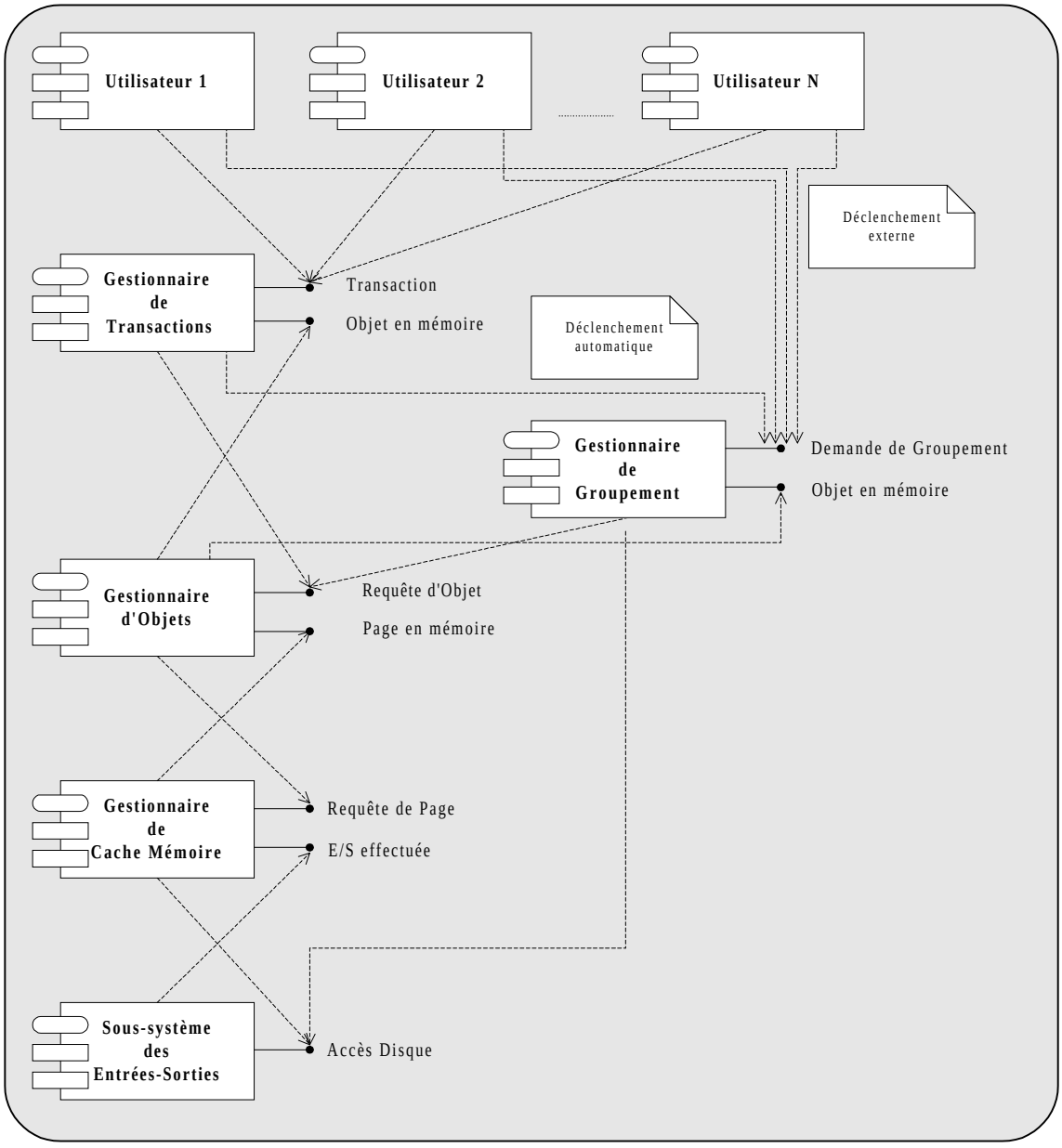

Figure 4.9 : Architecture du modèle d'évaluation

2.3.2. Généricité du modèle

Notre modèle générique permet de simuler le fonctionnement de n'importe quel type de SGBDOO. Il est notamment adapté aux différentes configurations des architectures client-serveur, qui sont dorénavant standards dans les SGBDOO. Il est d'ailleurs particulièrement approprié aux systèmes à *serveur de pages* (comme ObjectStore [Lamb et al. 91], ou O_2 [Deux et al. 91]), mais peut aussi être utilisé pour des systèmes à *serveur d'objets* (comme ORION [Kim et al. 90], ONTOS [Andrews et al. 91], ou VERSANT [Houdas 93]), à *serveur de bases de données*, ou encore à des approches hybrides *multi-serveurs* (comme GemStone [Servio Corporation 92]). L'organisation des composants de VOODB selon le type d'architecture client-serveur modélisé est donné à titre d'exemple dans les Tables 4.4, 4.5 et 4.6.

Client	Serveur
Utilisateur(s) Gestionnaire de Transactions Gestionnaire de Regroupement	Gestionnaire d'Objets Gestionnaire de Cache Mémoire Sous-système des Entrées/Sorties

Table 4.4 : VOODB en serveur d'objets

Client	Serveur
Utilisateur(s) Gestionnaire de Transactions Gestionnaire de Regroupement Gestionnaire d'Objets	Gestionnaire de Cache Mémoire Sous-système des Entrées/Sorties

Table 4.5 : VOODB en serveur de pages

Client	Serveur
Utilisateur(s)	Gestionnaire de Transactions Gestionnaire de Regroupement Gestionnaire d'Objets Gestionnaire de Cache Mémoire Sous-système des Entrées/Sorties

Table 4.6 : VOODB en serveur de bases de données

2.3.3. Paramètres de simulation

Les paramètres utilisés dans nos simulations sont divisés en quatre catégories (Figure 4.10), selon qu'ils définissent :

- le système (Table 4.7),
- les composants qui influencent les performances (Table 4.8),
- la base de données (Table 4.9),

- les transactions exécutées sur cette base (Table 4.10).

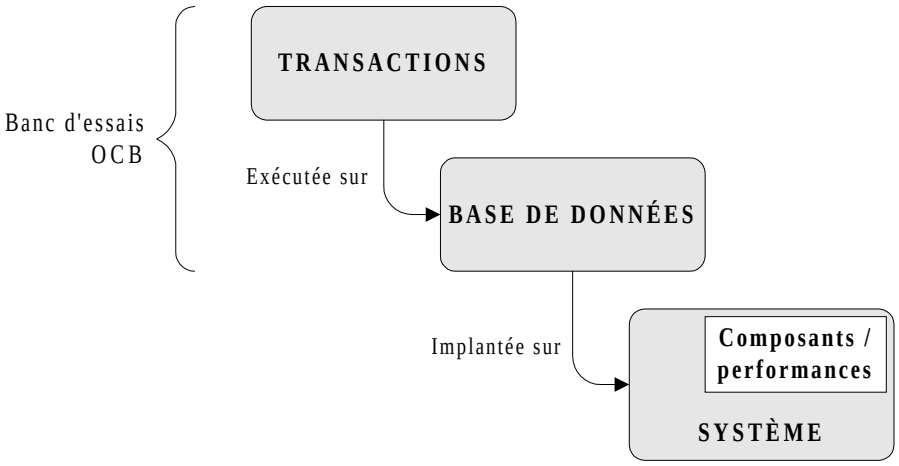

Figure 4.10 : Catégories de paramètres de VOODB

Paramètre	Code	Domaine de valeur	Valeur par défaut
Classe de système	SYSCLASS	{Centralisé | Serveur d'Objets | Serveur de Pages | Serveur de BD | Autre*}	Serveur de Pages
Niveau de multiprogrammation	MULTILVL		10
Nombre d'utilisateurs	NUSERS		1
Taille d'une page disque	PGSIZE	{512 | 1024 | 2048 | 4096 } octets	4096 octets
Taille du cache de page	BUFFSIZE		100 pages
Temps d'acquisition des verrous	GETLOCK		0,5 ms
Temps de restitution des verrous	RELLOCK		0,5 ms
Temps de recherche sur disque	DISKSEA		7,4 ms
Temps de latence disque	DISKLAT		4,3 ms
Temps de transfert disque	DISKTRA		0,5 ms
Débit réseau	NETTHRU		1 Mo/s

Table 4.7 : Paramètres définissant le système

Note : De précédentes études [Darmont et Gruenwald 96b] [Darmont et al. 97] ont mis en évidence le fait que l'impact des temps opératoires en mémoire primaire est négligeable par rapport à ceux nécessaires aux entrées-sorties. Les paramètres concernant les opérations en mémoire (chargement d'un mot mémoire, tests, accès à des tables de hachage, *swizzling*, etc.) ont donc été omis. En effet, un processeur bénéficiant d'une puissance de calcul de seulement 1 Mips procède à une opération en mémoire en 10^{-3} ms, alors que le temps nécessaire à un accès disque est de l'ordre de plusieurs millisecondes.

* Valeur définie par l'utilisateur.

Paramètre	Code	Domaine de valeur	Valeur par défaut
Politique de groupement d'objets	CLUSTP	{Aucune | Autre*}	Aucune
Placement initial des objets	INITPL	{Séquentiel | Séquentiel optimisé | Autre*}	Séquentiel optimisé
Politique de remplacement de pages dans le cache	PGREP	{RANDOM | FIFO | LFU | LRU-K | CLOCK | GCLOCK | Autre*}	LRU (LRU-1)
Politique de préchargement des objets	PREFETCH	{Aucune | Autre*}	Aucune

Table 4.8 : Paramètres définissant les politiques de regroupement et de gestion de cache

Paramètre	Code	Domaine de valeur	Valeur par défaut
Nombre de classes	NC		50
Nombre maximum de références, par classe	MAXNREF(i)		10
Taille de base des instances de chaque classe	BASESIZE(i)		50 octets
Nombre total d'objets	NO		20000
Nombre de types de référence	NREFT		3
Distribution aléatoire des types de référence	DIST1	{Uniforme | Normale | Autre*}	Uniforme
Distribution aléatoire des références aux classes	DIST2	{Uniforme | Normale | Autre*}	Uniforme
Distribution aléatoire des objets au sein des classes	DIST3	{Uniforme | Normale | Autre*}	Uniforme
Distribution aléatoire des références d'objets	DIST4	{Uniforme | Normale | Autre*}	Uniforme
Localité de référence au niveau des classes	CLOCREF	1 – NC	NC
Localité de référence au niveau des objets	OLOCREF	1 – NO	NO

Table 4.9 : Paramètres définissant la base d'objets d'OCB

2.3.4. Implémentation

2.3.4.1. Choix de simulateur

En premier lieu, nous avons choisi pour implémenter notre modèle d'évaluation le logiciel QNAP2 v9.2 (*Queuing Network Analysis Package* 2nd generation) [Simulog 95], qui regroupe les fonctionnalités essentielles suivantes :

- c'est un outil de simulation validé et fiable ;
- il autorise l'emploi d'une approche orientée objet depuis sa version 6.0 ;
- il comprend un langage algorithmique complet, dérivé de Pascal, permettant l'implémentation relativement aisée d'algorithmes complexes (groupement d'objets, politiques de remplacement de pages dans le cache, politiques de préchargement des objets, etc.).

* Valeur définie par l'utilisateur.

Paramètre	Code	Domaine de valeur	Valeur par défaut
Temps de latence moyen entre deux transactions	THINK		4 s
Probabilité d'occurrence des accès ensemblistes	PSET	0 – 1	0,25
Probabilité d'occurrence des parcours simples	PSIMPLE	0 – 1	0,25
Probabilité d'occurrence des parcours hiérarchiques	PHIER	0 – 1	0,25
Probabilité d'occurrence des parcours stochastiques	PSTOCH	0 – 1	0,25
Probabilité d'une demande de regroupement externe	PCLUSTD	0 – 1	0
Profondeur des accès ensemblistes	SETDEPTH		3
Profondeur des parcours simples	SIMDEPTH		3
Profondeur des parcours hiérarchiques	HIEDEPTH		5
Profondeur des parcours stochastiques	STODEPTH		50
Distribution aléatoire des objets racines des transactions	DIST5	{Uniforme | Normale | Autre*}	Uniforme

Table 4.10 : Paramètres définissant les transactions orientées groupement d'OCB

Cependant, la langage de simulation supporté par QNAP2 est interprété. Les modèles écrits en QNAP2 sont donc beaucoup plus lents à l'exécution que s'ils étaient codés dans un langage compilé. Nos propres modèles de simulation étaient d'autant plus lents qu'ils mettaient en œuvre de nombreuses opérations de gestion de la base de données OCB, en plus des opérations normales de simulation prises en charge par QNAP2. Nous ne pouvions donc pas envisager la campagne de simulation intensive initialement prévue. Par exemple, les expériences de simulation les plus simples (sans prise en compte du regroupement d'objets) prenaient huit heures, et les plus complexes plus d'une semaine. Nous ne pouvions donc pas obtenir de résultats probants au-delà des plus basiques.

Nous avons finalement envisagé l'utilisation du langage C++ [Stroustrup 97], qui est à la fois un langage compilé et orienté objet. Cela nous a permis de réutiliser une grande partie du code C++ de la version d'OCB utilisée pour évaluer les performances des systèmes Texas [Singhal et al. 92] et O_2 [Deux et al. 91]. Nous avons cependant dû coder un petit simulateur pour prendre en charge la partie gestion de la simulation, auparavant complètement administrée par QNAP2. Cet outil, DESP-C++, est présenté en détail au Chapitre 5. Il a été validé autant que possible. Son utilisation nous a permis des gains de temps importants, les expériences de simulation se déroulant de 20 à 1000 fois plus vite avec DESP-C++ qu'avec QNAP2, selon la complexité du modèle de simulation (plus un modèle est compliqué, plus les performances de QNAP2 en temps d'exécution se dégradent).

* Valeur définie par l'utilisateur.

2.3.4.2. Traduction du modèle de connaissance

Une fois le modèle de connaissance défini, sa traduction en un modèle d'évaluation directement exploitable est quasi-automatique, que l'environnement cible soit un logiciel de simulation général ou un langage de programmation. Chaque entité du modèle de connaissance apparaît d'une façon ou d'une autre dans le modèle d'évaluation. Par exemple, dans un environnement orienté objet, les ressources actives et les ressources passives deviennent des classes instanciées et les règles de fonctionnement se traduisent en méthodes.

Pour illustrer la façon dont le modèle de connaissance est traduit en programme de simulation, nous présentons sur un exemple la correspondance entre chaque élément du modèle de connaissance et leur code dans le modèle d'évaluation. Pour cela, nous nous plaçons au niveau le plus « bas » dans VOODB : le Sous-système des Entrées-Sorties, qui administre la lecture et l'écriture sur le disque d'ensembles de pages (contiguës ou non). Nous présentons le code QNAP2 et DESP-C++ de chaque sous-système :

- *sous-système physique :* ressource active RA6 (Sous-système des Entrées-Sorties) et ressources passives RP1 (processeur) et RP2 (disque) ;
- *sous-système de charge :* demandes d'entrées-sorties ;
- *sous-système de contrôle :* règle de fonctionnement R5 (Accéder disque).

Dans les figures qui suivent, les zones grisées et numérotées représentent les lignes de code qui se correspondent en QNAP2 (colonne de gauche) et DESP-C++ (colonne de droite).

Sous-système physique

La Figure 4.11 présente la définition de la classe « ressource passive » et de son constructeur. La partie correspondant à DESP-C++ est plus complexe, car elle intègre explicitement des éléments concernant la simulation, alors qu'ils sont pris en compte implicitement par QNAP2. Les ressources passives RP1 (processeur) et RP2 (disque) sont des instances de cette classe « ressource passive ». Sur la Figure 4.11, la zone n° 1 correspond à la définition de la file d'attente des clients à servir par la ressource, ainsi qu'à la définition de la capacité de cette ressource (c'est-à-dire le nombre de clients pouvant être servis simultanément). La zone n° 2 correspond, dans le constructeur de la classe, à l'affectation du nom et de la capacité de la ressource passive.

La Figure 4.12 présente la définition de la ressource active RA6 (le Sous-système des Entrées-Sorties) et de son comportement. La zone n° 1 est la déclaration d'un indicateur de le dernière page accédée sur le disque (pour gérer les accès contigus) et de pointeurs vers les ressources passives RP1 et RP2 (processeur et disque). La zone n° 2 est la déclaration de la méthode contenant la règle R5.

```
& Sous-système physique
& Définition des ressources passives

& Classe ressource passive

/DECLARE/ OBJECT rp;
            QUEUE q;      & File d'attente
            INTEGER mult; & Capacité rs
          END;

& Station - ressource passive

/STATION/ NAME=*rp.q;
          TYPE=RESOURCE,MULTIPLE(mult);

& Constructeur objet ressource passive

/DECLARE/ PROCEDURE newrp(nomrp,valmult);

          VAR REF rp nomrp; & Nom ressource
          INTEGER valmult;  & Valeur MULTIPLE
                            & (capacité)
          BEGIN
            nomrp:=NEW(rp);
            nomrp.mult:=valmult;
          END;
```

①

②

```
// Sous-système physique
// Définition des ressources (passives)
// Les ressources actives sont des
// sous-classe

// Définition classe Resource

class Resource {
  public:
    // Méthodes publiques
    Resource(char n[STRS], int cap,
Simulation *sim); // Constructeur
    ~Resource(); // Destructeur
    void PurgeQueue(); // Vide file d'attente
    void P(int event, Client *client, int
prior); // Réservation ressource
    void V(); // Libération ressource
    Simulation *Sim(); // Ret. l'obj. simul.
    void ResetCounters(); // Init. compteurs
    void ResetStats(); // Init stats globales
    void Stats(); // Calcul des stats
    void DisplayStats(); // Affichage stats
    float Mean(short i); // Ret. moyenne i
    float Dev(short i); // Ret. ecart-type i
  private:
    // Méthodes internes
    void EnQueue(int eventcode, Client
*client, int priority); // Insertion
    int GetEventCode(); // 1er evnt file
    Client *GetClient(); // 1er cli file
    void DestroyTop(); // Détr. 1er élt file
    int QueueEmpty(); // Etat file d'attente
    // Attributs
    char name[STRS]; // Nom de la ressource
    QueueCell *top; // Tête de file d'attente
    QueueCell *bottom; // Fin file d'attente
    int capacity; // Capacité ressource
    int ccapacity; // Capacité courante
    Simulation *simul; // Objet simulation
    float response; // Tps réponse sur 1 rep
    float wait; // Tps attente sur 1 rep
    int nbserv; // Nombre de clients servis
    float stats[5],stats2[5]; // Stats cumul
    int n; // Stats (nombre d'expériences)
    float mean[5], dev[5]; // Moyennes - ec-t
    }; // 0 : Temps de reponse
       // 1 : Temps d'attente
       // 2 : Nombre de clients servis
       // 3 : Nombre de clients en cours
       // 4 : Nombre de clients en attente

// CLASSE Resource : Constructeur

Resource::Resource(char n[STRS], int cap,
Simulation *sim) {

  strcpy(name,n);
  capacity=cap;
  simul=sim;
  top=NULL;
  bottom=NULL; }
```

Figure 4.11 : Code de simulation des ressources passives (QNAP2/DESP-C++)

La zone n° 3 contient l'initialisation, dans le constructeur du Sous-système des Entrées-Sorties, de l'indicateur de dernière page et des pointeurs vers les ressources passives. La zone n° 4 correspond à la réservation de la ressource passive RP1 (le processeur).

```
& Sous-système physique : Ressource active RA6
```

```
& Définition du Sous-système des E/S
/DECLARE/ OBJECT ar6;
  QUEUE q;          & File d'attente
  INTEGER lastpage; & Dernière page accédée
  REF QUEUE rsproc; & Processeur (RP1)
  REF QUEUE rsdisk; & Disque (RP2)
END;
```

① ②

```
& Déclaration de la méthode du s.s. de charge
/DECLARE/ PROCEDURE accdisk(pgacc,lstpg); & R5
  INTEGER pgacc; & Nombre de pages à accéder
  INTEGER lstpg; & Dernière page accédée
FORWARD;

& Constructeur du Sous-système des E/S
/DECLARE/ PROCEDURE newar6(ptr,rsp,rsd);
  VAR REF ar6 ptr; & Pointeur de retour
  REF QUEUE rsp;   & Pointeur vers processeur
  REF QUEUE rsd;   & Pointeur vers disque
  BEGIN
    ptr:=NEW(ar6);
    WITH ptr DO
      BEGIN
        lastpage:=-999;
        rsproc:=rsp;
        rsdisk:=rsd;
      END;
  END;
```

③

```
& Comportement du Sous-système des E/S
/STATION/ NAME=*ar6.q;
  TYPE=INFINITE;
  SERVICE=
    BEGIN
      P(rsproc); & Réservation processeur
```

④

```
      P(rsdisk); & Réservation disque
      CUSTOMER::intreq.ptrcell:=
        CUSTOMER::ioreq.pgnums.tete;
      WHILE CUSTOMER::intreq.ptrcell<>NIL DO
        BEGIN
          & Appel à la règle R5
          accdisk(CUSTOMER::intreq.ptrcell.
            intval,lastpage);
          & Mise à jour dernière page accédée
          laspage:=
            CUSTOMER::intreq.ptrcell.intval;
          & Page suivante
          CUSTOMER::intreq.ptrcell:=
            CUSTOMER::intreq.ptrcell.next;
        END;
      V(rsdisk); & Libération disque
      V(rsproc); & Libération processeur
      SET(CUSTOMER::intreq.ok);
      & Libération mémoire
      purge(CUSTOMER::ioreq.pgnums);
    END;
  TRANSIT=OUT;
```

⑤ ⑥ ⑦

```
// Sous-système physique: Ressource active RA6
// Définition du Sous-système des E/S
class IOSubSys: public Resource {
  public:
    // Déclaration du constructeur
    IOSubSys(char n[STRS], int cap, Simulation
*sim, Resource *rsp, Resource *rsd);
    // Déclaration des méthodes des événements
    void Event60(Client *client);
    void Event61(Client *client);
    void Event62(Client *client);
    void Event63(Client *client);
private:
    // Dernière page accédée
    int lastpage;
    // Processeur (RP1) et Disque (RP2)
    Resource *rsproc, *rsdisk;

    // Déclaration méthode du s.s. de charge
    void AccDisk(Client *client); }; // R5
```

```
// Constructeur du Sous-système des E/S
IOSubSys::IOSubSys(char n[STRS], int cap,
Simulation *sim, Resource *rsp, Resource
*rsd): Resource(n, cap, sim) {
  rsproc=rsp;
  rsdisk=rsd;
  lastpage=-999; }

// Comportement du Sous-système des E/S
// Evénement #0 – Réservation RA6
void IOSubSys::Event60(Client *client) {
  this->P(61,client,1); }
// Evénement #1 – Réservation processeur
void IOSubSys::Event61(Client *client) {
  rsproc->P(62,client,1); }

// Evénement #2 – Réservation disque
void IOSubSys::Event62(Client *client) {
  client->pageindex=1; // 1ère page à accéder
  rsdisk->P(63,client,1); }

// Evénement #3 – Règle R5 : Accès disque
void IOSubSys::Event63(Client *client) {
  if (client->pageindex<=
  client->pgnums->NbElem()) {
    AccDisk(client); // Appel règle R5
    // Statistiques
    NBIO++;
    if (client->type<=4) TNBIO++;
  } else {

 // Fin

    rsdisk->V(); // Libération disque
    rsproc->V(); // Libération processeur
    this->V();   // Libération RA6
    if (client->type!=7) // Lecture
      Sim()->Sched()->Schedule(53,
      Sim()->Tnow(),client); // Page en mém.
    else // Ecriture, fin de traitement
      Sim()->KillClient(client); } }
```

Figure 4.12 : Code de simulation du Sous-système des Entrées-Sorties (QNAP2/DESP-C++)

La zone n° 5 correspond à la réservation de la ressource passive RP2 (le disque). La zone n° 6 est l'appel effectif à la méthode contenant la règle R5. Finalement, la zone n° 7 correspond à la libération des deux ressources passives.

Sous-système de charge

La Figure 4.13 présente la définition du sous-système de charge dans sa globalité, c'est-à-dire la spécification de toutes les entités (les clients) qui parcourent le système. Les demandes d'entrées-sorties sont incluses dans ces définitions. La zone n° 1 de la Figure 4.13 correspond à la déclaration de l'identificateur (unique) du client. La zone n° 2 est la définition des différents types de clients. Dans le cas de QNAP2, seuls les types de clients qui apparaissent dans les résultats statistiques sont nécessaires, les autres étant gérés en interne. La zone n° 3 décrit les éléments nécessaires à l'exécution des transactions, c'est-à-dire l'objet racine de chaque transaction, la profondeur courante dans le parcours de graphe, ainsi que deux listes utilisées par les méthodes d'accès aux objets : liste des objets racines (les nœuds dans les graphes) et liste des objets à accéder au niveau suivant d'un graphe. La zone n° 4 précise, pour les parcours hiérarchiques, le type de référence à suivre. Finalement, la zone n° 5 indique, pour les demandes d'entrées-sorties, une liste de pages disque à accéder.

Sous-système de contrôle

La Figure 4.14 présente le code de la méthode correspondant à la règle R5 (Accéder disque), qui gère les accès disques individuels. Le traitement consiste en fait à déterminer si la page courante est contiguë à la page précédente. Dans VOODB, les pages contiguës ont des numéros adjacents, ce qui rend cette détection facile. S'il y a contiguïté, le temps nécessaire à une entrée-sortie est égal à la somme du temps de latence et du temps de transfert propre au disque utilisé. Dans le cas contraire, il est nécessaire d'ajouter un temps de recherche, lui aussi propre au matériel employé (cf. Chapitre 2, Section 4.3).

Synthèse

La Table 4.11 récapitule la façon dont les entités du modèle de connaissance sont traduites en concepts QNAP2 et DESP-C++. Comme ces deux outils de simulation utilisent une *approche station* (où le comportement de chaque *ressource active* est décrit), la Table 4.11 fournit aussi une traduction pour le langage SLAM II [Pritsker 86], qui emploie une *approche transaction* (où les spécifications concernent les opérations subies par les *entités* qui traversent le système). Ceci a pour but de souligner qu'il est également facile d'implémenter VOODB à l'aide d'un simulateur utilisant une approche transaction.

```
& Sous-système de charge                    // Sous-système de charge : CLASSE Client
& Définition des clients QNAP2

                                            class Client {
/DECLARE/ & Identificateur client           public :
          CUSTOMER INTEGER cliid;     ①       int cliid; // Identificateur client
          & Pointeur sur une requete
          & (nouveau client)
          CUSTOMER REF CUSTOMER req;
          & Pointeur vers un FLAG
          & (synchronisation)
          CUSTOMER REF FLAG refsync;
          & Variables utilitaires
          & (indices de boucle, etc.)
          CUSTOMER INTEGER i,j,k;           short type; // Type de client
          // Types de client          ②                 // 0 = Transaction
          CLASS set;                                    // 1 = Acces ensembliste
          CLASS simp;                                   // 2 = Parcours simple
          CLASS hie;                                    // 3 = Parcours hierarchique
          CLASS sto;                                    // 4 = Parcours stochastique
                                                        // 5 = Demande de groupement
                                                        // 7 = Ecriture directe
                                                        // 9 = Client fictif
          & Transaction             ③       // Transaction
          & Objet racine                    // Objet racine
          CUSTOMER INTEGER rootoid;         int rootoid;
          & Profondeur courante             // Profondeur courante
          CUSTOMER INTEGER curdepth;        int curdepth;
          & Liste des "racines"             // Liste des "racines"
          CUSTOMER REF chlist roots;        ChList *roots;
          & Liste des objets à accéder      // Liste des objets à accéder
          CUSTOMER REF chlist toaccess;     ChList *toaccess;
                                            // Numéro d'itération dans la
                                            // transaction
                                            int trans_it;
                                            // Objet précédemment accédé
                                      ④     int prevoid;
          & Parcours hiérarchique           // Parcours hiérarchique
          & Type de référence               // Type de référence
          CUSTOMER INTEGER reftype;         short reftype;
          & Requête interne
          & Flag (synchronisation)
          CUSTOMER FLAG ok;
          & Pointeur cellule liste ch.
          CUSTOMER REF cell ptrcell;
          & Requête d'objet
          & OID
          INTEGER reqoid;               ⑤
          & Requête de pages                // Requête de page ou d'E/S
          & Numéros de page (cache)         // Numéros de page
          CUSTOMER REF chlist pgnums;       ChList *pgnums;
          & Demande d'entrée/sortie         // Index de page
          & Numéros de page (disque)        int pageindex;
          CUSTOMER REF chlist pgnums;
                                            // Attributs de base
                                            Client *next;
                                            Client *previous;

                                        // Ajouter ici les attributs supplémentaires
                                        };
```

Figure 4.13 : Code de simulation du Sous-système de charge (QNAP2/DESP-C++)

3. Bilan

Nous avons identifié au Chapitre 2 (Section 4.4.2) les avantages et les inconvénients de l'approche simulation telle qu'elle a été employée jusqu'ici, par rapport aux critères de pertinence, de fiabilité, de faisabilité et d'indépendance.

```
& Sous-système de contrôle
& R5 : Accéder Disque

/DECLARE/ PROCEDURE accdisk;

& Paramètres : INTEGER pgacc
&              Numéro de la page à accéder
&              INTEGER derpg
&              Dernière page accédée

  BEGIN
    IF (pgacc<(derpg-1)) OR (pgacc>(derpg+1))
      THEN CST(disksea); & Accès non contigu
    & Accès contigu : pas de temps de rch
    CST(disklat+disktra);
  END;
```

```
// Sous-système de contrôle
// CLASSE IOSubSys : Accès disque (R5)

void IOSubSys::AccDisk(Client *client) {

  int pagetoaccess;

  pagetoaccess=client->pgnums->
    Scan(client->pageindex++);
  if ((pagetoaccess<(lastpage-1)) ||
 (pagetoaccess>(lastpage+1))) {
    lastpage=pagetoaccess;
    // Accès non contigu
    Sim()->Sched()->Schedule(63,Sim()->
      Tnow()+DISKSEA+DISKLAT+DISKTRA,client);
  } else {
    lastpage=pagetoaccess;
    // Accès contigu : pas de temps de rech.
    Sim()->Sched()->Schedule(63,Sim()->
      Tnow()+DISKLAT+DISKTRA,client);
  } }
```

Figure 4.14 : Code de simulation de la règle R5 : Accéder disque (QNAP2/DESP-C++)

Sous-système	Entité	Traduction QNAP2	Traduction DESP-C++	Traduction SLAM II
Charge	(Sous-)Transaction	Objet CUSTOMER	Instance de la classe *Client*	Entité SLAM
Physique	Ressource passive	Objet RESOURCE STATION	Instance de la classe *Resource*	Bloc RESOURCE
	Ressource active	Objet STATION standard	Instance d'une classe ressource active héritant de la classe *Resource*	Ensemble de nœuds SLAM (ACTIVITY, EVENT, FREE, GOON…)
Contrôle	Règle de fonctionnement	PROCEDURE appelée dans la clause SERVICE d'une ressource active	Méthode d'une classe ressource active	Routine FORTRAN appelée dans un nœud EVENT

Table 4.11 : Traduction des entités du modèle de connaissance

Nous avons établi que les critères de faisabilité et d'indépendance étaient satisfaits, mais que la pertinence et la fiabilité des modèles existants pouvait être améliorée. Pour satisfaire ces deux critères dans notre propre modèle de simulation, nous avons mis en œuvre les éléments suivants.

- *Pertinence :* Bien que VOODB soit destiné à évaluer les performances d'algorithmes de regroupement d'objets, nous avons adopté une démarche générique, qui nous permet de poursuivre des buts plus variés. Par exemple, il est possible de désactiver les fonctionnalités de VOODB gérant le groupement d'objets, afin d'examiner le comportement ou les performances du SGBDOO modélisé de façon plus globale. Il est également possible d'étendre VOODB afin de prendre en compte d'autres composants influençant les performances, comme l'optimisation des requêtes ou le contrôle de concurrence. Or, ceci n'est pas possible avec les modèles dédiés au groupement d'objets existants. Par ailleurs, partant de la constatation que la définition d'une base d'objets et de transactions associées est indispensable lors de la conception d'un modèle de SGBDOO, nous

avons réutilisé les techniques en usage dans le domaine des bancs d'essais en intégrant OCB au sein de VOODB, ce qui permet d'assurer une certaine cohérence entre la base d'objets et les transactions qui l'utilisent.

- *Fiabilité :* Afin de produire des modèles de simulation fiables et de systématiser leur spécification, nous proposons une méthodologie de modélisation des SGBDOO. Cette méthodologie est un guide qui permet, étape par étape, de concevoir très précisément des modèles de connaissance et d'assurer leur traduction quasi-automatique en modèles d'évaluation directement exploitables.

Par ailleurs, nous avons le souci de valider notre modèle de simulation VOODB. En effet, pour être utile pour des évaluations de performance *a priori* (concernant des systèmes ou des algorithmes non encore implémentés), il doit être capable de fournir des résultats conformes à la réalité en ce qui concerne des systèmes existants. Cette étape de validation nécessaire a été effectuée en comparant des résultats de simulation avec les performances réelles de quelques SGBDOO, mesurées grâce à OCB. Cette étude est présentée dans le Chapitre 7.

Chapitre 5
L'environnement de simulation DESP-C++

Nous présentons dans ce chapitre le simulateur que nous avons conçu afin de permettre une implémentation efficace de notre modèle de simulation VOODB (cf. Chapitre 4). Cet outil a été baptisé DESP-C++ (*Discrete-Event Simulation Package for C++*). Nous expliquons les motivations qui nous ont entraîné à concevoir DESP-C++ plutôt que d'utiliser un logiciel de simulation existant. Nous décrivons en détail l'architecture de DESP-C++ et proposons un court aperçu sur sa mise en œuvre. Nous abordons enfin le problème de la validation de notre outil et avançons des éléments jouant en faveur de sa validité.

1. Introduction

De nombreux outils de simulation aléatoire sont disponibles aujourd'hui. Plusieurs langages de simulation généraux ont été développés depuis le milieu des années 60, comme SIMULA [Dahl et Nygaard 66], GPSS [Herscovitch et Schneider 65], SLAM II [Pritsker 86], SIMAN [Pegden et al. 90] ou QNAP2 [Simulog 95]. Ils proposent diverses fonctionnalités et sont considérés comme valides. Bien qu'ils demandent tous un certain temps d'apprentissage, une fois leurs concepts acquis, ils rendent la conception de différents modèles de simulation bien plus facile que s'il fallait programmer un simulateur dédié pour chaque modèle. Cependant, les simulateurs dédiés restent utiles lorsque de bonnes performances sont requises à l'exécution des modèles. De plus, la plupart des langages de simulation généraux ne permettent pas réellement l'utilisation d'une approche orientée objet. Plusieurs environnements et langages de simulation orientés objet commerciaux ont été conçus depuis une dizaine d'années (MODSIM III [Bryan 89], SIMPLE++ [AESOP 95], Silk [Healy et Kilgore 97] et SimJava [Page et al. 97], basés sur le langage Java…), mais ils demandent également un temps d'apprentissage substantiel. C'est pourquoi des outils de simulation plus simples sont parallèlement apparus en tant que packages C++ ou Java. DESP-C++ est l'un d'entre eux.

DESP-C++ a initialement été conçu pour remédier aux carences de QNAP2 en terme de temps d'exécution pour le modèle de simulation VOODB (cf. Chapitre 4, Section

2.3.4.1). Les temps de simulation très longs nécessaires à QNAP2 étaient dus principalement à la surcouche chargée de la gestion de la base d'objets, qui manipule des structures de données complexes, ainsi qu'aux fonctionnalités orientées objet que nous avons utilisées pour étendre le langage standard de QNAP2 [Hill 93].

En résumé, nous avions besoin d'un langage de simulation rapide, bon marché et raisonnablement simple. Le langage C++ [Stroustrup 97] est apparu comme une alternative avantageuse, à la condition que nous codions un moteur de simulation. Pour des raisons de facilité d'implémentation, nous avons opté pour un noyau de simulation aléatoire à événements discrets. Cela a en fait facilité l'adaptation du modèle QNAP2, car QNAP2 emploie également cette approche.

2. Caractéristiques et fonctionnalités de DESP-C++

2.1. Caractéristiques de DESP-C++

La motivation de concevoir notre propre simulateur vient du fait que les outils existants auxquels nous avions accès ne correspondaient pas à nos besoins, principalement en termes de validité et de simplicité d'emploi. Les qualités que nous avons voulu donner à DESP-C++ sont les suivantes.

- *Validité :* Pour fournir des résultats de simulation fiables, DESP-C++ devait être validé, c'est-à-dire que nous devions nous assurer que le moteur de simulation se comportait bien de la façon prévue. Pour cela, nous avons implémenté les mêmes modèles en QNAP2 et avec DESP-C++, puis vérifié que les résultats concordaient (cf. Section 5). La validité est un point important et c'est en fait la raison pour laquelle nous n'avons pas sélectionné un package de simulation C++ existant et avons choisi de bâtir le nôtre. Par exemple, absolument aucune expérience de validation n'est présentée dans [Little et McCue] pour C++SIM. [Fishwick 92] ne propose que des exemples possibles de modèles SimPack, mais sans indiquer si ces modèles sont fonctionnellement corrects, c'est-à-dire cohérents avec le système réel qu'ils modélisent.
- *Simplicité :* Les bases concernant la programmation orientée objet, la modélisation et la simulation étant acquises, nous désirions vivement que DESP-C++ soit très facile à utiliser, en comparaison de logiciels de simulation complexes comme SLAM II, QNAP2 ou même des autres packages de simulation C++ comme SimPack et C++SIM, qui proposent tous beaucoup plus de fonctionnalités que ce dont nous avons réellement besoin.
- *Efficacité :* Nos expériences de simulation avec QNAP2 s'étant révélées trop longues à réaliser, nous avions besoin que DESP-C++ soit raisonnablement rapide.

- *Portabilité :* Puisque nos modèles de simulation étaient susceptibles d'être utilisés sur plusieurs plates-formes (stations de travail Sun, Silicon Graphics ou IBM, micro-ordinateurs sous Linux ou Windows), son code devait être portable. C'est une raison supplémentaire qui nous a fait choisir C++ comme langage de programmation pour DESP.
- *Compacité :* Afin de rester simple et extensible, notre noyau de simulation devait être assez petit et compréhensible. Le code de DESP-C++ a en effet une longueur de moins de 1300 lignes, ceci incluant quelques fonctions utilitaires.
- *Extensibilité :* Nous désirions autoriser la possibilité de développer n'importe quel type de modèle à partir de notre noyau de simulation, y compris des modèles complexes. C'est pourquoi nous avons choisi une structure « ouverte » et simple, très facile à modifier et à étendre.
- *Universalité :* Le langage C++ est dorénavant utilisé par un nombre grandissant de programmeurs. En l'adoptant, nous évitons à l'utilisateur de DESP-C++ l'apprentissage d'une syntaxe spécifique et permettons la création rapide de modèles de simulation.
- *Bas prix :* DESP-C++ est disponible en *freeware*.

2.2. Fonctionnalités de DESP-C++

Concrètement, DESP-C++ fournit un ensemble de classes permettant de gérer et d'ordonnancer des événements de simulation, à l'instar de n'importe quel simulateur à événements discrets. La simulation à événements discrets peut s'appliquer à tout système discret (c'est-à-dire, dont l'évolution dans le temps s'accomplit de manière discrète), ce qui couvre une très large gamme de systèmes [Leroudier 80].

Par ailleurs, DESP-C++ utilise une *approche station*. Dans l'approche station, l'observateur (ou le concepteur) décrit, dans le formalisme choisi, le fonctionnement de chaque ressource active du système [Gourgand et Kellert 93]. Il définit en fait les relations qui lient les ressources actives aux diverses entités passives qui les visitent. Au contraire, dans l'*approche transaction*, l'observateur décrit le fonctionnement du système en spécifiant, pour chaque type de flux d'entités qui traversent ce système, le cheminement de ces entités et les traitements successifs qu'elles subissent [Gourgand et Kellert 93].

Nous avons dans un premier temps considéré le choix de l'approche transaction, car elle répondait bien à nos besoins avec le modèle de simulation VOODB, qui simule l'exécution de transactions au sein d'un système de gestion de bases de données orienté objet. Cependant, QNAP2 utilise l'approche station et nous avons souhaité une adaptation facile de VOODB de QNAP2 à DESP-C++. C'est pourquoi nous avons finalement opté pour l'approche station.

Ainsi, avec DESP-C++, le système à simuler est décrit par un réseau de files d'attentes constitué d'un ensemble de ressources communiquant entre elles. Ces ressources sont divisées en deux catégories : les *ressources actives* qui accomplissent réellement une tâche et les ressources passives qui ne participent directement à aucun traitement, mais sont utilisées par les ressources actives pour effectuer leurs opérations. La tâche des utilisateurs de DESP-C++ est d'instancier les classes « ressource » en spécifiant leurs paramètres et les événements qui leur sont associés (pour les ressources actives). Des *clients* voyagent au sein du réseau constitué par les ressources actives et sont « servis » par ces dernières.

Par exemple, soit un ordinateur où plusieurs processus exécutent des programmes en parallèle (Figure 5.1). Les programmes peuvent être vus comme les clients du système et les processus comme les ressources actives. Le processeur, la mémoire primaire, le(s) disque(s) dur(s), etc. constituent les ressources passives du système.

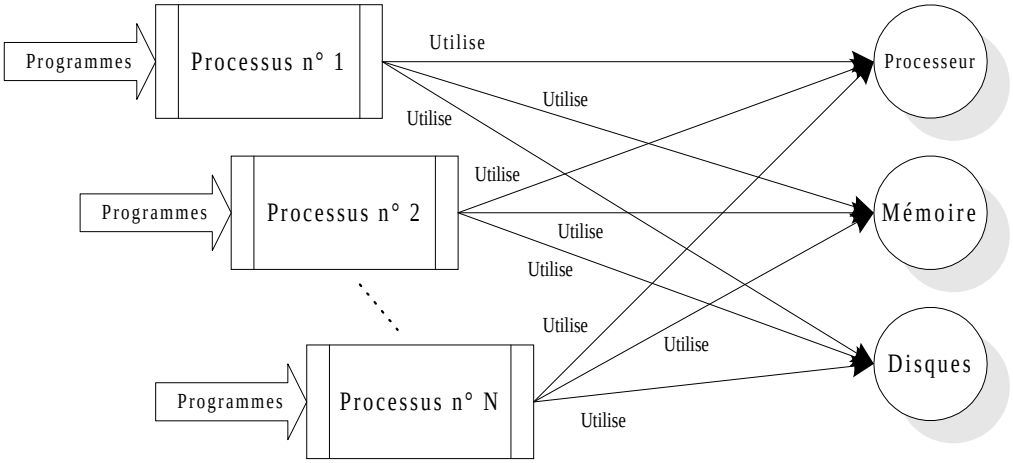

Figure 5.1 : Exemple – Traitements parallèles au sein d'un ordinateur

Le comportement d'un modèle de simulation est évalué grâce à un ensemble de statistiques (concrètement, des valeurs moyennes et des écarts-types). Les intervalles de confiance sont calculés par la réplication des expériences de simulation. Par défaut, DESP-C++ fournit les statistiques suivantes pour chaque ressource (qu'elle soit active ou passive) :

- temps de réponse moyen,
- temps d'attente moyen pour les clients (avant d'être servis),
- nombre moyen de clients servis,
- nombre moyen de clients dont le service est en cours,
- nombre moyen de clients en attente d'être servis.

3. Architecture de DESP

3.1. Architecture globale

L'architecture complète de DESP est présentée dans la Figure 5.2 sous la forme d'un diagramme de classes UML. DESP est organisé autour de la classe *Simulation*, dont les attributs sont les données de contrôle de la simulation (date de début et de fin de simulation, date courante, germe du générateur aléatoire). La classe *Simulation* comporte également une liste de références vers tous les clients présents dans le système, afin qu'ils puissent être détruits en fin de simulation et que la mémoire soit libérée. La classe *Simulation* constitue l'interface de DESP. Elle doit être instanciée dans le programme principal qui fait appel à DESP-C++. Les simulations sont activées par la méthode *Run()*, le nombre de réplications étant indiqué en paramètre.

Chaque instance de la classe *Simulation* est liée à un objet échéancier (*Scheduler*), qui est constitué pratiquement d'une liste d'événements à exécuter (triés par date d'événement). Chaque événement possède également un code unique et est lié à l'objet *Client* qui « subit » cet événement. Les méthodes de l'échéancier gèrent la liste des événements (insertion, suppression, recherche d'un événement).

Une instance de la classe Simulation est également liée à un objet gestionnaire d'événements (*EventManager*) qui administre l'exécution des événements par les ressources actives (objets *ActiveResource*) en utilisant les ressources passives (objets *Resource*), à travers la méthode *ExecuteEvent()*. Le gestionnaire d'événements est également chargé de déclencher l'initialisation et le calcul des statistiques pour chaque ressource du système. La méthode *Init()* initialise les statistiques pour toute une expérience de simulation. La méthode *InitRep()* fait de même pour une réplication donnée. La méthode *Stats()* calcule des statistiques intermédiaires concernant une réplication. Finalement, la méthode *DisplayStats()* calcule et affiche les résultats statistiques finaux.

Un objet ressource (*Resource*) est essentiellement une file d'attente d'événements triés par ordre de priorité, chaque événement étant de nouveau associé à un objet *Client*. Chaque ressource est définie par son nom (*name*), qui n'est pas nécessairement unique, mais devrait l'être de préférence pour des raisons de clarté des résultats, ainsi que par sa capacité maximum (*capacity*), c'est-à-dire le nombre maximum de clients pouvant être servis en même temps. La capacité courante *ccapacity* indique combien de clients supplémentaires potentiels peuvent utiliser la ressource. Les méthodes typiques *P()* et *V()*, qui servent à réserver et restituer la ressource, respectivement, constituent l'interface d'une ressource. Des méthodes privées administrent la file d'attente (insertion, suppression, recherche d'un événement).

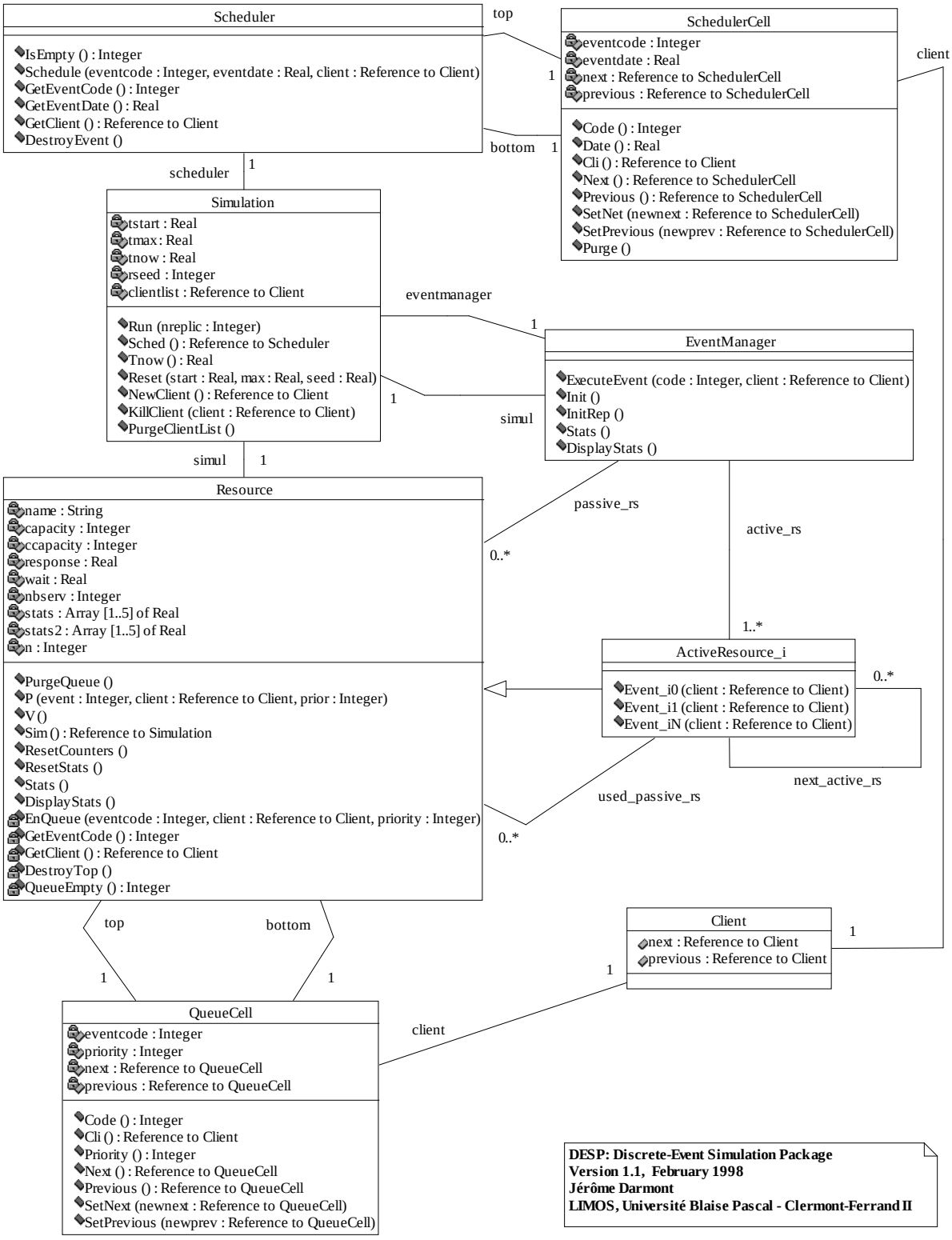

Figure 5.2 : Architecture de DESP

Une ressource possède également des attributs (les compteurs *wait*, *response*, *stats[]*, etc.) et des méthodes relevant de la gestion des statistiques au niveau de la ressource individuelle (initialisation globale, initialisation par réplication, calcul par réplication, calcul global des statistiques concernant cette ressource). Ces méthodes sont appelées par

le gestionnaire d'événements pendant les phases correspondantes de mise à jour des statistiques.

Toutes les ressources actives héritent de la classe *Resource*. Elles incluent juste en supplément des méthodes contenant le code d'exécution des événements qui leur sont associés. L'utilisateur de DESP peut leur ajouter des attributs publics supplémentaires, si cela s'avère nécessaire pour un modèle particulier.

Pour finir, les *Clients*, en tant que simples entités passives parcourant le système, sont uniquement conçus comme des éléments de listes chaînées. Cependant, ils peuvent être personnalisés par l'utilisateur pour transporter n'importe quel type d'information : il suffit d'ajouter des attributs publics à la classe *Client*. Par exemple, ces données peuvent être utilisées par les ressources actives afin d'effectuer un traitement personnalisé pour chaque client.

3.2. Organisation en modules

Toutes les classes présentées à la Section 3.1 sont de surcroît organisées en fichiers et en modules, comme le montre la Figure 5.3. Sur la gauche de la figure se trouvent les modules *Simulation* et *Utilitaires*, qui ne peuvent pas normalement être modifiés par l'utilisateur. Ils contiennent diverses fonctions utilitaires, dont le générateur aléatoire Lewis-Payne [Lewis et Payne 73] (qui est le meilleur générateur de nombres pseudoaléatoires existant actuellement, en raison de sa très grande période), l'implémentation de plusieurs types de lois de distribution aléatoire, ainsi que le moteur de simulation lui-même. Sur la droite de la figure se trouve le module des *Événements*, qui lui peut être modifié. Il contient la définition des ressources du système, celle des clients et celle des événements de simulation. Les flèches sur la Figure 5.3 montrent comment les trois modules utilisent les méthodes de chacun des autres. Les fichiers `*c.h` contiennent la définition des classes et les fichiers `*m.h` le code de méthodes. Les autres fichiers contiennent des fonctions utilitaires.

3.3. Fonctionnement du noyau de simulation

Le noyau de simulation en lui-même est très simple. Son code C++ complet est présenté dans la Figure 5.4. Son fonctionnement de base est le suivant :

1. les statistiques globales sont initialisées ;
2. pour chaque réplication :

    2.1. les statistiques concernant la réplication courante sont initialisées,

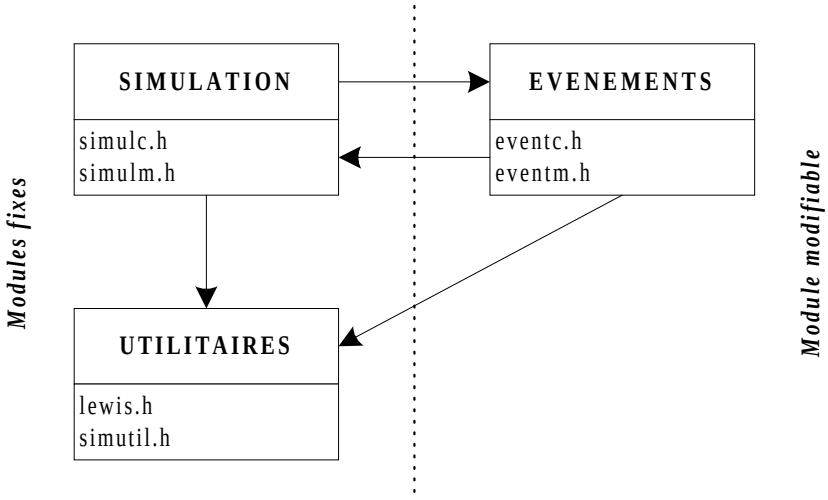

Figure 5.3 : Modules de DESP

2.2. tant que la réplication n'est pas terminée, des événements fournis par l'échéancier (*Scheduler*) sont exécutés par le gestionnaire d'événements (*EventManager*). Bien sûr, les événements eux-mêmes peuvent planifier d'autres événements de façon à ce que le processus se répète,

2.3. les statistiques concernant la réplication courante sont calculées,

2.4. tous les objets *Client* restant dans le système sont détruits afin que la réplication suivante ne soit pas influencée par leur présence,

3. les statistiques globales sont calculées et affichées.

4. Mise en œuvre de DESP-C++

Une fois l'étape de modélisation d'un système effectuée (cf. Chapitre 4), la traduction d'un modèle en C++ est aisée. Programmer un modèle de simulation à événements discrets avec DESP-C++ se fait principalement en alimentant le module *Événements* de la Figure 5.3, c'est-à-dire en spécifiant les ressources du système et les événements de simulation. Ceci s'effectue en trois étapes.

1. Édition du fichier `eventc.h`.
   - Toutes les ressources actives doivent être définies comme des classes héritant de la classe *Resource*. Chaque ressource active doit posséder un pointeur vers toutes les ressources passives qu'elle utilise (comme la ressource appelée *Passive* dans la Figure 5.5) et toutes les autres ressources actives vers lesquelles elle est susceptible d'envoyer des clients.

```
// CLASSE Simulation: Exécution des simulations

void Simulation::Run(int nreplic) {

  int i, nextevent;
  Client *client;

  // Initialisation globale
  eventmanager->Init();

  // Boucle des réplications
  for (i=1; i<=nreplic; i++) {

    // Initialisation des réplications
    tnow=tstart;
    eventmanager->InitRep();
    client=NewClient();
    eventmanager->ExecuteEvent(0,client); // Planification premier événement

    // Moteur de simulation
    while ((tnow<tmax) && (!scheduler->IsEmpty())) {
      nextevent=scheduler->GetEventCode();
      tnow=scheduler->GetEventDate();
      client=scheduler->GetClient();
      scheduler->DestroyEvent();
      eventmanager->ExecuteEvent(nextevent,client);
    }

    // Calcul des statistiques concernant la réplication
    eventmanager->Stats();

    // Destruction des clients encore présents dans le système
    PurgeClientList();
  }

// Résultats globaux
  eventmanager->DisplayStats();
}
```

Figure 5.4 : Code C++ du noyau de simulation de DESP

```
// Exemple de ressource active

class Sample_AR: public Resource {
  public:
    // Constructeur
    Sample_AR(char name[STRS], int capacity, Simulation *sim, Resource *passive);
    // Événements associés à la ressource Sample_AR
    void AR_Event0(Client *client);
    void AR_Event1(Client *client);
    void AR_Event2(Client *client);
    void AR_Event3(Client *client);
  private:
    Resource *Passive;
};
```

Figure 5.5 : Exemple de définition de ressource active

- Des pointeurs vers toutes les ressources actives et passives doivent être déclarés comme attributs de la classe *EventManager* (Figure 5.6).

```
class EventManager {
  // Méthodes publiques (skipped)
  private:
    // Attributs
    Simulation *simul; // Pointeur vers l'objet Simulation
    // Ressources passives
    Resource *sample_pr;
    // Ressources actives
    Sample_AR *sample_ar;
};
```

Figure 5.6 : Déclaration des ressources dans la classe *EventManager*

- Si nécessaire, de nouveaux attributs peuvent être ajoutés à la classe *Client* (Figure 5.7).

```
class Client {
  public :
    // Attributs habituels
    Client *next;
    Client *previous;
    // Attributs supplémentaires
    float operating_time;
};
```

Figure 5.7 : Définition d'attributs supplémentaires dans la classe *Client*

2. Édition du fichier `eventm.h`.

   - Dans le constructeur et le destructeur de la classe *EventManager*, il faut respectivement instancier ou détruire toutes les ressources actives et passives (Figure 5.8).
   - Dans la classe *EventManager* et la méthode *ExecuteEvent(),* pour chaque ressource active et chaque événement associé, il faut ajouter une ligne destinée à déclencher cet événement (Figure 5.9).
   - Toutes les ressources actives et passives doivent être prises en compte dans les autres méthodes de la classe *EventManager*. Un exemple est donné pour la méthode *Init()* dans la Figure 5.10.

```
// CLASSE EventManager : Constructeur

EventManager::EventManager(Simulation *sim) {
  simul=sim;
  // Instanciation des ressources passives
  sample_pr=new Resource("PR",2,simul);
  // Instanciation des ressources actives
  sample_ar=new Sample_AR("AR",1,simul,sample_pr);
}

// CLASSE EventManager : Destructeur

EventManager::~EventManager() {
  // Destruction des ressources passives
  delete sample_pr;
  // Destruction des ressources actives
  delete sample_ar;
}
```

Figure 5.8 : Instanciation et destruction des ressources

```
// CLASSE EventManager : Exécution des événements

void EventManager::ExecuteEvent(int code, Client *client) {
  switch(code) {
  case 0: sample_ar->AR_Event0(client);break; // Événement initial OBLIGATOIRE !!
  case 1: sample_ar->AR_Event1(client);break;
  case 2: sample_ar->AR_Event2(client);break;
  case 3: sample_ar->AR_Event3(client);break;
  default: printf("Error: unknown event #%d at time %f\n",code,simul->Tnow());
  }
}
```

Figure 5.9 : Déclenchement des événements dans la méthode *ExecuteEvent()*

```
void EventManager::Init() {
  // Ressources passives
  sample_pr->ResetStats();
  // Ressources actives
  sample_ar->ResetStats();
}
```

Figure 5.10 : Initialisation des statistiques des ressources

- Le constructeur de chaque ressource active doit être spécifié s'il diffère du constructeur de la classe standard *Resource*. Chaque événement déclenché par une ressource active doit également figurer en tant que méthode de cette ressource active (Figure 5.11).

```
// CLASSE Sample_AR : Constructeur

Sample_AR::Sample_AR(char name[STRS], int capacity, Simulation *sim, Resource
*passive):Resource(name, capacity, sim) {Passive=passive;}

// CLASSE Sample_AR : Événement 0, réservation de la ressource active

void Sample_AR::Event0(Client *client) {
  this->P(1,client,1); // événement suivant: #1, priorité dans la file: 1 }

// CLASSE Sample_AR : Événement 1, réservation de la ressource passive

void Sample_AR::Event1(Client *client) {
  Resource->P(2,client,1); // événement suivant: #2, priorité dans la file: 1 }

// CLASSE Sample_AR : Événement 2, traitement d'une opération

void Sample_AR::Event2(Client *client) {
  Sim()->Sched()->Schedule(3,Sim()->Tnow()+client->operating_time,client);
  // événement suivant: #3, planifié après le temps operating_time
}

// CLASSE Sample_AR : Événement 3, libération des ressources

void Sample_AR::Event3(Client *client) {
  Resource->V();
  this->V();
  Sim()->Sched()->Schedule(0,Sim()->Tnow(),client);
  // réitère le processus (événement 0)
}
```

Figure 5.11 : Méthodes de la ressource active prise comme exemple

3. Pour finir, il faut écrire un programme principal. C'est la partie la plus facile. Il suffit simplement d'inclure les modules de DESP-C++, de créer un objet *Simulation* et d'exécuter sa méthode *Run()*. Un exemple est fourni en Figure 5.12.

```
// Exemple de programme utilisant DESP-C++

#include "simutil.h"
#include "simulc.h"
#include "eventc.h"
#include "simulm.h"
#include "eventm.h"

void main() {
  Simulation *sim = new Simulation(TEMPS_DEBUT, TEMPS_FIN, GERME_ALEATOIRE);
  sim->Run(NOMBRE_DE_REPLICATIONS);
}
```

Figure 5.12 : Exemple de programme principal

5. Expériences de validation

Être capable d'effectuer des simulations est une chose, mais obtenir des résultats valides et fiables en est une autre. Afin de prouver que notre moteur de simulation fournit effectivement des résultats corrects, nous avons décidé d'implémenter des modèles identiques avec DESP-C++ et QNAP2, puis de comparer les résultats de simulation. Comme QNAP2 est un outil validé et fiable, des résultats concordants doivent permettre de valider DESP-C++ à son tour.

Nous avons commencé par étudier un modèle classique de « flow shop » très simple (ce qui nous a également aider à déboguer notre moteur de simulation), puis nous nous sommes intéressés à un problème un peu plus complexe en termes de partage de ressources : le problème des philosophes. Le processus de validation a été complété par la comparaison des résultats de simulation concernant VOODB, qui est un modèle de simulation bien plus complexe.

5.1. « Flow shop » simple

Notre premier modèle concerne le système de production organisé en « flow shop » qui est présenté dans la Figure 5.13. Des matières premières subissent une première transformation dans la Machine n° 1 pendant un temps dépendant d'une loi aléatoire exponentielle (de moyenne 10 minutes). Les produits semi-finis sont ensuite transportés par un robot mobile jusqu'à un stock tampon en amont de la Machine n° 2. Le temps de transport dépend d'une loi aléatoire uniforme (dont les valeurs s'échelonnent entre 4 et 6 minutes). Les produits semi-finis subissent ensuite une seconde transformation dans la Machine n° 2 pendant un temps dépendant d'une loi aléatoire exponentielle (de moyenne 12 minutes). Les produits finis sont finalement transportés hors du système par le robot mobile. Le temps de transport dépend toujours d'une loi aléatoire uniforme (dont les valeurs s'échelonnent entre 4 et 6 minutes).

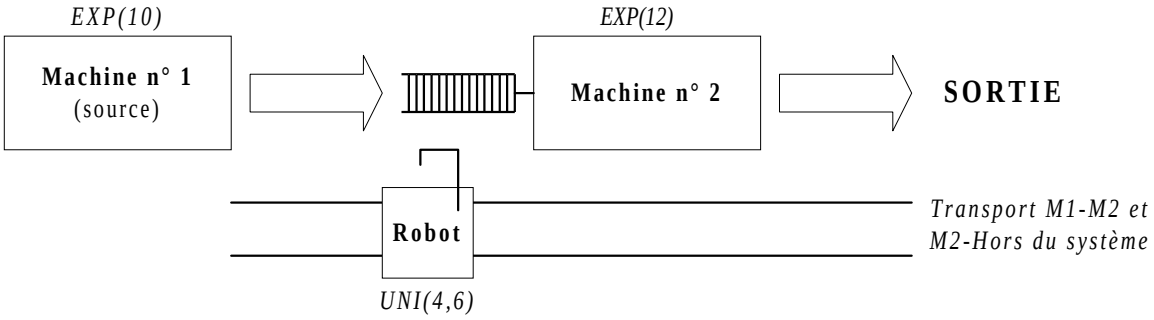

Figure 5.13 : Système de production simple

La Figure 5.14 illustre la façon dont la méthodologie de modélisation que nous proposons s'applique pour produire un modèle de ce système de production simple. Ce diagramme d'activité UML montre le processus de transformation subi par les clients (les produits) au sein des ressources actives (Machine n° 1 et Machine n° 2), qui constituent les lignes d'eau (*swimlanes*) du diagramme. Bien que les ressources passives n'apparaissent pas sur la Figure 5.14, elles doivent impérativement être indiquées. Ici, le système n'a qu'une ressource passive : le robot mobile qui transporte les produits.

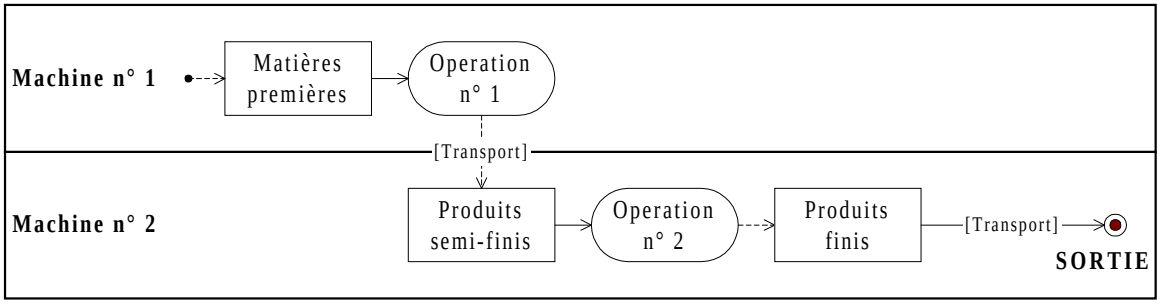

Figure 5.14 : Modèle de connaissance du « flow shop »

Afin d'évaluer la conformité des résultats, nous avons comparé le temps de réponse et le nombre de clients servis par chaque ressource, calculés par QNAP2 et DESP-C++. Nous avons également fait varier le nombre de réplications des simulations. Les résultats obtenus, présentés dans la Figure 5.15 et la Figure 5.16, montrent que DESP-C++ fournit exactement les mêmes résultats que QNAP2.

Par ailleurs, nous avons mesuré le temps d'exécution pour ces deux modèles, de façon à vérifier que le gain de performance avec DESP-C++ est suffisant. Le modèle de simulation QNAP2 a été exécuté sur une station de travail Silicon Graphics sous IRIX V4. Le modèle de simulation DESP-C++ a été exécuté sur un PC Pentium 133 sous Windows 95. En moyenne, les temps de simulation sont 20 fois plus courts avec DESP-C++ qu'avec QNAP2 (Figure 5.17).

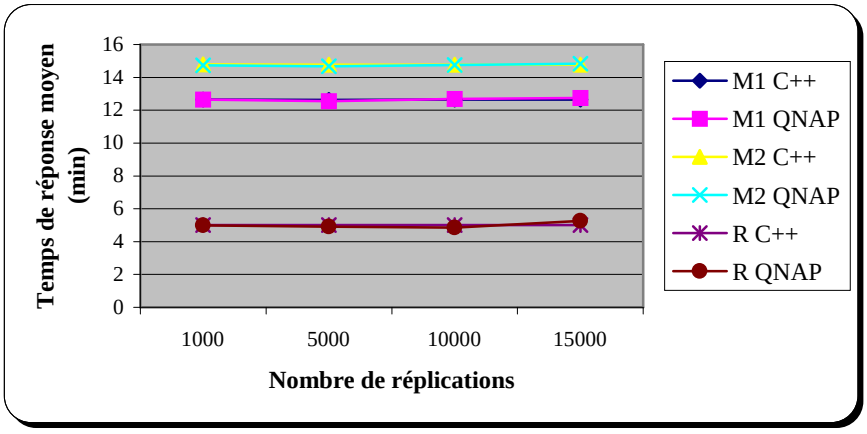

Figure 5.15 : Comparaison du temps de réponse pour QNAP2 et DESP-C++ (*flow shop*)

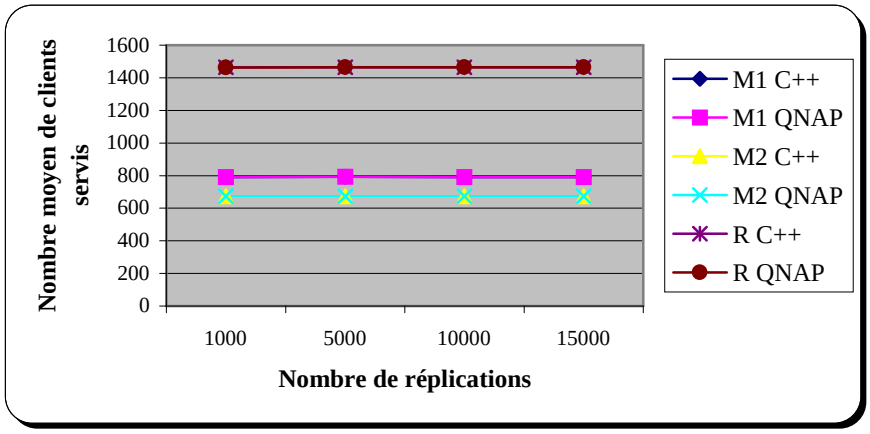

Figure 5.16 : Comparaison du nombre de clients servis pour QNAP2 et DESP-C++ (*flow shop*)

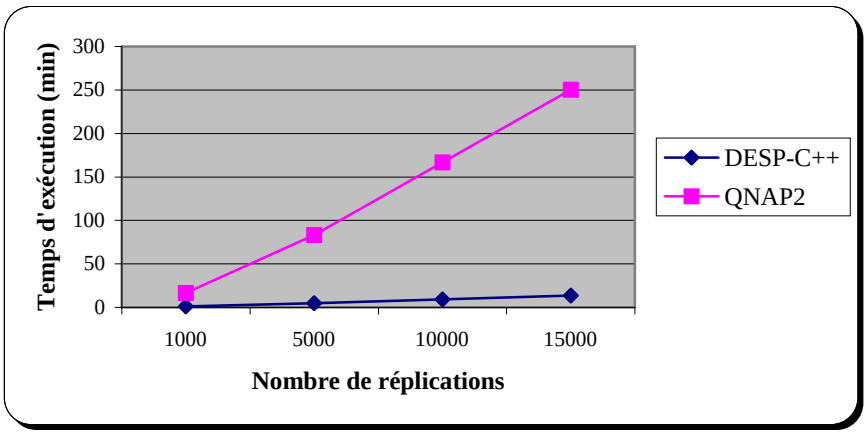

Figure 5.17 : Comparaison du temps d'exécution pour QNAP2 et DESP-C++ (*flow shop*)

Cette expérience a constitué une première et encourageante validation de DESP-C++. Cependant, il était nécessaire de vérifier que les résultats étaient aussi valables avec des modèles plus élaborés.

5.2. Problème des philosophes

Pour poursuivre notre processus de validation, nous avons considéré le problème classique des philosophes (Figure 5.18), qui présente un cas d'interblocage intéressant. Quatre philosophes qui ne font rien d'autre que méditer (pendant un temps dépendant d'une loi aléatoire exponentielle de moyenne 2 minutes) et manger (pendant un temps dépendant d'une loi aléatoire exponentielle de moyenne 5 minutes) sont assis autour d'une table. Entre chaque philosophe se trouve une seule fourchette. Or, un philosophe a besoin de deux fourchettes pour manger.

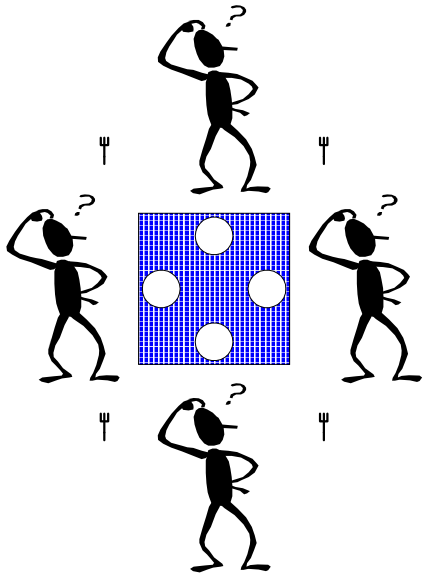

Figure 5.18 : Problème des philosophes

L'application de la méthodologie de modélisation que nous préconisons à ce problème aboutit au modèle présenté dans la Figure 5.19 sous la forme d'un diagramme d'activité. Il décrit le comportement de chaque philosophe. Les philosophes constituent les ressources actives de ce système et les fourchettes les ressources passives.

Nous avons à nouveau comparé le temps de réponse et le nombre de clients servis par chaque ressource du système (les quatre philosophes et les quatre fourchettes), calculés par QNAP2 et DESP-C++ (et toujours en variant le nombre de réplication des simulations). Les résultats présentés dans la Figure 5.20 et la Figure 5.21 montrent que DESP-C++ fournit à nouveau les mêmes résultats que QNAP2.

Nous avons également mesuré le temps d'exécution pour les deux modèles (QNAP2 et DESP-C++). Cette fois-ci, le modèle QNAP2 a été exécuté sur une station de travail IBM RISC 6000 sous AIX version 4. Le modèle DESP-C++ a lui été exécuté sur un PC Pentium-II 266 sous Windows 95 (OSR2). En moyenne, les simulations avec DESP-C++ se sont à nouveau déroulées 20 fois plus rapidement qu'avec QNAP2 (Figure 5.22).

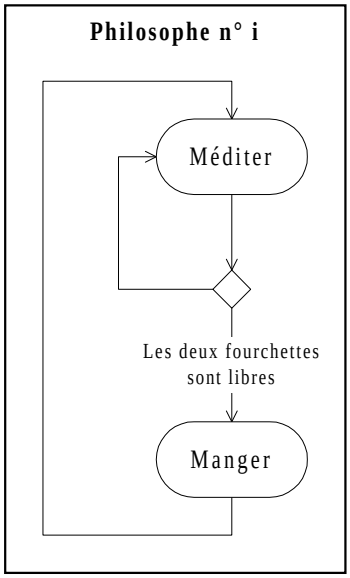

Figure 5.19 : Modèle de connaissance du comportement individuel des philosophes

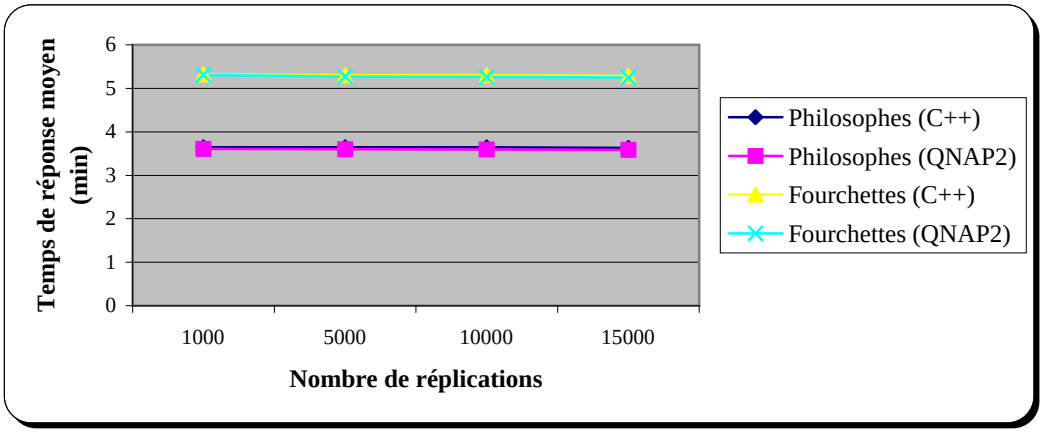

Figure 5.20 : Comparaison du temps de réponse pour QNAP2 et DESP-C++ (philosophes)

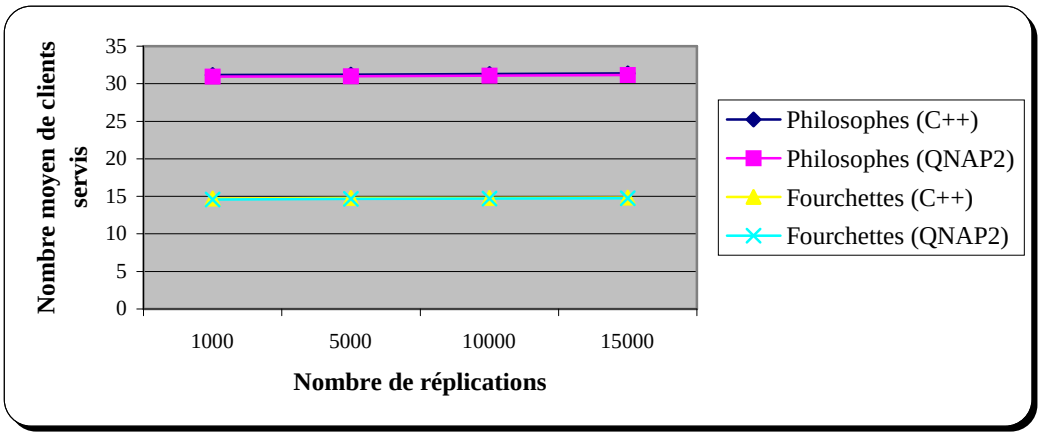

Figure 5.21 : Comparaison du nombre de clients servis pour QNAP2 et DESP-C++ (philosophes)

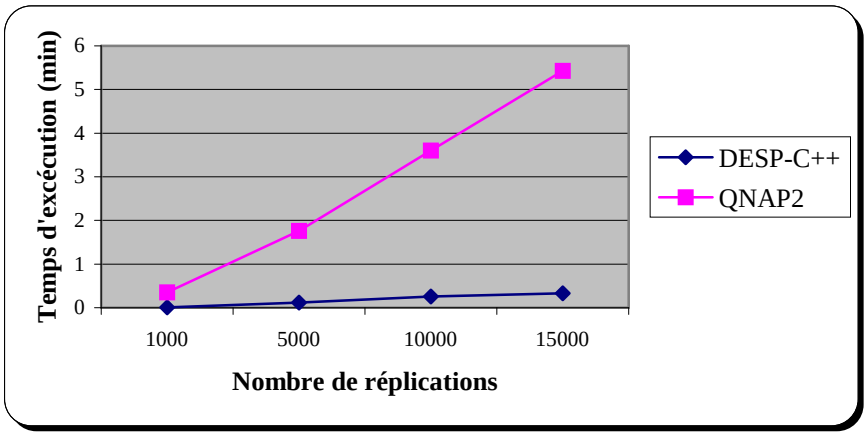

Figure 5.22 : Comparaison du temps d'exécution pour QNAP2 et DESP-C++ (philosophes)

5.3. VOODB

5.3.1. Contexte

Nous présentons dans cette section des résultats de simulation obtenus avec deux versions de notre modèle de simulation VOODB (une version QNAP2 et une version DESP-C++), qui est présenté dans le Chapitre 4. Des résultats concordants pour une même configuration de VOODB seraient une bonne indication de la validité de notre simulateur.

Comme l'objectif de VOODB est de permettre la comparaison des performances de techniques de groupement d'objets, nous avons inclus dans notre modèle l'algorithme DSTC. Cependant, afin de procéder à la validation de DESP-C++ étape par étape, nous n'avons pas dans un premier temps inclus le regroupement d'objets dans nos expériences. Les effets de DSTC ont été simulés avec QNAP2 et DESP-C++ dans une seconde série de tests.

Nous avons utilisé les paramètres par défaut de VOODB pour toutes ces simulations. Le temps de simulation a été fixé à 1000000 millisecondes, ce qui correspond à l'exécution d'environ 250 transactions, compte tenu du temps de latence entre deux transactions qui est par défaut fixé à 4 secondes. Chaque simulation a été répliquée 100 fois afin d'obtenir des résultats moyens significatifs.

QNAP2 est installé sur une station de travail IBM RISC 6000 sous AIX version 4. DESP-C++ a été utilisé sur un PC Pentium-II MMX 266 sous Windows 95 (OSR2).

5.3.2. Simulation sans groupement d'objets

5.3.2.1. Taille des modèles et temps d'exécution

Nous comparons en premier lieu la taille respective des modèles QNAP2 et DESP-C++, ainsi que le temps nécessaire à leur exécution (Table 5.1). Le rapport entre les données concernant QNAP2 et celles concernant DESP-C++ est également présenté.

Environnement	**QNAP2**	**DESP-C++**	**Rapport**
Lignes de code	2630	4333	0,6
Temps d'exécution (min)	6000	30	200

Table 5.1 : Taille et temps d'exécution des modèles de simulation (pas de groupement)

Le modèle de simulation QNAP2 paraît plus compact que le modèle DESP-C++. Cependant, le compte des lignes de code DESP-C++ intègre la taille du noyau de simulation, qui est d'environ 1300 lignes. Ceci ramène la taille effective du code consacré à VOODB aux alentours de 3000 lignes, ce qui est équivalent à la taille du modèle QNAP2. Ces modèles sont de complexité moyenne, ce qui les rend relativement compréhensibles et donc modifiables par une tierce personne.

En ce qui concerne le temps d'exécution de ces modèles, l'objectif de pallier la lenteur du langage interprété de QNAP2 est pleinement atteint, puisque le modèle DESP-C++ s'exécute 200 fois plus vite que le modèle QNAP2. Il faut noter que les réplications durent en moyenne une heure avec QNAP2. C'est la raison pour laquelle le nombre de transactions exécutées lors de chaque simulation a été ramené à 250 seulement. Un nombre plus important de transactions entraîne des temps de simulation prohibitifs avec QNAP2.

5.3.2.2. Résultats de simulation

Les résultats de simulation proprement dits sont résumés dans les tables qui suivent. Ils sont tous présentés sous la forme suivante :

- résultats obtenus avec QNAP2 ;
- résultats obtenus avec DESP-C++ ;
- rapport entre les deux résultats obtenus. Notre objectif est d'obtenir des rapports entre les résultats QNAP2 et DESP-C++ aussi proches de 1 que possible.

La Table 5.2 indique le nombre moyen de transactions exécutées. Le nombre de transactions obtenues est comme prévu proche de 250, en moyenne. Les transactions sont pratiquement équiréparties entre les quatre types d'opérations utilisés dans VOODB, ce qui correspond bien au paramétrage de notre modèle. Les résultats obtenus avec QNAP2 et DESP-C++ sont très similaires.

	QNAP2	DESP-C++	Rapport
Accès ensemblistes	63,27	62,20	1,0172
Parcours simples	62,54	62,38	1,0026
Parcours hiérarchiques	60,90	63,01	0,9666
Parcours stochastiques	62,63	62,52	1,0018
TOTAL	249,36	250,11	0,9970

Table 5.2 : Nombre moyen de transactions exécutées

La Table 5.3 liste, par type de transaction, le temps de réponse du système (en secondes). Les temps de réponse obtenus sont directement proportionnels au nombre d'objets accédés par chaque type de transaction (Table 5.4). Les rapports entre les résultats fournis par QNAP2 et ceux fournis par DESP-C++ sont ici encore satisfaisants.

	QNAP2	DESP-C++	Rapport
Accès ensemblistes	4,17	4,09	1,0211
Parcours simples	4,08	4,03	1,0142
Parcours hiérarchiques	2,40	2,23	1,0803
Parcours stochastiques	0,53	0,50	1,0600
TOTAL	2,85	2,66	1,0717

Table 5.3 : Temps de réponse moyen par type de transaction (en s)

	QNAP2	DESP-C++	Rapport
Accès ensemblistes	93,55	92,11	1,0157
Parcours simples	94,20	91,84	1,0257
Parcours hiérarchiques	54,35	51,28	1,0599
Parcours stochastiques	12,20	11,09	1,1009
TOTAL	64,40	61,5	1,0472

Table 5.4 : Nombre moyen d'objets accédés par type de transaction

Pour finir, la Table 5.5 présente des mesures de performances plus globales, concernant l'espace disque nécessaire au stockage de la base d'objets, le nombre d'entrées-sorties nécessaires à l'exécution des transactions et le débit du système (exprimé en nombre de transactions par seconde). Le débit quasi-optimal du système s'explique par le fait que le temps moyen d'exécution des transactions (2,66 s) est inférieur au temps de latence entre deux transactions (4 s). Les résultats obtenus avec QNAP2 et DESP-C++ sont toujours très similaires.

	QNAP2	DESP-C++	Rapport
Nombre moyen de pages utilisées	2823,18	2731,58	1,0335
Nombre moyen d'E/S	15335,09	15085,40	1,0165
Débit moyen du système (tr./s)	0,2490	0,25	0,9963

Table 5.5 : Performances globales (pas de groupement)

Les résultats que nous venons d'exposer sont récapitulés dans la Figure 5.23, qui présente de façon synthétique le rapport entre les résultats fournis par QNAP2 et DESP-C++ pour les critères de performance de la Table 5.6. Globalement, les résultats de simulation sont homogènes à 97 %.

Critère de performance	Code
Nombre moyen d'objets accédés par les accès ensemblistes	(1)
Nombre moyen d'objets accédés par les parcours simples	(2)
Nombre moyen d'objets accédés par les parcours hiérarchiques	(3)
Nombre moyen d'objets accédés par les parcours stochastiques	(4)
Nombre moyen d'objets accédés (globalement)	(5)
Nombre moyen d'entrées-sorties	(6)
Débit moyen	(7)

Table 5.6 : Critères de performance (pas de groupement)

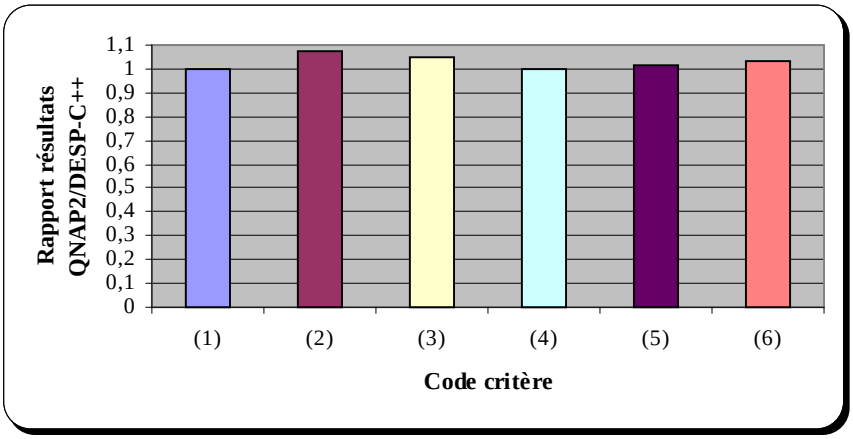

Figure 5.23 : Rapport entre les résultats obtenus avec QNAP2 et DESP-C++ (pas de groupement)

5.3.3. Prise en compte du groupement d'objets

Dans toutes ces premières expériences de simulation intégrant DSTC, nous n'avons pas fait varier les paramètres qui régissent le comportement de la stratégie de regroupement. En effet, notre but était de nous assurer que QNAP2 et DESP-C++ fournissent bien les mêmes résultats de simulation, et non d'optimiser l'usage de DSTC. Ces paramètres sont récapitulés dans la Table 5.7.

Paramètre	Valeur
Tf_a	2
Tf_e	1
Tf_c	1
w	0,7

Table 5.7 : Valeur des paramètres de DSTC en simulation

5.3.3.1. Taille des modèles et temps d'exécution

La taille des modèles QNAP2 et DESP-C++ intégrant l'algorithme de groupement d'objets DSTC, ainsi que leur temps d'exécution, sont présentés dans la Table 5.8. Grossièrement, l'introduction de DSTC au sein de VOODB ajoute environ 1000 lignes de code à chacun des modèles. Ils demeurent donc toujours identiques en termes de complexité.

Environnement	**QNAP2**	**DESP-C++**	**Rapport**
Lignes de code	3626	5440	0,6665
Temps d'exécution (min)	81000	63	1285,71

Table 5.8 : Taille et temps d'exécution des modèles de simulation (groupement)

Par ailleurs, la Table 5.8 illustre la raison pour laquelle nous avons dû abandonner le logiciel QNAP2 au profit de DESP-C++. Le temps d'exécution du modèle QNAP2 équivaut à plus de 56 jours complets de calcul. En fait, pour obtenir les résultats présentés dans la Section 5.3.3.2 raisonnablement rapidement, nous avons dû réduire le nombre de réplications de 100 à 10 avec QNAP2. Le temps de simulation réel a donc été de 8100 minutes (soit 5,6 jours). Nous avons extrapolé et multiplié ce résultat par 10 dans la Table 5.8, car nous avons conservé le nombre de réplications à 100 pour les simulations avec DESP-C++. Ces dernières s'exécutent en un peu plus d'une heure, ce qui nous permet de faire des expériences beaucoup plus nombreuses et plus variées avec DESP-C++ qu'avec QNAP2.

5.3.3.2. Résultats de simulation

La Table 5.9 présente les résultats de simulation obtenus avec QNAP2 et DESP-C++, ainsi que le rapport entre ces valeurs numériques. Les expériences de simulation que nous avons menées sont rigoureusement identiques à celles de la Section 5.3.2. La seule différence est le groupement dynamique des objets sur disque par DSTC. Nous avons évalué les critères de performances suivants, qui sont chacun associés à un code :

- *nombre moyen de transaction :* il se situe toujours aux alentours de 250, en moyenne ;
- *nombre moyen d'objets accédés par transaction :* nous n'avons pas détaillé ici le nombre moyen d'objets accédés pour chaque type de transaction. Les résultats sont parfaitement similaires à ceux présentés dans la Table 5.4 ;
- *temps de réponse moyen par transaction :* ici encore, nous n'avons pas détaillé le temps de réponse pour chaque type de transaction. On peut remarquer que, globalement, l'introduction du groupement d'objets permet d'améliorer le temps de réponse moyen ;

- *durée moyenne des regroupements :* cette durée fait apparaître la surcharge engendrée par le regroupement d'objets, en terme de temps ;
- *débit moyen du système :* comme le temps de réponse, il ne souffre pas de l'introduction du groupement. Il n'est pas non plus amélioré, étant déjà proche de l'optimum (cf. Section 5.3.2) ;
- *nombre d'entrées-sorties dues aux transactions :* c'est le nombre d'entrée-sorties nécessaires pour l'exécution des transactions uniquement. On peut noter que le groupement induit une baisse de ce nombre d'entrées-sorties, et donc une amélioration du temps de réponse ;
- *nombre d'entrées-sorties dues au groupement :* cette mesure représente la surcharge due au groupement des objets, en termes d'entrées-sorties nécessaires aux opérations de regroupement. La surcharge constatée (environ 250 E/S) est inférieure au gain réalisé sur les entrées-sorties (environ 2000 E/S) ;
- *nombre moyen de pages utilisées :* c'est le nombre de pages nécessaires au stockage de la base d'objets. Il est en légère augmentation, du fait que des objets ont été déplacés dans de nouvelles pages pour être regroupés ensemble.

	QNAP2	**DESP-C++**	**Rapport**	***Code***
Nombre moyen de transactions	246,00	250,73	0,9811	(1)
Nombre moyen d'objets accédés par tr.	67,29,	61,14	1,1006	(3)
Temps de réponse moyen par tr. (s)	2,12,	1,86	1,1407	(2)
Durée moyenne des regroupements (s)	0,10,	0,10	1,0636	(4)
Débit moyen du système (tr./s)	0,24,	0,25	0,9818	(5)
Nombre moyen d'E/S (transactions)	13073,18,	12261,67	1,0661	(6)
Nombre moyen d'E/S (groupement)	243,81,	259,02	0,9413	(7)
Nombre moyen de pages utilisées	3066,36,	3045,98	1,0066	(8)

Table 5.9 : Résultats de simulation QNAP2 et DESP-C++ (groupement)

La Figure 5.24 présente de façon synthétique le rapport entre les résultats fournis par QNAP2 et DESP-C++ pour les critères de performance de la Table 5.6. Les rapports sont un peu moins bons que ceux constatés à la Section 5.3.2. Globalement, les résultats de simulation sont homogènes à 96 %. Une hypothèse raisonnable est de penser que c'est le plus faible nombre de réplications avec QNAP2 qui est à l'origine de cette perte de cohérence dans nos résultats. Par ailleurs, les résultats fournis par la simulation sont plus à interpréter comme des tendances que des valeurs précises. Aussi pouvons-nous considérer que les résultats fournis par QNAP2 et DESP-C++ sont toujours cohérents après introduction du groupement d'objets dans VOODB.

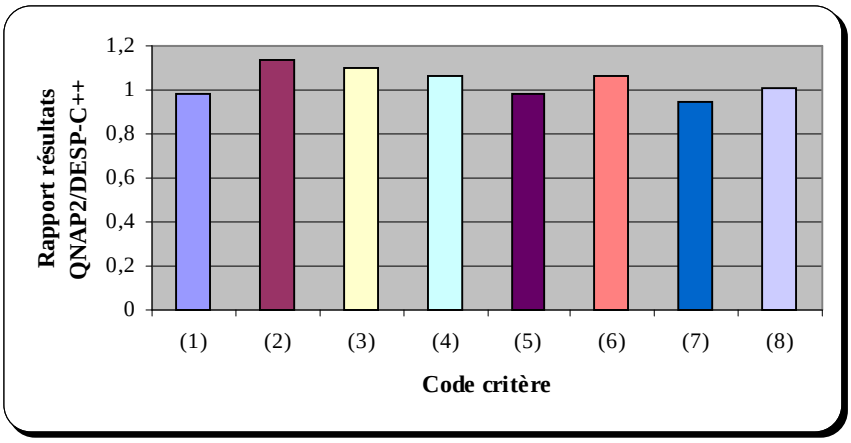

Figure 5.24 : Rapport entre les résultats obtenus avec QNAP2 et DESP-C++ (groupement)

6. Bilan

Nous avons fourni dans cette section des éléments permettant de démontrer que l'outil que nous proposons, DESP-C++, est un moteur de simulation valide. Pour ce faire, nous l'avons comparé avec succès à QNAP2, qui est un simulateur validé et fiable, en terme de résultats fournis. Par ce moyen, nous avons également illustré la flexibilité de notre solution, en concevant trois modèles de simulation très différents les uns des autres (un système de production, un problème d'interblocage et un SGBDOO).

Néanmoins, DESP-C++ peut être amélioré de multiples façons. Les outils statistiques que nous fournissons par défaut (réplication des simulations et calcul de valeurs moyennes et d'écarts-types) sont très simples. Des méthodes plus élaborées, comme la méthode de régénération ou la méthode spectrale pourraient permettre d'obtenir de meilleurs intervalles de confiance.

La conception C++ de notre outil pourrait également être améliorée, dans le but de rendre DESP-C++ plus facile à utiliser. Une réorganisation des modules fonctionnels (cf. Figure 5.3) ou une implémentation en tant que librairie C++ peuvent être envisagées. De même, une interface graphique adaptée pourrait grandement faciliter la spécification d'un modèle de simulation avec notre package.

Pour finir, certaines portions du code de DESP-C++ peuvent être optimisées afin de rendre les simulations encore plus rapides. Ce n'était pas une priorité pour nous, mais cela pourrait se révéler utile. Par exemple, les classes *Scheduler* et *Resource* utilisent actuellement des structures de données basiques pour leurs files d'attente (des listes chaînées bidirectionnelles). Des structures de données plus efficaces pourraient être employées à leur place, comme celles des librairies C++ LEDA [Mehlhorn et al. 95] ou STL [Stepanov et Lee 95], qui sont très puissantes. STL (*Standard Template Library*) est d'ailleurs devenue une librairie standard de C++ depuis 1998 [NSITC 98].

Chapitre 6
Systèmes et algorithme de regroupement utilisés

Nous appliquons dans le Chapitre 7 nos outils d'évaluation de performances à l'étude du SGBDOO O_2 [Deux et al. 91], du gestionnaire d'objets persistants Texas [Singhal et al. 92] et de la technique de regroupement dynamique d'objets DSTC [Bullat et Schneider 96]. Il est donc nécessaire de présenter en détail ces systèmes et cet algorithme de regroupement. C'est l'objectif de ce chapitre.

1. Système O_2

Nous nous intéressons plus particulièrement dans cette présentation aux aspects concernant l'implémentation du SGBDOO O_2, c'est-à-dire aux composants d'O_2 qui en définissent les performances. Le modèle de données d'O_2, son langage de définition de données (LDD) et les langages de programmation d'O_2 sont largement abordés dans la littérature. Aussi ne les détaillons-nous pas ici. Le lecteur intéressé pourra néanmoins se référer aux articles et ouvrages suivants : [Bancilhon et al. 88] [Deux et al. 90] [Deux et al. 91] [Bancilhon et al. 92] [Adiba et Collet 93].

1.1. Présentation d'O_2

O_2 [Deux et al. 91] [Vélez et al. 91] a été développé de 1986 à 1990 dans le cadre du Groupement d'Intérêt Public Altaïr comprenant l'INRIA, l'Université Paris XI, le CNRS, Bull et IN2 (une filiale de Siemens). À l'origine, O_2 a été conçu dans une optique « Recherche et Développement » pour faire partie d'une nouvelle génération de SGBD, alliant les fonctionnalités des SGBD classiques (persistance, accès concurrent, langage de requête…) à ceux des langages orientés objet (objets complexes, identité des objets, encapsulation, héritage, surcharge…) [Atkinson et al. 89]. O_2 est destiné aussi bien aux applications de gestion traditionnelles qu'aux applications d'ingénierie (CAO, PAO, AGL…).

O_2 a été commercialisé en 1991 par la société O_2 Technology, qui a par la suite fusionné avec les sociétés Unidata et VMARK pour former la compagnie Ardent Software. O_2 n'a cessé d'évoluer jusqu'à sa version actuelle (5.0), qui est disponible sur de nombreuses plates-formes (Sun, HP, IBM, Bull, Silicon Graphics, DEC, Windows NT, Linux).

1.2. Composants du système

La structure du SGBDOO O_2 est modulaire. Les composants d'O_2 sont récapitulés dans la Figure 6.1.

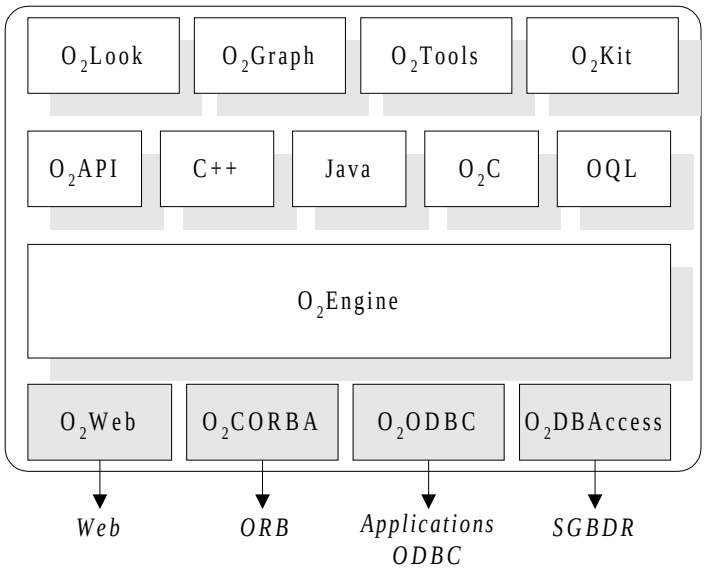

Figure 6.1 : Composants du système O_2

Le noyau du système (O_2Engine) fournit les mécanismes permettant d'accéder et de mettre à jour des données persistantes et de gérer les accès concurrents à ces données. Il assure également l'indépendance logique et physique des données. En tant que système orienté objet, O_2Engine gère les données sous la forme de schémas, de classes, d'objets et de méthodes. Il permet l'héritage, la surcharge, et l'édition dynamique des liens. Dans la version 5.0 d'O_2, O_2Engine supporte des mécanismes de versions qui facilitent l'utilisation du système pour des applications d'ingénierie (O_2Version), ainsi que la gestion de copies multiples d'objets (O_2Replication) [Ardent Software 98]. Il dispose également de moteurs de recherche (O_2Search) permettant d'accéder de façon dynamique à n'importe quel type de document (HTML, PDF…). Des outils de haut niveau permettent finalement de créer, modifier, optimiser et interroger des index spatiaux associés à des collections d'objets persistants (O_2Spatial).

Les applications faisant appel à O_2 ne dépendent pas d'un langage particulier. En effet, elles peuvent être codées en utilisant au choix :

- l'interface O_2API (*Application Programming Interface*) qui offre un accès direct à O_2Engine ;
- les langages C++ ou Java ;
- le langage O_2C, qui est un langage de quatrième génération basé sur C et spécialement conçu pour manipuler des objets. O_2C permet l'écriture de méthodes et de programmes d'application ;
- le langage de requêtes standard OQL (*Object Query Language*), qui étend SQL, entre autres en fournissant des opérateurs de manipulation d'objets complexes et en supportant l'activation de méthodes au sein d'une requête.

Le développement de ces applications est facilité par les outils suivants :

- O_2Look, un environnement de développement graphique très puissant incluant un générateur d'interfaces graphiques ;
- O_2Graph, un outil de conception de base de données orientée objet permettant de générer et de manipuler des graphes d'objets ;
- O_2Kit, un ensemble de classes offrant des services pour la manipulation de données couramment utilisées dans les applications (dates, texte, boîtes de dialogue…) ;
- O_2Tools, un environnement graphique de programmation d'applications.

Pour finir, O_2 est capable d'interagir avec d'autres systèmes grâce à un ensemble complet d'options de connectivité [Ardent Software 98] :

- O_2Web permet l'accès à n'importe quelle base de données O_2 via une connexion HTTP, en incluant des requêtes OQL dans des URL grâce à un script CGI ;
- O_2CORBA permet de communiquer et d'interagir avec des clients et des serveurs CORBA (*Common Object Request Broker Architecture*), qui permettent de manipuler, grâce à différents langages, des objets éventuellement répartis sur des machines variées au sein d'un environnement distribué hétérogène [Baker 97] ;
- O_2DBAccess autorise le développement d'applications O_2 qui accèdent à des bases de données relationnelles (connexion, envoi de requêtes SQL, traduction des résultats en objets O_2) ;
- O_2ODBC permet à des applications ODBC sur plate-forme PC/Windows d'utiliser des bases de données O_2.

1.3. Modes d'exécution

O_2 utilise deux modes pour l'exécution d'une application. Le premier mode, appelé mode *développement*, permet au programmeur de modifier de façon interactive le schéma de la base de données, en créant ou en supprimant des classes, des méthodes ou des objets nommés. Il autorise également la définition d'applications, de programmes et de méthodes, ainsi que leur exécution et leur test. Dans ce mode, O_2 est hautement interactif et peut faire complètement usage de fonctionnalités comme l'édition tardive des liens. En contrepartie de cette flexibilité, les temps d'exécution des méthodes sont allongés et des erreurs à l'exécution, dues à l'incohérence éventuelle de la base, sont possibles. Par exemple, une méthode peut recourir à une autre méthode alors que cette dernière a été supprimée. Ce mode est en fait utile en phase de développement des applications, lorsque les concepteurs ont plus besoin de dynamicité que de sécurité et de performance.

Lorsqu'une application est terminée et complètement déboguée, elle n'est plus manipulée par ses programmeurs, mais par les utilisateurs finaux. Dans ce contexte, la sécurité et la performance sont cruciales. Par contre, les évolutions de schéma ne sont plus nécessaires. C'est pourquoi O_2 fournit un second mode d'exécution, appelé mode *exécution* ou mode *production* [Adiba et Collet 93]. Dans ce mode, le schéma est figé, ce qui permet au système d'optimiser l'édition tardive des liens en remplaçant la résolution dynamique des noms par des appels de fonction, autant que possible. En contrepartie, les opérations sur le schéma (insertion ou suppression de classes ou de méthodes) ne sont plus permises. Le mode exécution est déclenché pour une application par une instruction de compilation qui produit un code exécutable optimisé. La cohérence entre l'application compilée et celle utilisée en mode développement (sous l'interpréteur de commandes) est assurée en interdisant les mises à jour des parties du schéma concernant l'application compilée.

En mode exécution, O_2 utilise uniquement les composants nécessaires à l'exécution d'une application C++, Java ou O_2C : le moteur O_2Engine et le générateur d'interfaces O_2Look. Par contre, en mode développement, tous les composants d'O_2 sont accessibles au programmeur, qui interagit directement avec l'environnement de programmation O_2API.

1.4. Noyau O_2Engine

1.4.1. Architecture générale

1.4.1.1. Architecture fonctionnelle

O_2Engine compte trois composants : un *gestionnaire de schéma*, un *gestionnaire d'objets* et un *gestionnaire de disque*, répartis en trois couches.

La couche supérieure, constituée du gestionnaire de schéma, administre la création, la suppression, la mise à jour et la recherche des classes, des méthodes et des objets nommés. Le gestionnaire de schéma gère également l'héritage et vérifie l'intégrité du schéma.

Le gestionnaire d'objets est la couche intermédiaire. Il gère des objets complexes et leur identité et fait la conversion entre les objets abstraits du modèle O_2 et leur représentation sur disque. Il réalise la persistance en utilisant les fonctions du gestionnaire de disque : ramasse-miettes, gestion des index, regroupement des objets.

Enfin, la couche basse est constituée du gestionnaire de disque WiSS (*Wisconsin Storage System*) [Chou et al. 85], qui gère la persistance, la concurrence d'accès aux données et la reprise après panne. WiSS manie des enregistrements « plats ». L'unité de persistance de base est la page disque. WiSS permet un contrôle total de la localisation physique des pages sur disque, car il n'utilise pas les primitives du système d'exploitation. Cela permet un regroupement efficace des objets.

Les différents composants d'O_2Engine, leur fonction et les données qu'ils manipulent sont récapitulés dans la Figure 6.2.

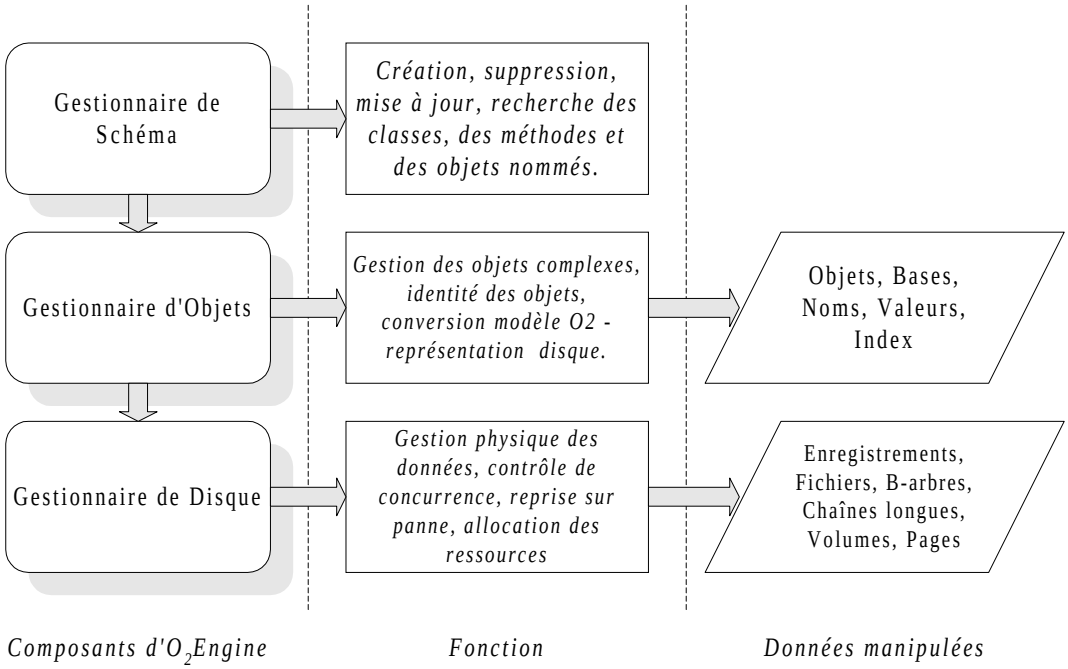

Figure 6.2 : Architecture fonctionnelle d'O_2Engine

1.4.1.2. Architecture matérielle

O_2 possède une architecture de type client-serveur, et plus précisément de type serveur de pages (cf. Chapitre 1, Section 1.2.2.1). Un processus serveur O_2 peut communiquer avec plusieurs processus clients O_2 répartis sur différents sites (Figure 6.3). L'unité d'échange entre processus serveur et processus client est une page physique de 4 Ko. La

communication se fait par une interface de type RPC (*Remote Procedure Call* : appel de procédure à distance).

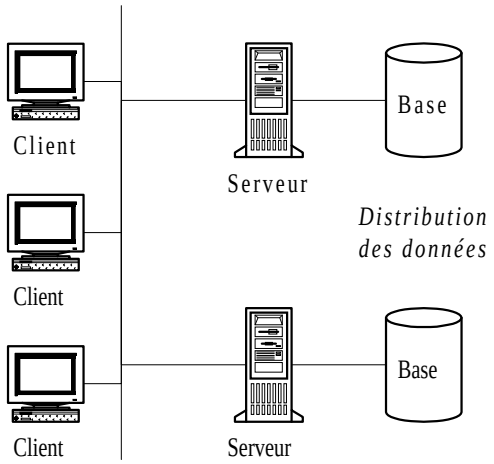

Figure 6.3 : Architecture client-serveur d'O_2

Le système O_2 existe donc en version *serveur* et en version *client*. Les deux versions ont pratiquement la même interface. La seule différence entre elles tient en fait en leur implémentation : la version client est mono-utilisateur et travaille en mémoire primaire, la version serveur est multi-utilisateurs et travaille sur disque. Les versions du gestionnaire de schéma et du gestionnaire d'objets diffèrent également d'une version à l'autre. Le gestionnaire de disque n'est présent que dans la version serveur.

1.4.2. Gestionnaire de schéma

Le gestionnaire de schéma utilise pleinement les fonctions du gestionnaire d'objets, c'est-à-dire qu'il existe une classe *Classe* et une classe *Méthode* dans le système (même si ceci est transparent pour les utilisateurs). Ainsi, les composants d'un schéma sont représentés et gérés de la même manière que des objets persistants. Cette approche permet à ces composants de bénéficier des mécanismes d'accès, de transactions et de reprise définis pour les objets. De plus, elle accélère les temps de développement grâce à l'uniformité du système.

Le gestionnaire de schéma permet la création, l'accès, la mise à jour et la suppression de classes, de méthodes et d'objets nommés. Il assure la sémantique de l'héritage et vérifie la cohérence du schéma. Les classes dans O_2 appartiennent à la fois à une hiérarchie d'héritage et à un graphe de composition. Cependant, l'ordre de création des classes n'est pas imposé, aussi certaines classes peuvent être momentanément spécifiées de façon incomplète à leur création (par exemple, en pointant vers une classe composante inexistante). Elles sont dénommées « shadow classes ». En revanche, la suppression des classes est ordonnée : il est interdit de supprimer une classe qui possède des instances, qui

est liée à une autre classe par des liens d'héritage ou de composition ou qui est référencée dans la signature d'une méthode ou la définition d'une valeur nommée. Le problème des classes qui se référencent mutuellement est réglé grâce à la possibilité d'effacer une liste de classes d'un coup.

Le gestionnaire de schéma fournit finalement des mécanismes permettant d'importer ou d'exporter des schémas et des bases de données.

1.4.3. Gestionnaire d'objets

Le gestionnaire d'objets est le cœur d'O_2Engine. Il gère les objets et leurs identificateurs et assure l'envoi de messages. Il administre aussi les valeurs complexes et leurs opérations structurelles. Il implante le modèle de persistance d'O_2, les index et les stratégies de regroupement d'objets. Il est également chargé de supprimer les objets non référencés.

Le gestionnaire d'objets est subdivisé en trois niveaux (Figure 6.4) :

- le plus bas niveau (le plus proche du gestionnaire de disque) est composé de cinq gestionnaires qui administrent respectivement les objets complexes, le passage de messages, les transactions, le regroupement et les index ;
- le niveau intermédiaire est un gestionnaire de cache d'objets ;
- la couche supérieure est dédiée à la communication. Les requêtes en provenance du client sont retransmises au module approprié.

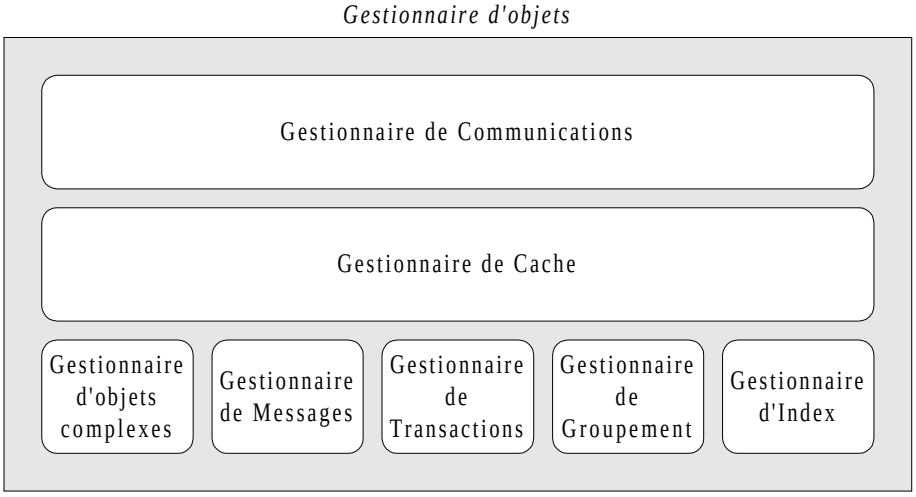

Figure 6.4 : Modules du gestionnaire d'objets

1.4.3.1. Gestionnaire d'objets complexes

Ce module remplit les tâches suivantes :

- création et suppression d'objets ;
- recherche d'objets nommés ;
- support de méthodes prédéfinies pour tous les objets (égalité entre objets, copie, affichage, édition…) ;
- support des objets structurés ensemble, liste et tuple.

Identification des objets

Les objets sont identifiés par un OID physique, qui évite la maintenance et l'accès à une table de descripteurs (cf. Chapitre 1, Section 1.4.3.2). Pratiquement, un objet est stocké sur disque dans un enregistrement WiSS. Son OID est donc l'identificateur de cet enregistrement (*Record Identifier*, RID). Cependant, lorsque les objets sont déplacés sur le disque, ils ne changent pas d'OID, ce qui nécessiterait la maintenance de liens vers tous les objets référençant l'objet déplacé, afin de mettre à jour leurs références. Un pointeur de suivi est simplement mis en place à l'ancienne localisation de l'objet déplacé. Les OID persistants sont affectés à la validation des transactions (*commit*). Auparavant, les objets nouvellement créés se voient attribuer des OID temporaires. Toutes les références à ces objets doivent être mises à jour lorsqu'ils reçoivent leur OID persistant.

Représentation des objets

Le modèle O_2 distingue les objets des valeurs. Au sein du gestionnaire d'objets, les valeurs structurées ont un OID et sont stockées comme des objets normaux, pour des raisons d'homogénéité. Le niveau d'indirection induit engendre un coût, mais il est compensé par le regroupement des valeurs avec l'objet auxquelles elles appartiennent.

1.4.3.2. Gestionnaire de messages

Ce module existe à la fois sur le client et le serveur. En mode développement, il supporte l'édition tardive des liens et l'application du code binaire sélectionné à l'objet cible. Le temps nécessaire à cette édition de lien est constant, car il n'y a pas de recherche des ancêtres d'une classe. En fait, les informations concernant les méthodes héritées sont dupliquées le long de la hiérarchie d'héritage.

En mode exécution, le passage des messages est implémenté par des appels de fonctions, autant que possible (c'est-à-dire lorsque le message n'est pas surchargé dans une sous-classe). Autrement, un appel de fonction adapté à chaque sous-classe est généré dans le code, de façon statique.

1.4.3.3. Gestionnaire de transactions

Le système O_2 est utilisé soit par des applications (en mode exécution), soit par des programmeurs (en mode développement). Les transactions sont cependant toujours gérées au niveau du gestionnaire d'objets. En mode développement, le gestionnaire de schéma fait appel aux primitives du gestionnaire d'objets.

La notion de transaction dans O_2 diffère de la notion classique ACID. En effet, la concurrence qui s'applique au schéma est différenciée de la concurrence qui s'applique aux objets. Ces deux modes peuvent être activés et désactivés indépendamment l'un de l'autre. Il est possible, par exemple, de gérer la concurrence au niveau des objets, mais pas au niveau du schéma, en mode exécution. Par ailleurs, la reprise sur panne peut être activée ou non, au choix. Ceci peut être utile lors de la mise au point d'une application, où la sécurité peut être sacrifiée au profit de la performance. Finalement, une transaction peut s'exécuter en mode *faute d'objet* ou en mode *résident*. Le premier de ces modes correspond à une utilisation normale : les objets sont chargés à la demande. Le second mode est utilisé pour les applications s'exécutant entièrement en mémoire. Toutes les données persistantes sont alors chargées en mémoire centrale au début de la transaction, dans le but d'améliorer la vitesse d'exécution. Le gestionnaire de schéma, par exemple, fonctionne en mode résident.

Chaque transaction O_2 correspond à une transaction WiSS. La concurrence est gérée par WiSS au niveau des pages (et donc sur le serveur). Sur les clients, aucun contrôle de concurrence n'est effectué, puisque les clients sont considérés comme des utilisateurs uniques. Au niveau du schéma, la granularité de contrôle de concurrence à la page s'avère insuffisante. En effet, l'accès par deux programmeurs à deux méthodes différentes de la même classe doit être autorisé. Cela est rendu possible par un contrôle de concurrence à l'objet.

1.4.3.4. Gestionnaire de groupement

Le placement des objets sur le disque dépend d'informations fournies par l'administrateur de la base de données à l'aide d'arbres de placement (cf. Chapitre 1, Section 3.2.3). L'algorithme de regroupement des objets est activé à la validation des transactions et agit sur les objets et les valeurs nouvellement créés. L'unité de stockage physique est la page disque.

1.4.3.5. Gestionnaire d'index

Ce module implémente des index qui tiennent compte des spécificités du modèle de données d'O_2 (hiérarchies d'héritage et de composition). Les index concernant chacune de ces hiérarchies se basent directement sur les index gérés par WiSS.

Un index donné s'applique à l'extension d'une classe, mais est également capable de « suivre » la relation d'héritage en s'appliquant à toutes les sous-classes de la classe initiale. Il est alors qualifié d'*index de hiérarchie de classe*. Une solution alternative est d'établir un index par sous-classe. On parle alors d'*index de classe simple.*

1.4.3.6. Gestionnaire de cache

Cette couche logicielle est chargée de traduire les OID persistants en adresses mémoire. Cela inclut la gestion des fautes d'objet pour les objets requis par les applications et qui ne se trouvent pas en mémoire centrale. Ce module administre également l'espace occupé par les objets en mémoire. La politique de gestion du cache est à deux niveaux : WiSS gère un cache de pages et le gestionnaire d'objets un cache d'objets. Les objets sont au format disque dans le cache de pages et au format mémoire dans le cache d'objets.

Sur le serveur, une faute d'objet provoque la lecture d'un enregistrement WiSS et son transfert depuis le cache de pages dans le cache d'objets. Même si un objet ne correspond grossièrement qu'à un seul enregistrement WiSS, à chaque faute d'objets, tous les enregistrements situés sur la même page que l'objet considéré sont chargés en mémoire. Cette stratégie se base sur le fait que les objets reliés entre eux sont groupés sur la même page ou sur des pages adjacentes et que le chargement d'une page entière accélère les traitements à venir.

1.4.3.7. Gestionnaire de communications

Les choix de conception concernant l'implémentation de ce module répondent à quatre critères.

- *Simplicité :* Les outils utilisés sont standards et éprouvés (réseau local Ethernet, protocole de transport TCP/IP).
- *Transparence :* L'utilisateur ne doit pas se soucier de la distribution des tâches entre le client et le serveur, ni de l'hétérogénéité éventuelle entre ces machines.
- *Performance :* Un des principaux goulots d'étranglement dans une application client-serveur est le déplacement des objets, qui a été optimisé dans O_2.
- *Fiabilité :* Un mécanisme de détection des erreurs prévient l'utilisation infinie des ressources pouvant résulter d'une terminaison anormale de processus.

Lorsqu'une application démarre sur un client, un *processus d'application miroir* est mis en œuvre sur le serveur afin d'interagir avec les couches basses du système. Sur le serveur, un processus démon fonctionnant en permanence est chargé d'accepter les requêtes de connexion émanant des clients et de créer les processus d'applications miroirs correspondants. Ce processus détecte également les problèmes pouvant survenir à la fois sur le client et au sein des processus miroirs.

Les objets sont transférés d'un site à l'autre de façon indépendante. Cependant, lorsque plusieurs objets doivent être transférés à la fois, ils sont regroupés en un seul message afin de minimiser les accès au réseau.

1.4.4. Gestionnaire de disque

Le système WiSS fournit des structures persistantes, des enregistrements, des B-arbres, des fonctions de hachage et des enregistrements longs. L'unité de stockage pour tous ces éléments est la page disque. Par exemple, un B-arbre est stocké dans un fichier dont les pages sont structurées en arbre. Une chaîne longue correspond à un enregistrement multi-pages dans un fichier.

Note : Un fichier au sens d'O_2 peut correspondre à un ou plusieurs fichiers du point de vue du système d'exploitation.

WiSS fournit également un modèle classique de transactions (à un niveau), gère les accès concurrents (grâce à un algorithme de verrouillage à deux phases) aux pages disque et met en œuvre des techniques de journaux « image-après » pour la reprise sur panne.

2. Gestionnaire d'objets persistants Texas

2.1. Présentation de Texas

Texas est un gestionnaire d'objets persistants pour C++ [Singhal et al. 92] [Wilson et Kakkad 92]. Il a été conçu et développé de 1992 à 1996 par Sheetal Kakkad, Vivek Singhal, Donovan Kolbly et Paul Wilson, à l'Université du Texas, Austin.

2.1.1. Objectifs

Texas a été conçu pour répondre aux objectifs suivants :

- implémenter la persistance dans un langage orienté-objet connu, C++ [Stroustrup 97] ;
- adopter une nouvelle approche évitant la gestion de tables d'objets, qui forment généralement un goulot d'étranglement dans les SGBDOO utilisant des OID logiques (cf. Chapitre 1, Section 1.4.3.2.) ;
- utiliser la technique de mémoire virtuelle pour gérer les objets sur disque et en mémoire de la même façon ;
- prendre en compte les bases de données d'une taille nécessitant des adressages supérieurs à 32 bits ;

- offrir une persistance orthogonale au type (toutes les classes, qu'elles soient standards ou définies par l'utilisateur, doivent pouvoir devenir persistantes) ;
- garantir une bonne portabilité, les bases de données devant être des supports d'information partageables entre plusieurs machines d'architectures différentes. En s'appuyant sur un langage et des systèmes d'exploitation standards et portables, le principe utilisé par Texas permet cette polyvalence ;
- permettre la manipulation d'objets persistants ou temporaires de façon uniforme pour le programmeur.

2.1.2. Caractéristiques

Texas est un outil se classant parmi les systèmes « à mémoire virtuelle ». À l'exécution, le format des données en mémoire est celui de C++. La seule différence entre un objet persistant et un objet temporaire est que le premier est créé sur une page persistante, conservée sur disque.

Le C++ accède donc de façon parfaitement uniforme aux objets persistants et aux objets temporaires. Une base de données est constituée de l'ensemble des pages persistantes stockées sur le disque, dans le format qu'elles ont en mémoire primaire.

Lorsque toutes les pages utilisées par une application sont chargées en mémoire virtuelle, la vitesse d'exécution est celle de C++. Texas utilise un *swizzling* orienté page : lorsqu'une page du fichier de base de données persistant est logée en mémoire, il convertit les références de tous les objets contenus dans cette page au format mémoire. L'accès aux objets se fait alors par des adresses physiques, évitant ainsi l'approche « table d'objets ». ObjectStore [Lamb et al. 91], qui repose sur le même principe, doit son succès commercial à des performances nettement supérieures à celles de ses concurrents.

Finalement, Texas se caractérise par sa facilité de mise en œuvre, la connaissance de C++ étant le seul pré-requis à son utilisation.

2.1.3. Inconvénients de cette nouvelle approche

L'approche employée par Texas présente des inconvénients qui sont bien connus.

- *Saturation de la mémoire physique :* Si le référentiel de Texas permet un adressage sur 32 bits, il n'empêche pas la mémoire physique disponible d'être rapidement saturée par des bases de données de volume important. Les performances sont alors très atténuées. En effet, du fait du manque de place en mémoire, certaines pages (déterminées par le système d'exploitation) sont placées (*swappées*) sur disque. L'accès à celles-ci coûte alors une entrée-sortie, et l'avantage donné par le principe des systèmes à mémoire virtuelle, c'est-à-dire la vitesse d'accès aux objets, est en grande partie perdu.

- *Politique de chargement d'objet :* Dans Texas, toutes les adresses d'une page de la base sont converties en adresses mémoires lorsque la page est chargée. Les pages correspondant à ces adresses doivent donc être réservées en mémoire. Ce procédé récursif est clairement exponentiel. Imaginons qu'une page possède comme données des tables d'identifiants d'objets, la réservation des pages peut porter sur le maximum d'adresses que contient une page. Un grand nombre de pages sont donc réservées pour rien et aucun mécanisme ne tempère cet encombrement progressif de la mémoire. D'autres politiques de *swizzling* [Richardson et Carey 89] [White et DeWitt 92] ne convertissent que les objets qui sont réellement utilisés. Mais la validité des pointeurs est alors contrôlée de façon logicielle, puisque des objets en mémoire peuvent contenir des attributs n'ayant pas encore été convertis.
- *Outil de persistance plus que SGBD :* Nos travaux concernent l'évaluation des performances des SGBDOO. Or, Texas est plutôt un « outil de persistance » pour C++. Il ne possède pas de langage de requête, par exemple. Sa structure ouverte et modulaire est cependant très intéressante dans un cadre de recherche, car elle rend Texas facilement modifiable. Il est relativement aisé d'y implanter des algorithmes de regroupement d'objets, par exemple.
- *OID physiques :* Le principal défaut dans l'approche de Texas et des systèmes similaires est l'utilisation d'OID physiques. En effet, les politiques d'optimisation de gestion d'objets, notamment le regroupement et la gestion de cache, nécessitent des déplacements d'objets. Lorsqu'un objet est déplacé, les objets qui le référencent possèdent alors des pointeurs invalides. Des techniques de pointeurs de suivi (*forward pointers*) existent [Vélez et al. 91], mais elles peuvent s'avérer fastidieuses en cas de déplacements successifs. C'est un problème important puisque les SGBDOO actuels étendent leurs fonctionnalités vers la distribution et la migration d'objets [Kim et al. 90] [Kim et al. 91].

2.2. Principes du gestionnaire d'objets

2.2.1. Structure d'une base de données

La structure d'une base de données de Texas est simple. C'est un fichier dont chaque enregistrement correspond à une page de la mémoire virtuelle. En entête de ce fichier se trouve un dictionnaire qui contient des couples *(nom, adresse)*. Ce sont les *objets nommés* de Texas, qui permettent d'entrer dans la base de données. Ils sont aussi appelés *objets racines*, car ils sont racines de sous-graphes d'éléments s'utilisant les uns les autres.

Pendant l'exécution, Texas matérialise la base de données par un objet « stockage persistant » (*Pstore*) qui recense toutes les pages mémoires en relation avec la base de données.

À l'ouverture de la base de données, la première page est chargée. Elle contient des pointeurs vers les tables de gestion des objets longs, la liste des emplacements libres et la zone de stockage des objets nommés (Figure 6.5).

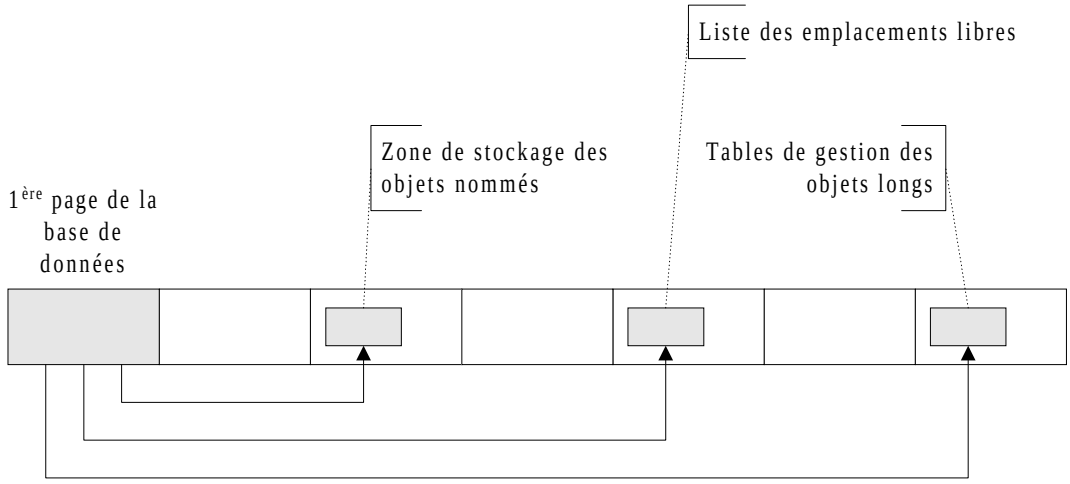

Figure 6.5 : Structure d'une base de données Texas

2.2.2. Conversion de pointeurs persistants

Une base de données Texas est stockée dans un seul fichier. Les références aux objets persistants sont stockées sous forme d'adresse disque. Ces adresses (identificateurs des objets) sont en fait un déplacement (*offset*) par rapport au début du fichier base de données.

La conversion format disque/format mémoire est souvent critique dans les SGBDOO. Dans Texas, elle est réalisée au moment du chargement de la page. Elle consiste à convertir les références aux objets persistants existant dans la page en adresses en mémoire virtuelle, en s'appuyant sur les mécanismes de protection d'accès interne au système d'exploitation.

Lors du chargement d'une page, toutes les références à des objets persistants sont converties, car elles représentent des objets potentiellement utilisables.

Lors du chargement d'une page contenant des objets nommés, celle-ci est simplement réservée en mémoire virtuelle et immédiatement protégée en lecture/écriture. Comme le premier accès à un espace persistant dans Texas se fait au travers d'un objet nommé, une faute de page est alors détectée sur la page en question et provoque son chargement en mémoire virtuelle.

Les pages de la base de données sont donc réservées et éventuellement chargées en mémoire au cours des navigations. L'ensemble des pages en mode « protégé en lecture » constitue ce que les auteurs de Texas appellent le « front de conversion ». Il représente l'ensemble des objets accessibles à un instant donné.

L'ensemble du mécanisme de conversion est présenté dans la Figure 6.6.

1. Chargement de la page *p* en mémoire virtuelle.

2. Recherche dans la page *p* de l'existence et de l'emplacement de toutes les références persistantes *r*.

3. Conversion de chacune des références *r* trouvées en adresses virtuelles. Deux cas de figure sont possibles :

- si une référence *r* désigne un objet déjà présent en mémoire à l'adresse *a*, alors *r* est convertie en l'adresse mémoire *a* ;
- dans le cas où *r* désigne un objet *o* présent sur une page disque *p*, non résidente en mémoire, alors :
  - . il y a réservation d'une nouvelle page en mémoire, susceptible de recevoir éventuellement la page *p* (cette page ne sera jamais chargée si l'objet *o* ou un autre objet de la même page n'est pas réellement utilisé),
  - . *r* est convertie en l'adresse mémoire correspondant au futur emplacement de l'objet *o* dans cette page réservée,
  - . la page réservée en mémoire virtuelle est protégée en lecture pour intercepter la première tentative d'utilisation et déclencher le chargement proprement dit de la page *p*.

4. À chaque interception de faute de page, retour à la phase 1.

Figure 6.6 : Mécanisme de conversion des pointeurs persistants dans Texas

Exemple : Soient cinq objets A, B, C, D et E. A est un objet nommé qui utilise les autres objets. Il est donc retrouvé à partir de son nom. L'accès au dictionnaire contenu dans la première page provoque une faute de page, puisqu'il contient des pointeurs sur des objets qui ne figurent pas en mémoire. Le mécanisme est amorcé (Figure 6.7). Les flèches représentent des liens entre objets, les rectangles des pages sur le disque et les ronds pleins des objets. Une relation R_i lie une page persistante et une page en mémoire. Une page grisée est protégée en lecture. Les autres pages sont utilisables en mémoire (données chargées depuis le disque et pointeurs valides).

L'objet A est déréférencé (on accède à A par l'intermédiaire de son pointeur). Ceci entraîne une faute de page. Texas charge les données de la page de A et convertit tous les pointeurs contenus dans la page de A (*swizzling*). Cette conversion implique la réservation des pages référencées par ces pointeurs, afin qu'une adresse en mémoire virtuelle soit associée à chacun d'entre eux (Figure 6.8).

Si l'objet B est utilisé, comme la page qui le contient est protégée en lecture, une faute de page est déclenchée. Les données sont chargées du disque vers la page par une fonction qui convertit tous les pointeurs qu'elle contient. Cela provoque le logement en mémoire de toutes les pages potentiellement référençables, donc de la page qui contient C (Figure 6.9).

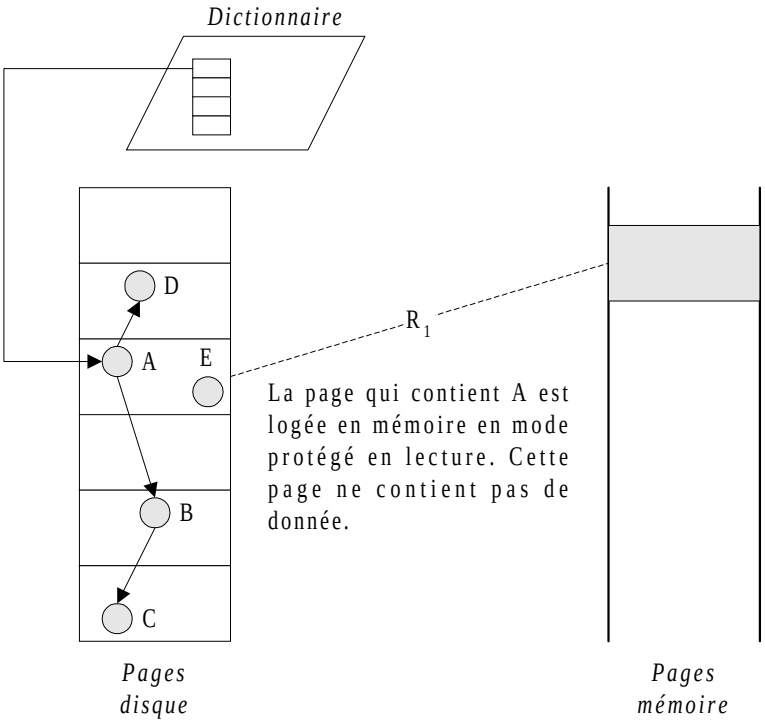

Figure 6.7 : État après accès au dictionnaire des objets nommés

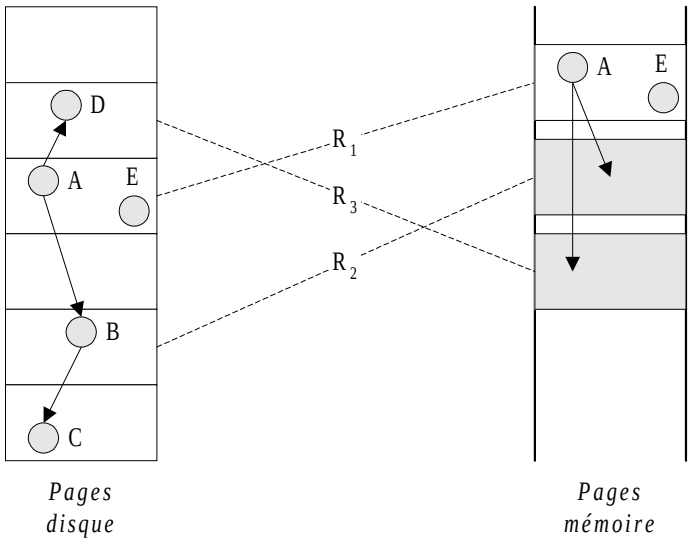

Figure 6.8 : Configuration après accès à l'objet A

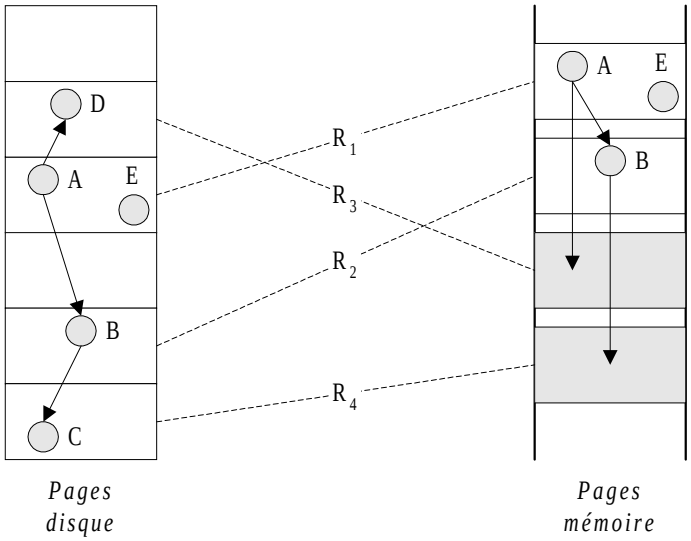

Figure 6.9 : Configuration après accès à l'objet B

2.2.3. Aspects transactionnels

Lorsqu'une page est amenée pour la première fois en mémoire et qu'elle contient des données valides (qu'elle est utilisable), elle est « protégée en écriture ». Toute tentative de modification provoque une faute de page. L'entité relation *(page persistante, page mémoire)* a alors son attribut « statut de la page » affecté de la valeur *dirty* (la valeur *clean* étant affectée par défaut aux pages mémoire non modifiées).

Quand une transaction est validée, Texas explore toutes les relations R_i en parcourant la table des conversions d'adresses (*swizzlermap*), qui contient la correspondance *(page persistante, page mémoire)*, puis lit le flag « statut de la page ». Toutes les pages marquées *dirty* retournent sur le disque. Comme elles sont en format mémoire au moment de la validation, ces pages subissent l'opération inverse du *swizzling* : une conversion d'adresses de la mémoire vers le disque (*unswizzling*).

La fin d'une transaction signifie implicitement le début de la suivante. Ainsi, les pages modifiées passent à nouveau par la phase de *swizzling* afin d'être utilisables par le programme, reçoivent la valeur *clean* comme statut (page non modifiée depuis la dernière transaction) et sont protégées en écriture. Ainsi, dès qu'une modification survient sur l'une des pages, le procédé de faute de page peut recommencer.

2.2.4. Remarques

La méthode de *swizzling* que nous venons d'exposer est très astucieuse. Elle appelle néanmoins quelques remarques sur ses principes de base :

- le *swizzling* par page provoque le chargement de nombreux objets inutilisés ;

- les performances sont très liées à la taille de la mémoire physique. Lorsque celle-ci est saturée, un phénomène de *swap* coûteux en entrées-sorties se produit ;
- la mémoire virtuelle se remplit continuellement : il n'existe pas de mécanisme pour éliminer les pages qui ne sont plus utilisées.

2.3. Utilisation de Texas

2.3.1. Mise au point d'une application

La construction d'un exécutable C++ faisant appel à Texas suit les étapes suivantes.

- Une première compilation s'applique au code du programme utilisateur dans le but d'obtenir des informations sur les structures de données potentiellement persistantes. Cette phase est réalisée au moyen du compilateur GNU, dont le mode de débogage permet d'obtenir les descripteurs de types d'un programme. Ces derniers sont placés dans un fichier ASCII.
- Une seconde passe consiste à compiler le fichier des descripteurs de types, qui peut être vu comme une fonction pouvant gérer en mémoire un dictionnaire de l'ensemble des structures de données utilisées.
- Enfin, une édition de lien prenant en compte le programme objet précédent de gestion des types, le programme objet de l'utilisateur, ainsi que la bibliothèque des modules de Texas, donne le code exécutable final. Ainsi, les descripteurs de types sont disponibles en mémoire pendant le déroulement d'une application.

Toutes ces opérations sont effectuées en une commande, au moyen d'un fichier *Makefile* qui contient la totalité des instructions menant au code exécutable.

2.3.2. Commandes pour la programmation

Texas permet à des applications C++ de créer des objets persistants, nommés ou non. L'interface proposée par la librairie est simple : pour pouvoir l'utiliser, il suffit d'inclure le fichier d'en-tête `texas.hh` dans le fichier source de l'application. Les instructions ajoutées à C++ pour la manipulation des bases de données Texas sont les suivantes.

- *Initialisation de Texas :* Il est nécessaire d'appeler la fonction `TEXAS_MAIN_INIT` dans le programme principal de l'application. Cet appel doit avoir lieu avant toute opération sur la base persistante. Il indique que l'application utilise les fonctionnalités de Texas à partir de ce point. En fin de programme, il faut indiquer la fin de l'utilisation de Texas par un appel à la fonction `TEXAS_MAIN_UNINIT`.

  Ces appels sont particulièrement nécessaires avec le système d'exploitation OS/2, contrairement à Unix. Cependant, pour des raisons de portabilité, il est

préférable d'utiliser ces deux commandes dans tout programme faisant appel à Texas.

- *Création et ouverture d'une base de données :*

```
Pstore *ps = open_pstore("nom_bd.pstore") ;
```

Cette fonction renvoie un pointeur sur l'objet désignant l'espace persistant (de classe *Pstore*).

- *Création d'un objet persistant :*

```
Personne *p = pnew (ps, Personne)("Dupont", 26) ;
```

Cette fonction renvoie un pointeur sur l'objet *p* de classe *Personne* et initialise ses attributs *nom* et *âge*.

- *Gestion des objets nommés :*

Une fois que la base de données persistante a été ouverte, le programme peut manipuler les objets nommés persistants. Ces derniers constituent le point d'entrée d'une base. Ils sont aussi appelés objets racines.

```
Is_root(ps, name) ;
```

Cette fonction renvoie un booléen qui répond à la question : l'objet racine *name* existe-t-il dans la base *ps* ?

```
Add_root(ps, type, name, ptr) ;
```

Cette fonction ajoute un objet nommé (de type *type* et de nom *name*, référencé par le pointeur mémoire *ptr)* dans la base *ps* et renvoie un pointeur (adresse persistante) sur celui-ci.

```
Remove_root(ps, type, name, ptr) ;
```

Cette fonction supprime de la base *ps* l'objet nommé de type *type*, identifié par le nom *name* et référencé par le pointeur *ptr*.

```
Set_root(ps, type, name, ptr) ;
```

L'objet de type *type* pointé par *ptr* devient un objet nommé *name* dans la base *ps*.

```
Get_root(ps, type, name) ;
```

Cette fonction renvoie un pointeur (qui peut être nul) sur l'objet de type *type* nommé *name* dans la base *ps*.

- *Instructions transactionnelles :*

```
Commit_transaction(ps) ;
```

Cette fonction valide la transaction courante et démarre implicitement une nouvelle transaction. Les modifications effectuées sont reportées dans la base *ps*.

```
Abort_transaction(ps) ;
```

Cette fonction annule la transaction en cours sur la base de données *ps* et en démarre implicitement une nouvelle.

- *Fermeture d'une base de données :*

  ```
  Close_pstore(ps) ;
  ```

  Cette fonction valide la transaction en cours et ferme la base de données *ps*.

  ```
  Close_pstore_without_commit(ps) ;
  ```

  Cette fonction ferme la base de données *ps* sans valider la transaction en cours.

2.4. Architecture fonctionnelle de Texas

Étant destiné à la recherche dans le domaine des bases de données, Texas a été conçu de façon à faciliter la compréhension de son architecture. Cette architecture est modulaire et se compose d'objets qui coopèrent pour fournir ensemble les fonctionnalités d'un gestionnaire d'objets persistants. L'implémentation, réalisée en C++, a rendu l'outil portable sur de nombreuses plates-formes (stations de travail Sun Sparc sous SunOS ou Solaris, stations de travail DEC sous Ultrix, PC sous OS/2 ou Linux).

2.4.1. Module de gestion des espaces persistants

La fonction de la classe *Pstore* (*Persistant Store*), dans ce module, est de garder une trace des bases de données en cours d'utilisation. Chaque base de données correspond à une instance de *Pstore* dès son ouverture.

La classe *Pstore* est une sous-classe de la classe *ObjStore* qui gère essentiellement l'allocation de l'espace de travail en mémoire (*Heap*) associé à la base concernée. La classe *Pstore* ajoute les autres structures de données nécessaires à la persistance :

- un pointeur vers une zone mémoire (*Heap*), sur laquelle sont applicables toutes les fonctions de gestion d'espace mémoire ;
- un pointeur vers la table persistante des objets nommés (*RootTable*). Ces derniers sont les points d'entrée de la base de données. Ils sont chargés dès l'ouverture de la base ;
- un pointeur vers le gestionnaire d'objets longs (*MultiPageMap*). Cette table donne la liste des pages occupées, car celles-ci ne sont pas contiguës sur disque. Cette information doit donc être persistante ;
- un pointeur sur le gestionnaire des descripteurs de type. Il donne la correspondance, pour une base de données, entre les noms de types utilisés et leur identificateur, qui est un numéro d'entrée dans la table des descripteurs de types ;

- un pointeur sur la table des descripteurs de type. Accédée via l'identificateur de type, elle contient la description de chaque type ;
- un pointeur vers le module de conversion (*Swizzler*) qui effectue pour les pages relatives à l'objet *Pstore* le *swizzling* par page ;
- un pointeur vers la table des conversions d'adresses *(page mémoire, page persistante)* (*VtoPMap*).

Les principaux services offerts sont :

- l'ouverture d'une base de données (création d'un objet *Pstore*), c'est-à-dire, l'initialisation du gestionnaire de descripteur de types, l'initialisation du module de conversion d'adresses et la restauration à partir d'un journal (*log*), s'il existe ;
- la validation d'une transaction ;
- la fermeture d'une base de données (destruction de l'objet *Pstore* correspondant) ;
- l'extension du fichier persistant constituant une base de données.

2.4.2. Module de gestion des objets nommés

La table des objets nommés (*RootTable*) est chargée de mémoriser la localisation des objets persistants (adresse disque ou adresse mémoire virtuelle, suivant que la table est présente en mémoire ou sur disque). Elle est implémentée comme une table de hachage et est toujours le premier objet de la première page de la base de données. Elle possède aussi une copie des listes d'emplacements libres et de la table des objets longs. L'ensemble de ces données est sauvegardé au moment de la validation des transactions et chargé en mémoire dès l'ouverture de la base.

2.4.3. Module d'allocation de mémoire

Dans ce module, la classe *Heap* surcharge toutes les routines d'allocation mémoire standards telles `malloc`, `realloc`, `free`… Chaque base de données accédée est associée à un espace de travail persistant en mémoire. Les objets temporaires sont, eux, dans un espace de travail temporaire global.

L'algorithme d'allocation mémoire est simple. Chaque objet alloué en mémoire possède un en-tête de 8 octets. La mémoire est allouée par morceaux dont la taille en octets doit être une puissance de 2. Ainsi, le nombre d'octets requis devra être arrondi à la puissance de 2 supérieure, en comptant la taille de l'en-tête.

L'en-tête de l'objet contient plusieurs informations :

- un flag qui indique si l'emplacement est libre ou s'il comporte un objet,
- la taille de l'objet,

- le nombre d'octets réellement utilisés,
- un pointeur sur le prochain espace libre, si l'objet est libre.

Un tableau des espaces libres est maintenu (un pour chaque catégorie de taille). Initialement, quand la liste des espaces libres, pour une catégorie de taille donnée, est vide, le *Heap* appelle le module de gestion de la mémoire pour allouer une page depuis la mémoire virtuelle. La liste libre doit être persistante. Pour cela, la table des objets nommés contient une copie de la liste libre.

2.4.4. Module de gestion de mémoire

Le gestionnaire de mémoire répond aux demandes d'allocation de pages. La procédure `malloc` appelle la fonction `alloc-page` avec le nombre de pages comme argument, pour étendre l'espace mémoire virtuel associé au programme.

L'allocation se fait par page. Le gestionnaire de mémoire connaît la catégorie de taille de l'objet pour lequel une page doit être allouée. Si les objets sont plus petits que la taille de la page (c'est-à-dire que plus d'un objet peut tenir dans une page), le gestionnaire de mémoire met l'objet dans un morceau de taille appropriée, le lie avec une liste libre et retourne la tête de la liste.

2.4.5. Module de conversion d'adresses

Une table de conversion est associée à chaque objet racine, pour chaque base de données en cours d'utilisation. Sa fonction première est de maintenir une correspondance entre adresses en mémoire et adresses sur disque. Cette correspondance est utilisée au cours des conversions. Ceci est implémenté par deux tables de hachage *VtoPMap* et *PtoVMap* (adresses virtuelles vers adresses persistantes et inversement).

Cette table gère aussi l'état de chaque page mémoire. Cet état peut être *réservé* (la page est réservée et protégée en lecture/écriture, sans contenir aucune donnée), *clean* (la page a été utilisée, et son appel a provoqué le chargement de données) ou *dirty* (la page a été modifiée par une application). Une liste des pages marquées à *dirty* est maintenue et utilisée lors de la procédure de validation, afin de mettre à jour sur disque les pages modifiées par une transaction.

Chaque objet *Pstore* est également associé à un module de conversion (*Swizzler*) qui réalise effectivement les conversions et met à jour les correspondances dans la table de conversion. Il utilise les informations contenues dans les descripteurs de types pour localiser les pointeurs persistants à l'exécution. Tous les modules de conversion actifs sont enregistrés dans une structure statique globale, utilisée pour déterminer à quelle base une page mémoire donnée appartient.

Ce module gère également les fautes de page. Une faute de page peut provenir d'un accès à une page non encore utilisée (protégée en lecture/écriture et ne contenant pas encore de données) ou d'une tentative de modification sur une page protégée en écriture (non encore modifiée). Dans le premier cas, le module de conversion doit amener la page correspondante du disque en mémoire et faire toutes les opérations de conversion nécessaires dans cette page. Dans le deuxième cas, la page est marquée à *dirty* et ajoutée à la liste des pages modifiées. La protection en écriture de la page est levée.

2.4.6. Le module de gestion des objets longs

Les objets dont la taille est supérieure à la taille d'une page mémoire doivent être logés au sein d'un ensemble de pages contiguës en mémoire virtuelle. Il faut donc mémoriser le début et la fin de chaque objet.

La table des objets longs est implémentée sous forme d'une table de hachage persistante. Une copie en est sauvegardée dans la table des objets nommés, ce qui permet, lorsqu'une base est ouverte, de la restaurer.

2.4.7. Le module de recouvrement

Un journal est utilisé et permet de mettre en place deux phases de recouvrement après panne :

- écriture de toutes les pages modifiées (*dirty*) dans le journal et pré-validation ;
- mise à jour de la base de données.

Si un problème apparaît durant la première phase, la base de données n'étant pas encore modifiée, elle est toujours cohérente. Si un problème apparaît dans la deuxième phase, alors les mises à jour sont réexécutées depuis le journal autant de fois que nécessaire.

Le journal est donc un ensemble d'enregistrements qui contiennent chacun des données et l'adresse disque où les données sont écrites. Quand toutes les pages modifiées ont été écrites dans le journal, un enregistrement de pré-validation est écrit et le journal est déclaré complet. Pour ouvrir une base de données, le journal qui contient les enregistrements de pré-validation est utilisé.

Cette politique de recouvrement est sûre, mais peu efficace, car elle nécessite deux écritures pour chaque page modifiée afin d'assurer la cohérence de la base de données.

3. Technique de groupement d'objets DSTC

La stratégie de regroupement DSTC (*Dynamic, Statistical and Tunable Clustering*) a été proposée par Frédérique Bullat, de l'Université Blaise Pascal, Clermont-Ferrand II, dans le cadre de sa Thèse de Doctorat d'Université [Bullat 96]. Nous avons déjà brièvement présenté ses caractéristiques dans le Chapitre 1, Section 3.3.1.3. Nous détaillons ici plus avant les principes qui régissent cette stratégie de regroupement dynamique, ainsi que les structures de données utilisées en pratique pour sa mise en œuvre.

3.1. Objectifs

L'étude de différentes stratégies et propositions de regroupement dans [Bullat 95] a permis de définir précisément les objectifs de la méthode DSTC :

- remédier au placement distant d'un ensemble d'objets qui possèdent un lien fort à l'utilisation et réduire les entrées-sorties superflues ;
- obtenir un gain supérieur au coût de réorganisation, pour qu'il n'y ait pas de dégradation des performances. Plus précisément, soient T_r le temps de réorganisation, T_{ap} le temps d'utilisation de la base après regroupement et T_{av} le temps d'utilisation de la base avant regroupement. On doit avoir :

$$T_r + T_{ap} < T_{av} \ ;$$

- garantir une facilité d'adaptation à différents contextes d'exploitation ;
- coupler regroupement d'objets et gestion de cache mémoire.

La pertinence des spécifications de regroupement constitue l'élément majeur d'une stratégie de regroupement. Les objectifs suivants ont donc été fixés :

- spécification de regroupements basée sur les observations de l'utilisation effective des objets ;
- prise en compte de tous les types de relations inter-objets sans les différencier et donc, pas de généralisation à des liens entre classes ;
- résolution déterministe des conflits dus au partage d'objets.

En dehors de la qualité des spécifications considérées, il faut établir et maintenir un bon placement des objets. Ceci inclut les objectifs suivants :

- éviter les manipulations de l'utilisateur pour effectuer le regroupement ;
- adopter une unité de stockage supérieure à la page ;
- réduire les coûts de stockage (en volume) ;

- parvenir à la dynamicité en réorganisant la base aux moments les plus propices (lorsque les gains engendrés par le regroupement sont supérieurs aux coûts de réorganisation de la base et que le nombre de transactions est faible) ;
- ne déplacer un objet qu'après examen de l'ensemble des objets liés.

3.2. Principes

3.2.1. Philosophie générale

DSTC a pour objectif de regrouper sur le disque les objets souvent utilisés ensemble à des instants proches dans le temps. Seul le placement initial est influencé par le schéma conceptuel de la base. Un objet n'est déplacé que lorsqu'il possède une forte dépendance avec un objet éloigné physiquement.

3.2.2. Observation de la base

[Bullat 96] part des constats suivants pour préconiser un groupement basé sur l'observation de l'usage des objets.

- Comme plusieurs utilisateurs et plusieurs applications peuvent travailler en même temps sur une même base de données, il est difficile pour l'administrateur de déterminer un placement représentatif. Il s'agit pourtant de l'approche privilégiée par diverses stratégies, comme ORION [Kim et al. 90], O_2 [Benzaken et Delobel 90] ou ENCORE [Hornick et Zdonik 87].
- Un placement des objets par les programmeurs d'applications est également problématique, car plusieurs applications peuvent demander des placements non compatibles.
- De même, il est hasardeux de s'appuyer sur le schéma, car il est impossible de prédire le comportement de la base face à l'ensemble des utilisateurs et à l'ensemble des applications.

3.2.3. Spécifications par les statistiques

L'utilisation des statistiques a un effet modérateur. Elle permet de mettre en évidence l'utilisation moyenne des objets. Elle possède aussi un effet sélectif, puisque des valeurs filtres permettent d'éviter des déplacements trop fréquents qui seraient pénalisants pour les performances.

Une variation sensible et durable de l'utilisation d'un objet aura pour effet de modifier les statistiques et donc de provoquer le déplacement d'un ou plusieurs objets.

3.2.4. Prise en compte de tous types de liens

Il est nécessaire de tenir compte de tous les types de liens entre objets. Le constat physique doit donc être indépendant du schéma de la base ou de consignes fournies par les utilisateurs.

Il existe différents types de liens :

- des liens *structurels* (déduits du schéma),
- des liens *logiques* (déduits des applications),
- des liens *physiques* (déduits de la fragmentation physique d'objets). Seule la référence à l'identificateur est considérée comme un lien physique.

En utilisant le déréférencement d'un objet à partir d'un autre, tous ces types de liens peuvent être captés.

3.3. Stratégie de regroupement

Pour regrouper les objets en fonction de l'utilisation effective de la base, des statistiques sont employées. Si, dans un passé récent, des applications ont souvent utilisé des objets ensemble, il est possible de prédire qu'il en sera de même dans un futur proche. Les statistiques fournissent donc des fréquences d'accès qui décrivent le comportement futur de la base de données.

La collecte et la sélection de l'information consistent à capter les données sur l'utilisation des objets en respectant certaines contraintes :

- minimiser la quantité d'informations ;
- maximiser la pertinence des informations ;
- réduire le coût de stockage persistant ;
- minimiser les perturbations éventuelles sur les transactions en cours.

Cet objectif est atteint en étageant les informations à différents niveaux et en utilisant des filtres progressifs sur les statistiques maintenues en mémoire primaire. Ne sont stockées de façon persistante que les informations significatives.

La collecte des informations qui servent aux spécifications de DSTC repose essentiellement sur trois phases : observation, sélection, consolidation. Un regroupement des objets est ensuite possible.

3.3.1. Phase d'observation

La phase d'observation consiste en une collecte primaire d'informations sur l'utilisation d'une base de données par les transactions. Elle se déroule en permanence, tant qu'il y a des applications actives. Tous les types de liens (instances d'objets ou fragments physiques) conduisant les objets à être utilisés ensemble s'expriment par un parcours de pointeurs physiques dans la base. Donc, chaque déréférencement de pointeur persistant est considéré comme un lien orienté entre l'objet de départ et l'objet référencé.

3.3.1.1. Séquence d'appel

Une séquence d'appel est le déroulement dans le temps de tous les appels d'objets durant une période donnée (comptée en nombre d'objets utilisés). Lorsqu'est constaté le fait que l'objet O_i appelle l'objet O_j, cela signifie « qu'après avoir utilisé l'objet O_i, il est très probable d'utiliser l'objet O_j ».

La Figure 6.10 donne un exemple de graphe d'accès aux objets, produit à partir d'une séquence d'accès donnée au cours d'une transaction. Le poids des arcs $O_i \rightarrow O_j$ représente le nombre de fois où l'objet O_j a été référencé après l'objet O_i. Les compteurs en italiques indiquent le nombre d'accès à chaque objet. Dans cet exemple, les objets O_2, O_3 et O_4 sont accédés à partir de l'objet O_1. Au cours de la transaction, l'objet O_1 a été référencé 9 fois et le lien $O_1 \rightarrow O_2$ a été utilisé 3 fois.

Séquence d'accès :

O_1	O_2	O_6	
O_1	O_4	O_7	
O_1	O_2	O_6	
O_1	O_3		
O_1	O_4	O_3	
O_5	O_7	O_1	
O_1	O_2	O_6	
O_1	O_3		
O_1	O_2	O_6	O_2

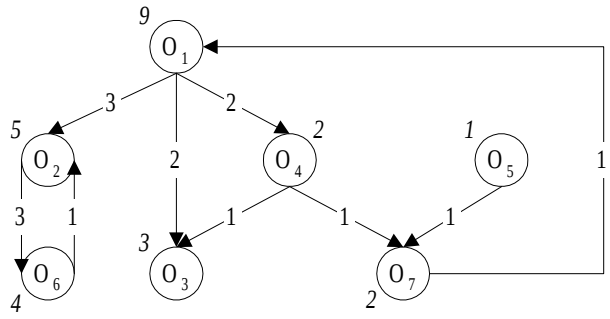

Figure 6.10 : Graphe d'accès associé à une séquence d'accès

Le graphe d'accès est relatif à une transaction. Il est produit sur la base d'une séquence d'accès donnée. Un graphe d'accès global, relatif à une période P_i, rassemble les informations des graphes d'accès relatifs aux transactions individuelles. Il est représenté par (Figure 6.11) :

- une matrice d'observation MO,
- un vecteur de fréquences d'accès aux objets VO (vecteur d'observation).

MO est une représentation matricielle illustrant les différents liens orientés observés durant une période. Une matrice d'observation est liée à une base de données. Donc, la valeur de MO(i, j) est égale au nombre de fois où le lien $O_i \rightarrow O_j$ est utilisé durant la période.

Le vecteur d'observation VO représente la fréquence d'accès aux objets durant la période. Donc, la valeur de VO(i) est égale au nombre de fois où O_i a été accédé durant la période.

MO	O_1	O_2	O_3	O_4	O_5	O_6	O_7
O_1		3	2	2			
O_2						3	
O_3							
O_4			1				1
O_5							1
O_6		1					
O_7	1						

VO
9
5
3
2
1
4
2

Figure 6.11 : Matrice et vecteur d'observation DSTC

La matrice et le vecteur d'observation sont temporaires. Si *n* est le nombre d'objets accédés, la matrice sera donc de taille n^2. Cependant, MO est une matrice creuse et peut donc être implémentée de façon efficace, par exemple grâce à une table de hachage.

Soit f_{ij} la fonction qui représente la fréquence d'accès à l'objet O_j quand l'objet courant est O_i (la « probabilité d'accéder à O_j quand on utilise O_i »).

$$f_{ij} = \frac{\text{MO(i, j)}}{\text{VO(i)}} \quad \text{avec VO(i)} \neq 0$$

3.3.1.2. Déroulement de la phase

Durant une période d'observation :

- la séquence d'accès de chaque transaction est mémorisée ;
- en fin de transaction (*commit*), un processus asynchrone analyse la séquence obtenue pour une transaction. Les résultats de cette analyse sont :
  - le nombre de fois où un lien de référence a été parcouru pendant la transaction, entre chaque paire d'objets liés. Ces informations sont ajoutées à celles contenues dans la matrice d'observation courante ;
  - le nombre d'accès à chaque objet utilisé pendant la transaction courante. Le vecteur d'observation est donc mis à jour.

La durée d'une période d'observation varie en fonction du taux d'utilisation de la base de données. La fin d'une période survient lorsque la taille de la matrice d'observation MO

est supérieure à un nombre d'objets accédés *n*. En fin de période, MO et VO sont représentatifs de l'activité de la base de données pour cette période.

3.3.2. Phase de sélection

La phase de sélection intervient en fin de période et consiste en une opération de filtrage et de calcul de fréquence d'utilisation conjointe.

3.3.2.1. Facteur de liaison élémentaire

En fin de période, un traitement est appliqué à la matrice d'observation MO afin d'obtenir la matrice de consolidation MC, qui est persistante. À l'aide de la matrice d'observation et du vecteur d'observation, les facteurs de liaisons élémentaires fe_{ij} sont calculés. Ces facteurs représentent la force d'attraction entre objets : il est souhaitable de regrouper l'objet O_i avec l'objet O_j si O_i est souvent accédé à partir de O_j ou inversement.

La phase d'observation fournit des liens orientés, donc le nombre d'accès ($O_i \rightarrow O_j$) est différent du nombre d'accès ($O_j \rightarrow O_i$). Alors :

$$fe_{ij} = fe_{ji} = f_{ij} + f_{ji} = \frac{\mathrm{MO(i,\ j)}}{\mathrm{VO(i)}} + \frac{\mathrm{MO(j,\ i)}}{\mathrm{VO(j)}}.$$

fe_{ij} est le cumul des fréquences d'accès relatives, c'est-à-dire les fréquences d'accès de O_i à partir de O_j et les fréquences d'accès de O_j à partir de O_i. L'addition exprime le « ou » logique (non exclusif) signifiant : « O_i utilise O_j ou O_j utilise O_i ».

fe_{ij} appartient à l'intervalle [0, 2]. Dans les faits, ce facteur de liaison élémentaire est rarement égal à 2, car cela signifierait que les deux objets concernés ne font que s'utiliser mutuellement.

3.3.2.2. Matrice élémentaire

Les facteurs de liaisons élémentaires calculés et jugés significatifs sont placés, en fin de période, dans une matrice élémentaire ME. Cette matrice est creuse et triangulaire (puisque $fe_{ij} = fe_{ji}$). Son contenu n'est pas persistant, car ME constitue en fait une structure temporaire de travail avant consolidation de la matrice d'observation.

Le graphe des liens élémentaires correspondant à ME, qui est non orienté et dont les arêtes sont pondérées par *fe*, est obtenu après les opérations de filtrage suivantes :

- sélection sur le nombre d'accès aux objets liés effectués durant la période. Cette valeur n'est pas significative s'il y a un nombre trop faible d'utilisations des objets (par exemple, si l'objet O_i n'est utilisé qu'une seule fois, puis utilise O_j,

alors $fe_{ij} = {}^{1}\!/\!_{1} = 1$. La valeur est élevée, mais calculée sur un nombre trop faible d'utilisations, elle n'a aucune signification — cf. fe_{57} sur la Figure 6.12) ;

- sélection sur la valeur de *fe* permettant d'éliminer tous les liens élémentaires restés faibles sur la période considérée. Cela permet de ne regrouper à terme que les objets fortement liés.

La Figure 6.13 et la Figure 6.14 représentent respectivement la matrice élémentaire ME obtenue après filtrage et le graphe des liens élémentaires qui lui est associé.

$$fe_{12} = fe_{21} = \left({}^{3}\!/\!_{9}\right) + \left({}^{0}\!/\!_{5}\right) = 0{,}33 \qquad fe_{13} = fe_{31} = \left({}^{2}\!/\!_{9}\right) + \left({}^{0}\!/\!_{3}\right) = 0{,}22$$

$$fe_{14} = fe_{41} = \left({}^{2}\!/\!_{9}\right) + \left({}^{0}\!/\!_{2}\right) = 0{,}22 \qquad fe_{26} = fe_{62} = \left({}^{3}\!/\!_{5}\right) + \left({}^{1}\!/\!_{4}\right) = 0{,}85$$

$$fe_{34} = fe_{43} = \left({}^{0}\!/\!_{2}\right) + \left({}^{1}\!/\!_{2}\right) = 0{,}50 \qquad fe_{47} = fe_{74} = \left({}^{1}\!/\!_{2}\right) + \left({}^{0}\!/\!_{2}\right) = 0{,}50$$

$$fe_{57} = fe_{75} = \left({}^{1}\!/\!_{1}\right) + \left({}^{0}\!/\!_{2}\right) = 1{,}00 \qquad fe_{71} = fe_{17} = \left({}^{1}\!/\!_{2}\right) + \left({}^{0}\!/\!_{9}\right) = 0{,}50$$

Les objets O_2 et O_6 sont les plus fortement liés.

Figure 6.12 : Facteurs de liaison élémentaires

ME	O_1	O_2	O_3	O_4	O_5	O_6	O_7
O_1		0,33	0,22	0,22			0,50
O_2						0,85	
O_3				0,50			
O_4							0,50
O_5							
O_6							
O_7							

Figure 6.13 : Matrice élémentaire

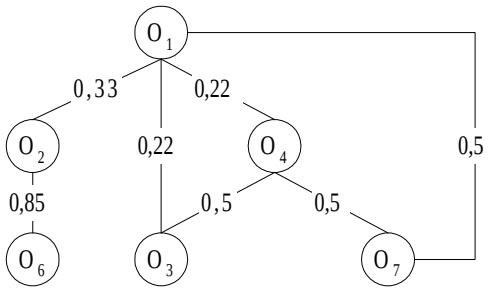

Figure 6.14 : Graphe des liens élémentaires

3.3.2.3. Déroulement de la phase

En fin de période, le contenu des structures MO et VO est traité afin de détecter les liens inter-objets susceptibles d'enrichir la matrice temporaire ME, avant de venir consolider le contenu de la matrice MC.

Deux filtres, Tf_e[*] et Tf_a[**] (Tf_e pour les facteurs élémentaires et Tf_a pour les fréquences d'accès aux objets), permettent de diminuer la quantité de statistiques à stocker et d'augmenter la pertinence des informations. Le fait que le graphe soit non orienté divise par ailleurs par deux le volume d'informations à stocker.

Lorsque la phase de sélection est terminée, MO et VO sont remis à zéro afin d'initialiser le démarrage d'une nouvelle période d'observation. La phase de consolidation débute en parallèle.

L'algorithme qui régit la phase de sélection est présenté en Figure 6.15.

```
// Traitement de tous les liens orientés observés
Pour i = 2, n faire
     Pour j = 1, n-1 faire
          Si MO(i,j) ≠ 0 ou MO(j,i) ≠ 0 alors
               // Si Oi ou Oj a été suffisamment utilisé au cours de la période
               // d'observation, un facteur de liaison leur est associé.
               Si VO(i) ≥ Tfa ou VO(j) ≥ Tfa alors
                    feij = MO(i,j)/VO(i) + MO(j,i)/VO(j)
                    // Les liens élémentaires trop faibles ne sont pas
                    // conservés.
                    Si feij < Tfe alors
                         feij = 0
                    Fin si
                    ME(i,j) = feij
               Fin si
          Fin si
     Fin pour
Fin pour
```

Figure 6.15 : Phase de sélection

3.3.3. Phase de consolidation

Il s'agit maintenant de pondérer les données contenues dans la matrice consolidée persistante MC avec ceux de la matrice temporaire ME, pour une période P_i donnée. L'objectif est d'accorder plus de crédit à des observations constatées sur plusieurs périodes.

[*] Tf_e : *Threshold value for Elementary linking Factors.*
[**] Tf_a : *Threshold value for Access Frequency.*

3.3.3.1. Facteur de liaison consolidé

Le facteur de liaison élémentaire *fe* est limité à une période. Les facteurs de liaison consolidés *fc* correspondent à des indicateurs représentatifs de l'utilisation conjointe des objets. Ils servent de base aux décisions de regroupement des objets.

$$fc_{ij}^{P_i} = fc_{ji}^{P_i} = w.fc_{ij}^{P_{i-1}} + (1-w).fe_{ij}$$

w (*weight*) est un coefficient de pondération introduit pour respecter les différents niveaux de pertinence. En effet, les facteurs relatifs à une période P_i sont moins pertinents que les facteurs consolidés sur plusieurs périodes. *w* doit donc être proche de 1. Par exemple, un coefficient *w* de 0,8 accordera un poids de 80 % aux anciennes informations consolidées et un poids de 20 % seulement aux nouveaux liens constatés. Cela introduit un effet mémoire et donc un contrôle de la stabilité de la stratégie.

3.3.3.2. Effet mémoire

Les facteurs de liaisons consolidés sont mis à jour à chaque changement de période. L'effet mémoire dépend donc de la valeur du coefficient de pondération *w*. L'objectif est de déclencher un regroupement des objets O_i et O_j lorsque fc_{ij} est supérieur à une valeur seuil donnée.

w freine l'augmentation ou la diminution d'un facteur de liaison consolidé. Ce dernier ne possède une valeur significative que lorsqu'il est constaté plusieurs fois. De la même façon, un lien qui apparaît dans une période de façon plus faible que dans la période précédente verra son facteur de liaison diminuer de façon sensible seulement si la diminution se confirme dans le temps. Si un lien n'est pas constaté depuis plusieurs périodes, son facteur de liaison est remis à zéro, ce qui permet d'épurer la matrice consolidée MC.

La Figure 6.16 illustre la façon dont les facteurs de liaisons consolidés sont mis à jour à partir d'un graphe des liens consolidés existant (Figure 6.17) et de la matrice élémentaire ME de la Figure 6.13. Différents cas de figure sont présentés :

- le lien élémentaire (O_4, O_7), constaté de façon forte (0,5), est relativisé (0,15) car il n'avait jamais été détecté auparavant ;
- le lien (O_2, O_6), faible jusqu'à présent (0,2), a été fortement présent au cours de la période écoulée (0,85). L'augmentation de la force attractive est relativisée par la pondération exercée (0,395) ;
- le lien (O_1, O_2), fort (0,6), a été moins constaté au cours de la période écoulée (0,33). Là encore, la diminution de la force attractive est relativisée par la pondération exercée (0,519) ;

- les liens (O_1, O_3), (O_1, O_4), (O_1, O_7) et (O_3, O_4) ont conservé leur utilisation habituelle, ils gardent donc leur valeur ;
- le lien (O_4, O_6), qui n'a pas été constaté au cours de cette période, conserve également sa valeur.

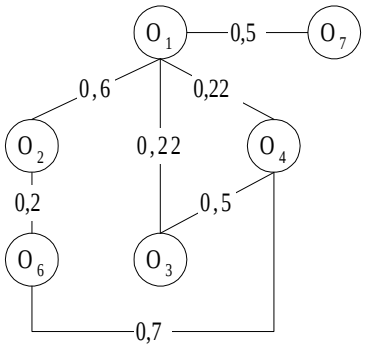

Figure 6.16 : Graphe des liens consolidés existants

$$w = 0{,}7$$

$$f_{47} = f_{74} = (0{,}7 \times 0) + (0{,}3 \times 0{,}5) = 0{,}15 \qquad f_{26} = f_{62} = (0{,}7 \times 0{,}2) + (0{,}3 \times 0{,}85) = 0{,}395$$
$$f_{12} = f_{21} = (0{,}7 \times 0{,}6) + (0{,}3 \times 0{,}33) = 0{,}519 \qquad f_{13} = f_{31} = (0{,}7 \times 0{,}22) + (0{,}3 \times 0{,}22) = 0{,}22$$
$$f_{14} = f_{41} = (0{,}7 \times 0{,}22) + (0{,}3 \times 0{,}22) = 0{,}22 \qquad f_{17} = f_{71} = (0{,}7 \times 0{,}5) + (0{,}3 \times 0{,}5) = 0{,}5$$
$$f_{34} = f_{43} = (0{,}7 \times 0{,}5) + (0{,}3 \times 0{,}5) = 0{,}5$$

Figure 6.17 : Mise à jour des facteurs de liaison consolidés

Le facteur *w* est très important. Il est déterminé suivant le contexte d'exécution du SGBD. Plus la configuration matérielle permet une place mémoire importante pour l'activité d'observation, plus les périodes d'observation P_i peuvent être étendues dans le temps (cela dépend de la taille *n*, en nombre d'objets, qu'il est possible d'attribuer à la matrice d'observation MO en mémoire).

3.3.3.3. Matrice consolidée

Les facteurs de consolidation *fc* sont stockés dans la matrice MC, qui est persistante, creuse et triangulaire. Chaque cellule MC(i, j) de MC contient :

- le facteur de consolidation fc_{ij} ;
- la date de dernière mise à jour de fc_{ij} ($date_{ij}$), qui reçoit l'identifiant de la période P_i à laquelle le lien (O_i, O_j) a été consolidé pour la dernière fois. Cette date permet d'éliminer les fc_{ij} obsolètes, c'est-à-dire pour lesquels le lien (O_i, O_j) n'a pas été observé au cours des périodes précédentes ;

- un flag ($group_{ij}$) indiquant si O_i et O_j ont déjà fait l'objet d'un regroupement sur le disque.

fc_{ij} est remis à zéro lors de la phase de consolidation quand :

$$P_{now} - date_{ij} > n_p \text{ (modulo p)}.$$

- P_{now} *:* Identifiant de la période courante (entier incrémenté).
- n_p *:* Nombre de périodes pendant lesquelles un lien consolidé est valide sans être reconstaté.
- *p :* Toutes les *p* périodes, le numéro servant à identifier P_i est remis à zéro.

3.3.3.4. Aspect décisionnel

Si un facteur de liaison consolidé fc_{ij} est supérieur au seuil Tf_c*, alors il possède une valeur suffisamment importante pour justifier le rapprochement physique de O_i et O_j. Dans ce cas, une demande de regroupement est enregistrée dans une table des demandes (*ReclusteringTable*). Les regroupements sont traités de façon différée.

La détermination du seuil Tf_c est essentielle :

- trop réactif : de trop nombreux déplacements physiques d'objets seront effectués ;
- trop peu réactif : l'intérêt d'une solution dynamique est diminué car des objets potentiellement mal placés sont maintenus à la même place trop longtemps.

3.3.3.5. Déroulement de la phase

L'algorithme qui régit la phase de consolidation est présenté en Figure 6.18.

3.3.4. Regroupement

La phase de consolidation fournit une table dont chaque enregistrement contient la désignation d'une cellule de MC. Chacun de ces enregistrements, contenant un couple *(OID1, OID2)*, indique qu'il faut regrouper les deux objets concernés.

3.3.4.1. Construction des Unités de Regroupement

Une Unité de Regroupement (UR) est un graphe d'objets en relation (contenant deux objets au minimum), correspondant à une ou plusieurs demandes de rapprochement liées par transitivité. Elle isole tous les liens qui appartiennent à un même sous-graphe, c'est-à-dire les composantes connexes présentes dans la table des demandes (Figure 6.19).

* Tf_c : *Threshold value for Consolidated linking Factors.*

```
// Traitement de chaque facteur élémentaire
Pour i = 1, n faire
     Pour j = 1, n-1 faire
          Si fe_ij ≠ 0 alors
               // Consolidation du facteur de liaison
               fc_ij = w*fc_ij + (1-w)*fe_ij
               // La date de mise à jour correspondante reçoit l'identifiant de
               // la période courante.
               date_ij = P_i+1
               // Si le facteur consolidé excède la valeur seuil et que les
               // objets concernés ne sont pas déjà regroupés
               Si fc_ij > Tf_c et group_ij = 0 alors
                    n_d = n_d + 1
                    // Enregistrement d'une demande de regroupement
                    demande (d) = (O_i, O_j)
               Fin si
          Fin si
     Fin pour
Fin pour
```

Figure 6.18 : Phase de consolidation

Table des demandes

O_1	O_4
O_7	O_3
O_2	O_9
O_5	O_3
O_{12}	O_4
O_8	O_7

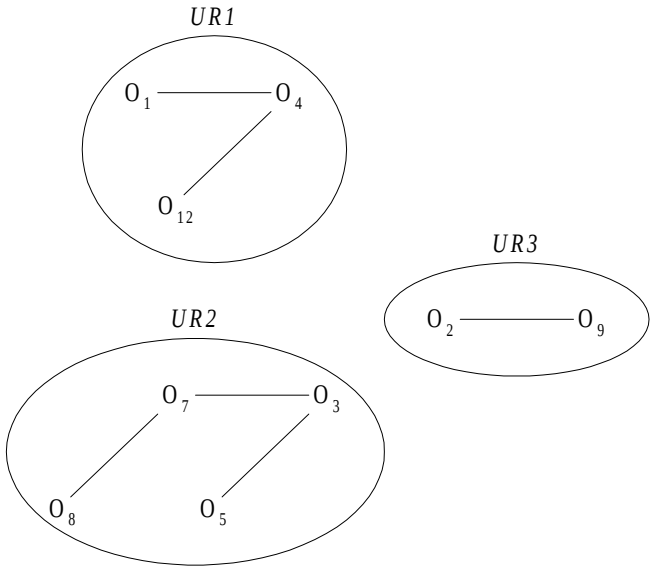

Figure 6.19 : Détection des Unités de Regroupement à partir de la table des demandes

La table des demandes est lue séquentiellement. Un lien est placé dans une Unité de Regroupement lorsqu'il possède déjà un objet commun avec cette UR. Un lien entre deux objets O_i et O_j, avec $O_i \in UR_i$ et $O_j \in URj$, provoque la fusion des deux unités de regroupement.

Pour identifier un sous-graphe, il est possible d'utiliser comme représentation n'importe lequel des liens de chaque unité de regroupement. La recherche des composantes connexes dans la matrice de consolidation MC permet de retrouver tous les objets liés du graphe. Sur l'exemple de la Figure 6.19, on trouve donc : (O_1, O_4), (O_3, O_5), (O_2, O_9).

3.3.4.2. Région de regroupement

La région de regroupement dans DSTC est le *cluster*. Un *cluster* est une unité physique qui permet le stockage des objets liés sur des pages physiquement contiguës. C'est aussi une unité logique, car elle est gérée par le SGBD pour obtenir les informations nécessaires à l'administrateur (nombre de clusters courants et taille).

Un *cluster* est créé à chaque traitement d'une Unité de Regroupement. Un objet appartient au plus à un *cluster*. Un *cluster* a au plus la taille d'une base de données.

3.3.4.3. Traitement différé des demandes

Les demandes de regroupement mémorisées dans la *ReclusteringTable* sont traitées quand la charge est faible, c'est-à-dire dès que le taux transactionnel courant est jugé suffisamment bas.

La charge du système (le taux d'utilisation CPU et le nombre d'entrées-sorties courant) est périodiquement examinée. Quand les deux valeurs sont inférieures à une charge minimum, un *trigger* `is_idle()` déclenche le traitement d'une Unité de Regroupement. C'est l'enregistrement le plus ancien qui est choisi pour le déplacement d'un objet.

À partir des UR traitées lors de la réduction de la table des demandes, les composantes connexes de cette UR sont recherchées dans la matrice de consolidation. Ainsi, tout le graphe d'objets en relation pour cette UR est retrouvé. L'ensemble de ces objets correspond à un *cluster* après l'exécution de l'algorithme de placement (cf. Section 3.3.4.4).

Le sous-graphe obtenu, composé d'arcs (valeurs de critères des liens) et de nœuds (OID des objets), doit être transformé en une séquence linéaire où chaque objet possède une place particulière relativement aux autres.

La création d'un *cluster* correspond à un déplacement explicite des objets concernés dans un but de regroupement. De nouveaux *clusters* apparaissent et les plus anciens peuvent se vider progressivement. Les objets peu utilisés n'apparaissant pas ou peu dans les séquences d'accès sont laissés dans leur *cluster* d'origine.

La Figure 6.21 présente la constitution d'un *cluster* à partir de la matrice de consolidation de la Figure 6.20. Cette matrice de consolidation fait apparaître des liens nouveaux issus de la dernière consolidation (ils sont indiqués en caractères gras italiques) et des liens constatés lors de précédentes périodes d'observation, puis consolidés. La recherche des sous-graphes dans la matrice de consolidation permet de trouver les composantes connexes à O_1, O_4 et O_{12}. Les objets O_1 et O_{12} appartiennent déjà à des *clusters* séparés, ce qui provoque la fusion de ces derniers.

La fusion de clusters existants est une conséquence implicite du regroupement. Soient les objets O_i et O_j et les *clusters* C_i et C_j. Si $O_i \in C_i$ et $O_j \in C_j$. Supposons qu'il est nécessaire de regrouper O_i et O_j, alors la recherche des composantes connexes au lien $O_i \rightarrow O_j$ donne comme résultat un sous-graphe équivalent à la fusion de C_i et C_j.

MC	O_1	O_2	O_3	O_4	O_5	O_7	O_9	O_{12}
O_1		0,9		***1,32***	1,2			
O_2								
O_3								
O_4								***0,93***
O_5								
O_7								1,1
O_9								
O_{12}								

Figure 6.20 : Matrice de consolidation exemple

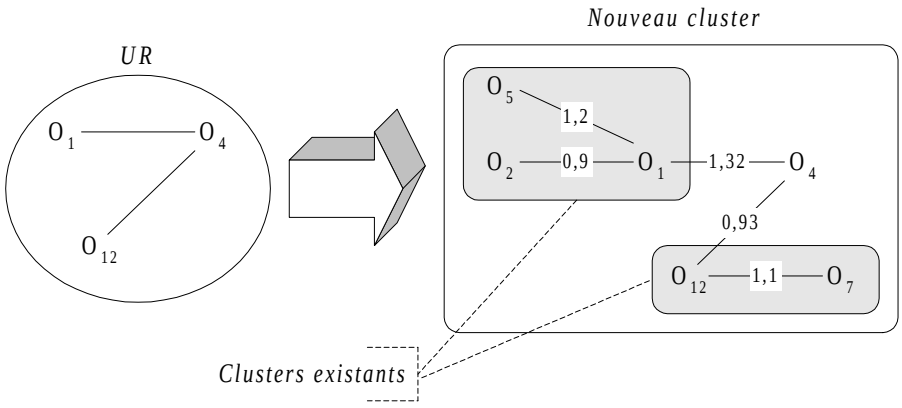

Figure 6.21 : Constitution d'un cluster

3.3.4.4. Algorithme de placement des objets

Le regroupement physique d'objets se fait à partir d'un graphe dont les liens représentent la force d'attraction entre les objets. Ce graphe correspond en fait à la matrice de consolidation. La répartition des objets sur les pages disque s'effectue grâce à un algorithme adapté de [Cheng et Hurson 91] pour le traitement de graphes non orientés. Il propose la formation d'une séquence linéaire d'objets qui est ensuite transcrite séquentiellement dans un *cluster*. Cet algorithme est présenté dans la Figure 1.9 et son fonctionnement est illustré dans la Figure 1.10 (cf. Chapitre 1, Section 3.3.2).

3.4. Mise en œuvre de DSTC dans Texas

L'implantation de la méthode DSTC au sein du gestionnaire d'objets persistants Texas exploite la structure modulaire de ce dernier. Sept nouveaux modules permettent de mettre en œuvre toutes les étapes qui constituent la stratégie DSTC.

3.4.1. Module de gestion de la séquence d'appel

Ce module est implanté sous le nom d'*ObjectList* dans Texas. Il met en œuvre un objet « séquence d'appel » (*ObjectList*) qui stocke pendant une transaction toutes les références inter-objets (cf. Section 3.3.1.1). C'est un objet temporaire qui contient des couples d'OID objet appelant/objet appelé. Il est alimenté à chaque déréférencement d'objets.

Lors de la validation d'une transaction (*commit*), l'objet *ObjectList* met à jour la matrice d'observation (objet *Observation*) de la période en cours. Une nouvelle séquence est alors initialisée pour une nouvelle transaction.

3.4.2. Module d'observation

Le module d'observation permet le compte des références entre objets ainsi que celui du nombre d'utilisations de chaque objet. L'objet *Observation* se compose de deux tables de hachage temporaires, de taille dynamique. Il est fait appel aux fonctions de hachage disponibles en standard au sein du code de Texas.

3.4.3. Module de consolidation

Ce module met en œuvre une classe nommée *Consolidation*. Cette matrice consolidée stocke les statistiques persistantes qui sont à l'origine des regroupements, c'est-à-dire les facteurs *fc* des objets ayant entre eux des relations fortes.

Le module de consolidation assure notamment le filtrage des informations en provenance de la matrice d'observation, grâce aux seuils Tf_a et Tf_c. Lorsque les facteurs *fc* calculés sont sélectionnés, une méthode permet d'effectuer une demande de regroupement. Une autre méthode est chargée de lancer la phase de réorganisation de la base de données, en sélectionnant en une fois tous les liens qui dépassent le seuil, puis en construisant tous les *clusters* relatifs à ces informations.

L'objet *Consolidation* est constitué d'une table de hachage persistante, alimentée en fin de période par les structures temporaires de l'objet *Observation* associées à la base de données correspondante. Cette table de hachage est également de taille variable et contient les facteurs consolidés obtenus sur la base des transactions exécutées.

3.4.4. Module de gestion des regroupements

Ce module gère un ensemble d'informations nécessaires à l'activité de réorganisation d'une base de données. Il est implanté au travers de la classe *Clustering*, dont les instances sont associées à une base de données et existent tant qu'il subsiste des demandes de regroupement pour la base de données correspondante. Un objet *Clustering* comprend essentiellement trois structures de données persistantes :

- la table des demandes (*ReclusteringTable*) ;
- la liste des objets déplacés après le travail de réorganisation (*MovedObjectList*) ;
- un objet liste des *clusters* qui permet de conserver des informations sur les *clusters* physiques qui existent dans la base de données.

La table des demandes sert de base de travail pour la réorganisation déclenchée au sein du module de gestion des demandes. La liste des objets déplacés est utilisée par le module de gestion des objets déplacés, afin de garantir l'intégrité référentielle de la base de données dans le contexte de Texas.

Le module de gestion des regroupements (*Clustering*) fournit également une fonction `StoreCluster()` qui effectue la réorganisation physique associée à un graphe de regroupement donné. Cette fonction est appelée par le module de gestion des demandes de regroupement (*ReclusteringTable*). La fonction `StoreCluster()` reçoit en paramètre la liste ordonnée des OID des objets à grouper (créée par le module d'ordonnancement *BuildCluster*). Chaque OID (adresse disque) permet l'accès direct et le chargement de la page contenant l'ensemble des objets concernés.

L'en-tête des pages renseigne sur la taille effective de chacun de ces objets. Un espace mémoire est alloué en fin de fichier base de données pour recevoir le nouveau *cluster* à former. Les objets y sont copiés dans l'ordre de la séquence. Au fur et à mesure, la place restante est contrôlée afin d'allouer une page supplémentaire si nécessaire.

Lorsque l'ensemble des objets a été copié, un bloc indiquant la fin du *cluster* est inscrit sur la dernière page. Pour chacun des objets ainsi déplacés, l'emplacement anciennement occupé est alors libéré (son adresse est ajoutée dans la liste des emplacements libres gérée par Texas).

Les adresses de l'ancien et du nouvel emplacement de chaque objet déplacé sont conservées au sein de la liste *MovedObjectList*, afin de pouvoir remédier aux problèmes d'intégrité référentielle engendrés.

Lors de la validation, les pages constituant le *cluster* prennent place sur le disque dans l'ordre où la réservation des pages a été effectuée. Cependant, si les pages sont contiguës en mémoire, cette contiguïté n'est pas garantie sur le disque par le système de fichiers d'Unix.

3.4.5. Module de gestion des demandes

Ce module, appelé *ReclusteringTable*, prend en charge le traitement de l'ensemble des demandes de regroupement. Il gère la construction des unités de regroupement à partir de la table des demandes.

Après le traitement de l'ensemble des réorganisations, il faut faire appel à la méthode *UpdatePointer*() du module de gestion des objets déplacés pour maintenir la cohérence des références persistantes.

3.4.6. Module d'ordonnancement des clusters

Ce module, nommé *BuildCluster*, est chargé de l'ordonnancement des objets au sein d'un *cluster*. L'algorithme de DSTC (cf. Section 3.3.4.4) a été implanté à l'aide de quatre classes nommées *SuperNode*, *Link*, *NodeDictionnary* et *LinkDictionnary*.

Pendant l'application de l'algorithme, un super-nœud représente un OID ou une séquence d'OID. À l'initialisation de l'algorithme, un super-nœud est créé pour chaque OID et chaque lien du graphe d'entrée devient une occurrence du dictionnaire des liens. Un lien qui a été traité sort du dictionnaire des liens. De même, un nœud qui a pris part à un regroupement, sort du dictionnaire des super-nœuds.

3.4.7. Module de gestion des objets déplacés

Lorsque les regroupements physiques ont lieu, les objets sont enlevés de leur page d'origine pour être placés dans une autre page appartenant au *cluster* d'accueil. Comme Texas utilise des OID physiques, le module de gestion des objets déplacés permet de mémoriser les déplacements d'objets afin de mettre à jour les pointeurs non valides.

Chapitre 7
Mise en œuvre pour l'évaluation des performances

Nous présentons dans ce chapitre quelques applications de nos outils d'évaluation des performances. Notre objectif est de valider les approches que nous proposons (banc d'essais OCB et modèle de simulation VOODB) en les mettant en œuvre avec des SGBDOO et des algorithmes de regroupement existants. Les systèmes que nous avons sélectionnés sont O_2 [Deux et al. 91] et le gestionnaire d'objets persistants Texas [Singhal et al. 92]. Nous avons également cherché à évaluer les performances de la technique de regroupement dynamique d'objets DSTC [Bullat et Schneider 96], qui est implantée dans Texas.

Nous présentons diverses évaluations de performances effectuées sur O_2, Texas et Texas + DSTC grâce à OCB. Les résultats obtenus sont comparés à ceux fournis par le modèle de simulation VOODB dans les mêmes conditions. Notre objectif est à la fois de démontrer la faisabilité et la pertinence de notre banc d'essais OCB et de valider notre modèle de simulation VOODB, en montrant qu'il fournit des résultats conformes à la réalité.

1. Expériences de validation

Nous présentons dans cette section les expériences que nous avons effectuées sur chaque système et la comparaison des résultats obtenus avec les résultats de simulation correspondants lorsqu'elle est significative. Nous distinguons les expériences effectuées sur les systèmes O_2 et Texas, qui sont rigoureusement identiques, des expériences plus spécifiques au regroupement d'objet que nous avons menées sur DSTC.

1.1. Expériences sur O_2 et Texas

1.1.1. Génération de la base

Nous avons en premier lieu étudié le temps moyen de génération de la base, afin de vérifier que la construction aléatoire de la base d'objets OCB est bien réalisable en pratique. Nous avons également évalué la taille moyenne effective des bases créées, ce qui est important puisque nous ne contrôlons dans OCB que le nombre de classes et le nombre d'instances dans la base. Pour cela, nous avons fait varier les deux paramètres les plus importants concernant la taille de la base de données OCB :

- le *nombre de classes* dans le schéma (*NC*), qui détermine indirectement la taille des objets. En effet, plus les classes sont nombreuses, plus le graphe d'héritage est étendu. Or, une classe hérite d'informations de sa superclasse. La taille de ses instances est donc supérieure ou égale à la taille des instances de sa superclasse. Ainsi, plus le graphe d'héritage est « profond », plus les instances des classes « feuilles » sont volumineuses. Ce phénomène est illustré par le graphe d'héritage de la Figure 7.1 (diagramme de classes UML) ;
- le *nombre total d'instances* de ces classes (*NO*).

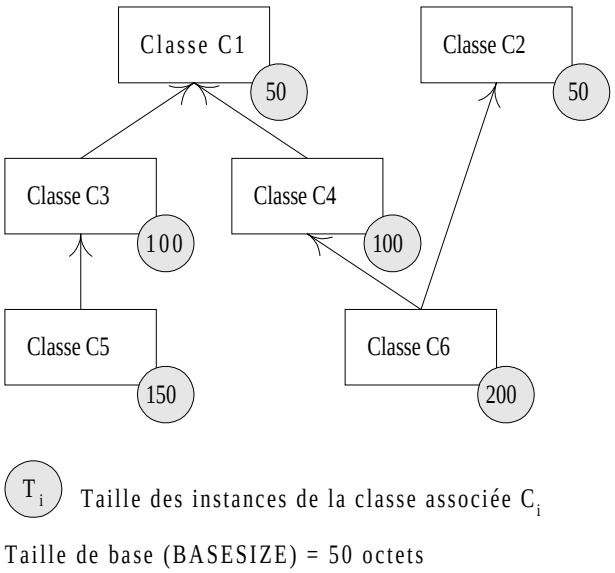

Figure 7.1 : Influence possible du graphe d'héritage OCB sur la taille des instances

Pour des schémas contenant respectivement 10, 20 et 50 classes, nous avons fait varier le nombre d'instances *NO* de 5000 à 50000. De plus, afin d'obtenir des résultats significatifs, nous avons répliqué ces expériences en changeant à chaque fois le germe du

générateur aléatoire d'OCB (*RSEED*), de façon à produire différentes bases et à dégager des valeurs moyennes pour nos résultats.

Comme dans cette expérience, seule la faisabilité du banc d'essais OCB est testée et que le temps de génération de la base de données n'entre pas en compte dans les simulations avec VOODB, nous ne comparons pas les résultats obtenus avec des résultats de simulation.

1.1.2. Utilisation de la base

Le champ des investigations possibles est très vaste, étant donné la multitude de paramètres susceptibles de varier dans le banc d'essais OCB. Dans un souci de concision, nous avons choisi de nous limiter aux expériences qui, selon nous, sont les plus significatives pour cerner le comportement des systèmes étudiés lors de l'utilisation de la base générée par OCB.

1.1.2.1. Variation de la base d'objets

Nous avons en premier lieu fait varier la taille de la base d'objets (nombre de classes et nombre d'instances dans la base) et mesuré les performances des systèmes étudiés (temps de réponse moyen par transaction et nombre d'entrées-sorties nécessaires à l'exécution de ces transactions).

Ces critères de performances sont exprimés de façon globale, et non par type de transaction, comme cela est indiqué dans les spécifications du banc d'essais OCB. Nous avons dû nous restreindre à des valeurs moyennes pour tous types de transaction en raison d'un bogue de la fonction système `getrusage()` de Linux. Cette primitive aurait dû permettre des chronométrages à la microseconde près. Or, il nous a fallu utiliser la fonction `time()`, qui n'est précise qu'à la seconde près. Comme les transactions ont une durée d'exécution de l'ordre de quelques dizaines de millisecondes, il a été impossible de mesurer le temps de réponse de chaque transaction. Nous avons donc mesuré le temps d'exécution total et l'avons divisé par le nombre de transactions exécutées pour obtenir le temps de réponse moyen pour une transaction. Afin de maintenir la cohérence des résultats, le nombre d'entrées-sorties nécessaires à l'exécution des transactions a également été évalué de façon globale.

Nous avons par ailleurs contrôlé la stabilité d'OCB en comptant le nombre moyen d'objets accédés par les principaux types de transaction (les parcours), pour chacune de nos expériences, afin de vérifier qu'il restait constant tout au long de l'expérience. Cette stabilité est nécessaire pour que nos évaluations de performance s'effectuent toujours dans les mêmes conditions. Le nombre d'objets accédés par les transactions ne doit en effet varier que pour les tailles de base les plus petites, lorsque la profondeur des parcours est susceptible de ne pas être atteinte.

Dans cette série d'expériences, le nombre de classes dans le schéma *NC* est fixé à 20 ou 50 et le nombre d'instances dans la base *NO* évolue entre 500 et 20000. Nous avons également étudié les effets de la densité de référence en fixant le paramètre *MAXNREF* à 5 ou 10. *MAXNREF* indique, pour chaque classe (et donc, chaque objet de cette classe), le nombre maximum de références vers d'autres classes (d'autres objets).

Les mesures effectuées sur les systèmes réels avec OCB ont fait l'objet de réplications. Les valeurs moyennes calculées sont comparées à celles obtenues en simulation avec VOODB.

1.1.2.2. Variation de la taille du cache ou de la mémoire

Dans un second temps, nous avons fait varier la taille de la mémoire cache du serveur (dans le cas d'O_2) ou de la mémoire vive disponible (dans le cas de Texas), afin de constater les effets sur les performances du système (temps de réponse moyen et nombre moyen d'entrées-sorties).

L'objectif est également de simuler la réaction du système lorsque le rapport $\frac{\text{taille mémoire}}{\text{taille base d'objets}}$ diminue. Comme nous répliquons plusieurs fois nos expériences afin d'obtenir des résultats significatifs, générer des bases trop grosses de façon répétitive aurait eu un coût prohibitif. Par contre, il est facile de jouer sur la taille du cache ou de la mémoire vive.

Sous O_2, la taille du cache du serveur est spécifiée grâce à des variables d'environnement. Notre version de Texas quant à elle est implantée sous Linux. Il est possible d'indiquer au démarrage au noyau de ce système d'exploitation la taille de la mémoire vive disponible. La taille du cache ou de la mémoire a varié dans ces expériences de 8 Mo à 64 Mo. La taille de la base d'objets était, elle, fixe (50 classes, 20000 instances). Les mesures effectuées sur les systèmes réels avec OCB sont comparées à celles réalisées en simulation avec VOODB.

Les expériences concernant O_2 et Texas sont récapitulées dans la Table 7.1.

1.2. Expériences sur DSTC

Nous nous sommes intéressés dans cette série d'expériences aux gains de performance permis par l'algorithme DSTC. Nous avons mis en évidence la capacité de regroupement de DSTC, en le plaçant dans des conditions très favorables. Pour cela, nous avons procédé en trois phases :

Expérience	Résultats	Paramètres qui varient
Génération de la base	Temps moyen de génération Taille effective de la base d'objets	Nombre de classes (10, 20, 50) Nombre d'instances (5000 à 50000)
Variation de la taille de la base	Nombre moyen d'objets accédés par les accès ensemblistes Nombre moyen d'objets accédés par les parcours simples Nombre moyen d'objets accédés par les parcours hiérarchiques Nombre moyen d'objets accédés par les parcours stochastiques Nombre moyen d'entrées-sorties Temps de réponse moyen	Nombre de classes (20, 50) Nombre max. de références (5, 10) Nombre d'instances (500 à 20000)
Variation de la taille du cache ou de la mémoire	Nombre moyen d'entrées-sorties Temps de réponse moyen	Taille du cache / de la mémoire vive (8 Mo à 64 Mo)

Table 7.1 : Récapitulatif des expériences réalisées sur O_2 et Texas

1) exécution de transactions très « typées » (parcours hiérarchiques selon un type donné de référence ou parcours simples, à partir de racines prédéterminées) et collecte des statistiques utilisées par DSTC (phases d'observation, de sélection et de consolidation) ;
2) regroupement des objets et placement physique sur disque ;
3) exécution des mêmes transactions qu'à la phase 1).

Pour évaluer les gains induits par l'utilisation de DSTC, nous avons mesuré les performances de Texas (temps de réponse moyen et nombre moyen d'entrées-sorties nécessaires à l'exécution des transactions) avant et après regroupement, ainsi que la surcharge due à l'utilisation de DSTC. Cette expérience a été répliquée plusieurs fois, puis reproduite en simulation pour permettre une comparaison.

Nous nous sommes également attachés à montrer que le comportement de la méthode de regroupement DSTC était similaire en simulation et dans le système Texas réel. Pour cela, le nombre de *clusters* constitués en moyenne et la taille moyenne de ces *clusters* (en nombre d'objets) ont été évalués.

Nous avons effectué trois séries de tests, à chaque fois sur une base de taille différente. La première base d'objets sélectionnée est petite (10 classes et 1000 instances, soit en moyenne 1,2 Mo) et tient entièrement en mémoire centrale. La deuxième série de tests a été effectuée avec la base d'objets standard d'OCB (50 classes, 20000 instances), qui est de taille moyenne (environ 20 Mo). La troisième série de tests était prévue sur une base plus grande, ne tenant pas entièrement en mémoire centrale, ce qui aurait dû permettre de constater des gains dus au regroupement plus importants. En effet, un bon placement des objets est d'autant plus critique que les pages disque sont souvent *swappées* hors de la mémoire vive. Cependant, des problèmes techniques rencontrés avec Texas et DSTC pour une telle base nous ont empêché de mener cette expérience à bien. Pour contourner le problème, nous avons donc conservé une base d'objets moyenne, mais nous avons diminué la mémoire disponible sous Linux de 64 Mo à 8 Mo afin que la base ne puisse pas y être

entièrement logée. Nous sommes parvenus à effectuer nos tests sans problème dans ces conditions.

Les résultats obtenus pour ces trois séries d'expériences sont présentés de façon identique. Ils sont donnés pour le banc d'essais effectué sur le système réel et pour la simulation. Ils concernent les critères de performance suivants, en termes de temps et de nombre d'entrées-sorties :

- *utilisation avant regroupement* (notée *AV*) *:* exécution des transactions OCB avant regroupement des objets ;
- *consolidation* (notée *C*) *:* surcharge due à la phase de consolidation de DSTC ;
- *réorganisation* (notée *R*) *:* surcharge due à la phase de réorganisation de la base par DSTC ;
- *surcharge due au regroupement* (notée *S*) *:* total de la surcharge due à la consolidation et à la réorganisation ($S = C + R$) ;
- *utilisation après regroupement* (notée *AP*) *:* exécution des transactions OCB après regroupement des objets ;
- *gain :* gain brut induit par le groupement des objets (rapport $\frac{AV}{AP}$).

Nous comptons également le nombre moyen de *clusters* créés par DSTC ainsi que le nombre moyen d'objets par *cluster*.

Les expériences concernant DSTC sont récapitulées dans la Table 7.2.

Résultats	Conditions
Nombre moyen d'entrées-sorties avant regroupement	Petite base (10 classes, 1000 instances)
Temps de réponse moyen avant regroupement	Base moyenne (50 classes, 20000 instances)
Surcharge due à la phase de consolidation (en E/S)	Base moyenne, 8 Mo de mémoire (⇔ grande base)
Surcharge due à la phase de consolidation (en temps)	
Surcharge due à la phase de réorganisation (en E/S)	Parcours hiérarchiques de profondeur 3
Surcharge due à la phase de réorganisation (en temps)	Parcours simples de profondeur 2
Nombre moyen de clusters créés	
Taille moyenne des clusters	
Nombre moyen d'entrées sorties après regroupement	
Temps de réponse moyen après regroupement	

Table 7.2 : Récapitulatif des expériences réalisées sur Texas/DSTC

2. Tests de performance sur O_2

2.1. Configuration matérielle

Le serveur O_2 que nous utilisons est installé sur une station de travail biprocesseur IBM RISC 6000 43P240. Chaque processeur est un Power PC 604e 166. La station dispose d'une mémoire centrale de 1 Go (RAM ECC). Le système d'exploitation est AIX version 4. Le disque utilisé est un IBM Ultrastar 9ZX. Ses performances (Table 7.3, cf. Chapitre 2, Section 4.3) ont été obtenues à partir des caractéristiques techniques disponibles dans [IBM 97].

Temps de recherche	ΔR = 6,3 ms
Temps de latence	ΔL = 2,99 ms
Temps de transfert	ΔT = 0,7 ms

Table 7.3: Performances du disque IBM Ultrastar 9ZX

La version d'O_2 que nous utilisons est la version 5.0. Le cache du serveur est par défaut configuré à une taille de 16 Mo.

Les paramètres définissant le système O_2 au sein du modèle VOODB sont présentés dans la Table 7.4. Les simulations ont été effectuées sur un PC Pentium-II 266 MMX sous Windows 95 (OSR2), disposant de 64 Mo de mémoire vive SDRAM et un PC Pentium-II 266 sous Linux version 2.0.34, disposant de 128 Mo de mémoire vive SDRAM.

Paramètre	Code	Valeur par défaut
Classe de système	SYSCLASS	Serveur de pages
Niveau de multiprogrammation	MULTILVL	10
Nombre d'utilisateurs	NUSERS	1
Taille d'une page disque	PGSIZE	4096 octets
Taille du cache de page	BUFFSIZE	3840 pages
Temps d'acquisition des verrous	GETLOCK	0,5 ms
Temps de restitution des verrous	RELLOCK	0,5 ms
Temps de recherche sur disque	DISKSEA	6,3 ms
Temps de latence disque	DISKLAT	2,99 ms
Temps de transfert disque	DISKTRA	0,7 ms
Débit réseau	NETTHRU	$+\infty$[*]

Table 7.4 : Paramétrage de VOODB pour le système O_2

[*] Dans notre configuration matérielle, le serveur et le client O_2 se trouvent sur la même machine physique.

2.2. Résultats

2.2.1. Génération de la base

2.2.1.1. Temps de génération moyen

La Figure 7.2 présente le temps de génération moyen de la base d'objets OCB, en fonction du nombre d'instances et du nombre de classes. Elle fait apparaître que ce temps de création augmente de façon linéaire lorsque le schéma est constitué de 10 et 20 classes. La progression est par contre plus accentuée pour 50 classes. Ceci vient du fait que lorsque le cache client d'O_2 est saturé, ce qui se produit avec les bases d'objets les plus volumineuses, une erreur d'exécution survient. Pour remédier à ce problème, il est nécessaire de jalonner le processus de génération de la base avec des validations (*commit*). Ce sont ces validations multiples, plus coûteuses qu'une validation unique en fin de création, qui entraînent cette augmentation du temps de génération de la base.

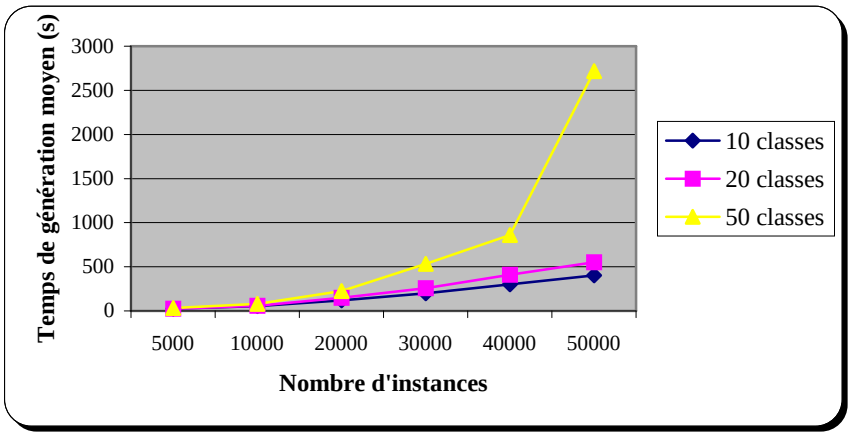

Figure 7.2 : Temps de génération en fonction de la taille de la base (O_2)

2.2.1.2. Taille effective de la base

La Figure 7.3 illustre la façon dont la taille réelle de la base d'objets générée aléatoirement évolue en fonction du nombre de classes et du nombre d'instances spécifié. Cette évolution est linaire, ce qui permet de prévoir aisément la taille finale d'une base de données lors du choix des paramètres *NC* et *NO*.

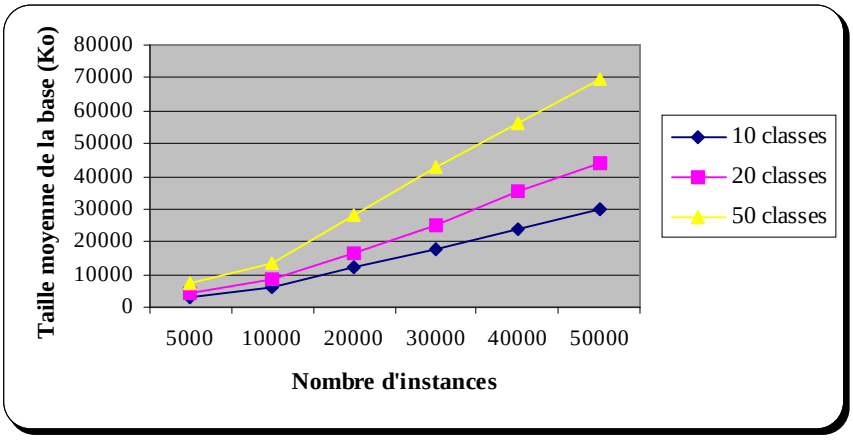

Figure 7.3 : Taille effective de la base en fonction du nombre de classes et d'instances (O_2)

2.2.2. Utilisation de la base

2.2.2.1. Variation de la base d'objets

Les Figures 7.4, 7.5, 7.6 et 7.7 présentent, pour quelques types de transaction, l'évolution du nombre d'objets accédés en moyenne en fonction du nombre d'instances dans la base d'objets, pour les quatre configurations de base que nous testons. Dans chaque cas, ce nombre est constant lorsque la taille de la base augmente, ce qui démontre la stabilité des transactions d'OCB. Par ailleurs, les résultats constatés sur le système réel sont proches de ceux obtenus en simulation.

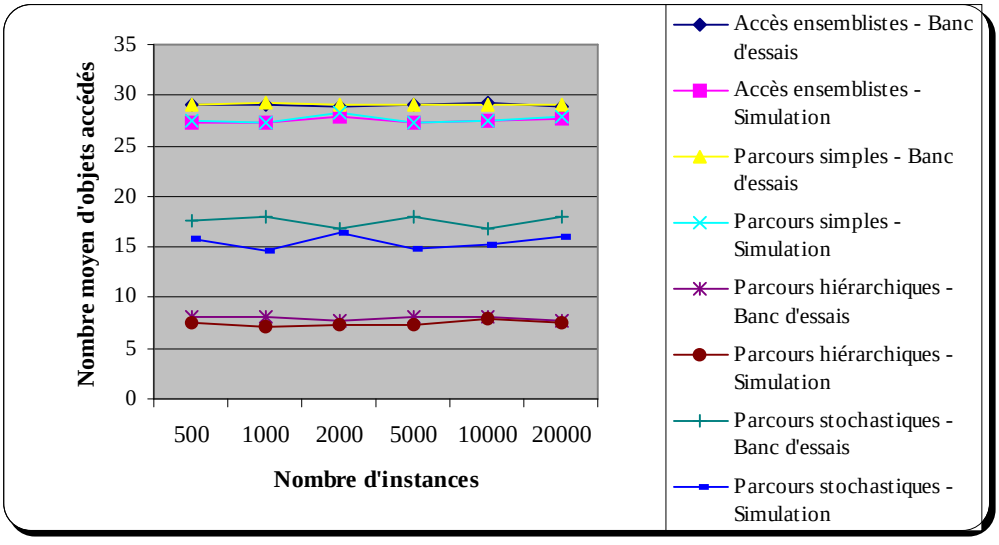

Figure 7.4: Nombre moyen d'objets accédés en fonction du nombre d'instances
O_2 – *NC* = 20, *MAXNREF* = 5

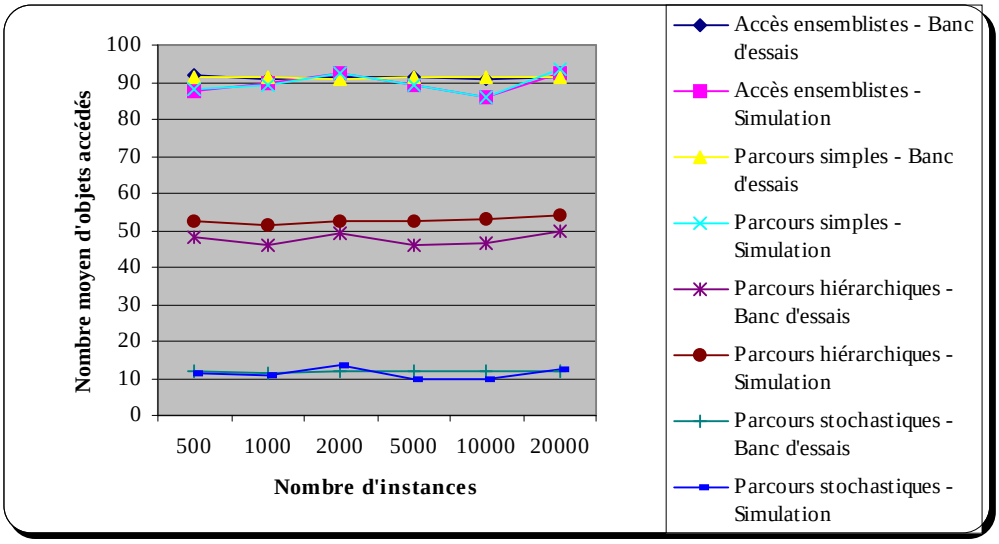

Figure 7.5 : Nombre moyen d'objets accédés en fonction du nombre d'instances
O_2 – *NC* = 20, *MAXNREF* = 10

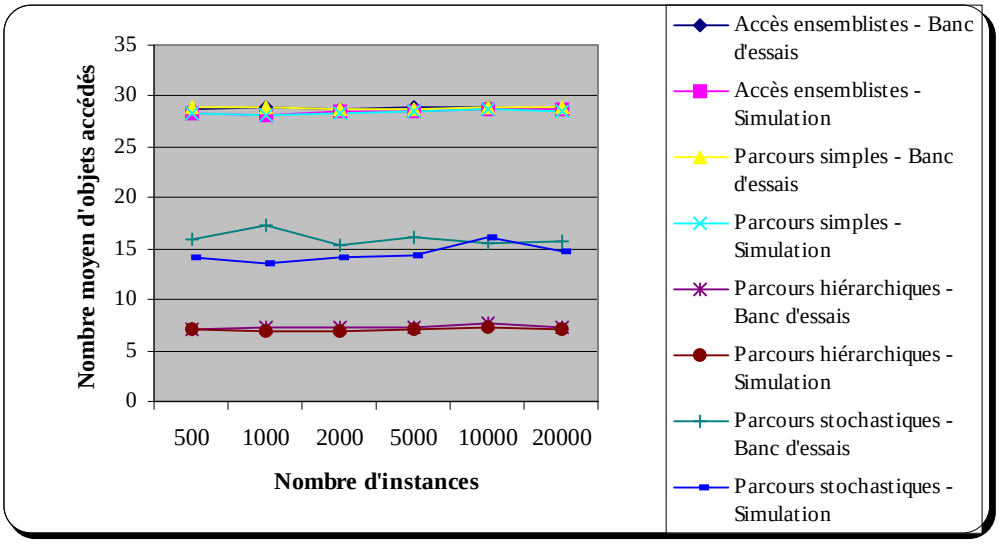

Figure 7.6 : Nombre moyen d'objets accédés en fonction du nombre d'instances
O_2 – *NC* = 50, *MAXNREF* = 5

Il est à noter que les accès ensemblistes et les parcours simples, qui parcourent le même graphe d'objets en largeur d'abord et en profondeur d'abord, respectivement, accèdent bien toujours en moyenne au même nombre d'objets.

La variation du nombre de références influence le comportement des transactions et donc, le nombre d'objets auxquels elles accèdent. En moyenne, quel que soit le nombre de classes, les accès ensemblistes, les parcours simples et les parcours hiérarchiques accèdent à un nombre nettement plus réduit d'objets lorsque le nombre de références passe de 10 à 5. En revanche, les parcours stochastiques utilisent environ deux fois plus d'objets dans ces conditions.

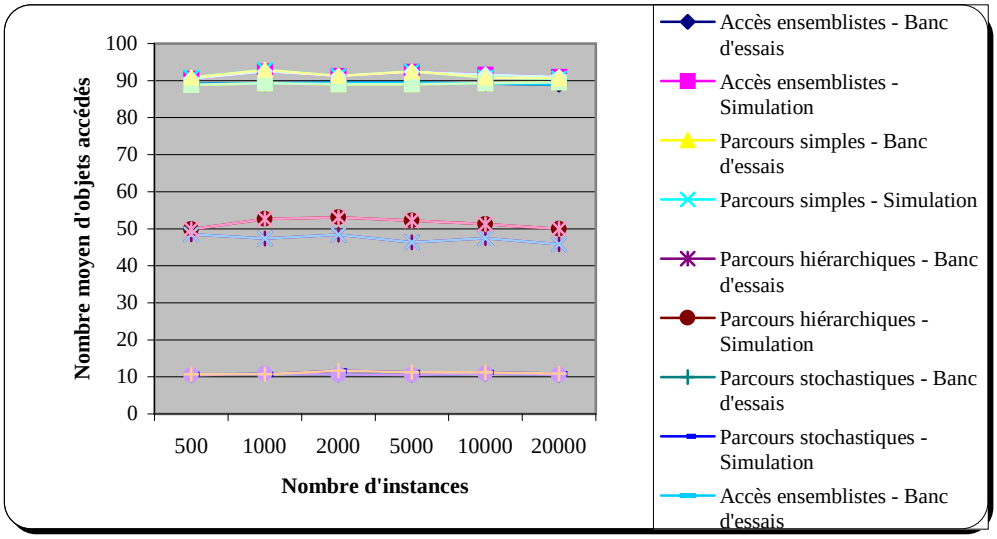

Figure 7.7 : Nombre moyen d'objets accédés en fonction du nombre d'instances
O_2 – *NC* = 50, *MAXNREF* = 10

Les Figures 7.9, 7.10, 7.11 et 7.12 concernent plus directement les performances d'O_2. Elles présentent le nombre d'entrées-sorties globalement nécessaires à l'exécution des transactions et le temps de réponse moyen par transaction, en fonction du nombre d'instances dans la base, pour nos quatre configurations de base d'objets. L'évolution de ces deux critères de performance est similaire, ce qui est normal, étant donné que le gros des traitements effectués par le système lors de l'exécution du banc d'essais OCB consiste à charger des objets depuis le disque. Nous pouvons également constater que les résultats de simulation et les résultats mesurés sur le système réel diffèrent légèrement en valeur absolue mais affichent clairement la même tendance. Le comportement de notre modèle de simulation VOODB est donc bien conforme à la réalité.

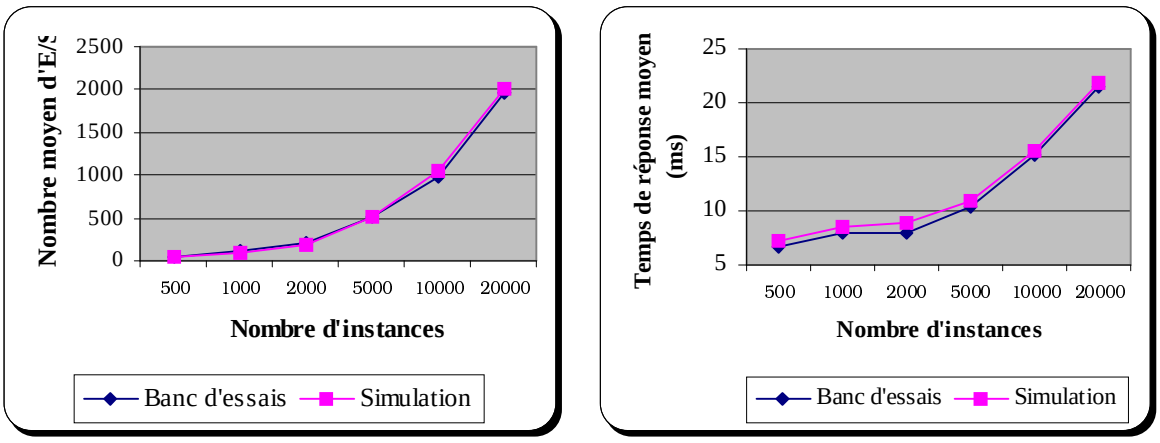

Figure 7.8 : Nombre d'entrées-sorties et temps de réponse moyens en fonction du nombre d'instances
O_2 – *NC* = 20, *MAXNREF* = 5

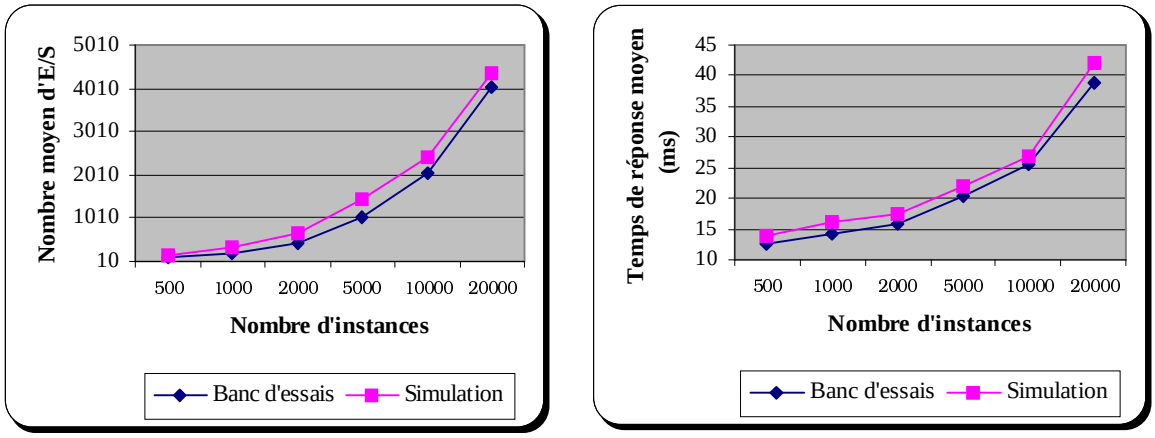

Figure 7.9 : Nombre d'entrées-sorties et temps de réponse moyens en fonction du nombre d'instances
O_2 – *NC* = 20, *MAXNREF* = 10

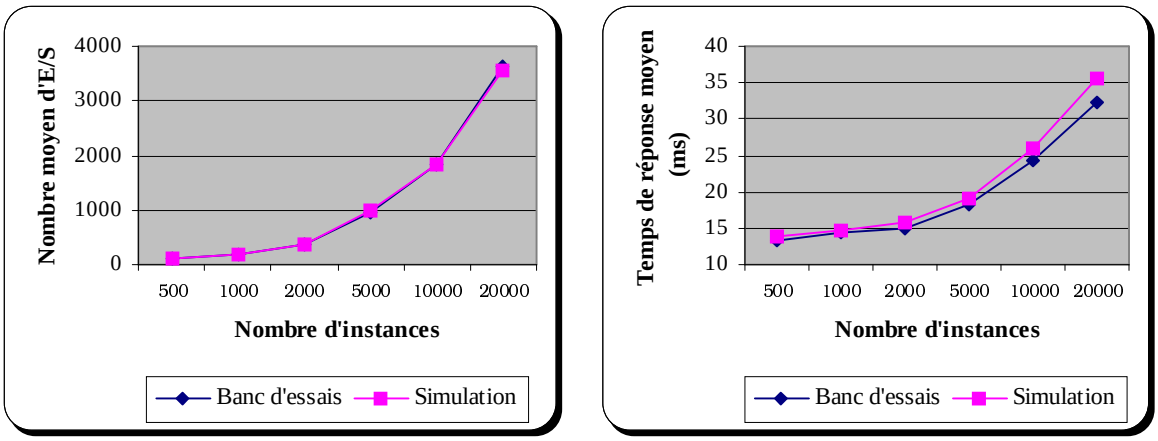

Figure 7.10 : Nombre d'entrées-sorties et temps de réponse moyens en fonction du nombre d'instances
O_2 – *NC* = 50, *MAXNREF* = 5

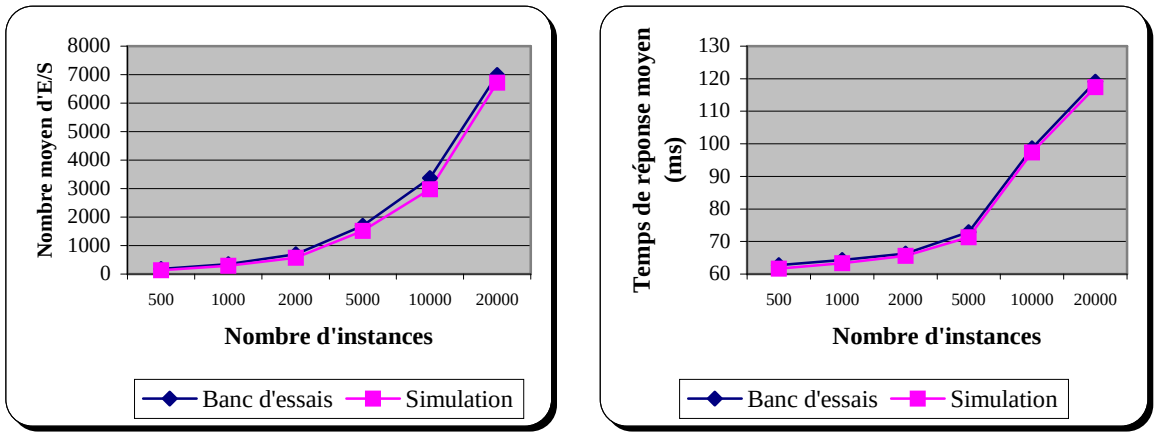

Figure 7.11 : Nombre d'entrées-sorties et temps de réponse moyens en fonction du nombre d'instances
O_2 – *NC* = 50, *MAXNREF* = 10

2.2.2.2. Variation de la taille du cache

Pour cette expérience, nous avons fixé la taille de la base à 50 classes et 20000 instances, ce qui correspond à une taille effective de 28 Mo en moyenne. La taille du cache du serveur O_2 varie de 8 Mo à 64 Mo. En simulation, nous avons donc fait varier la taille du cache mémoire de VOODB de 2000 à 16000 pages. Les résultats obtenus en termes de nombre moyen d'entrées-sorties et de temps de réponse moyen sont présentés respectivement dans la Figure 7.12 et la Figure 7.13. Ces dernières montrent que les performances d'O_2 se dégradent de façon significative lorsque la taille de la base d'objets devient supérieure à la taille du cache. Cette dégradation des performances est linéaire. Ces figures montrent par ailleurs que les performances d'O_2 peuvent à nouveau être retrouvées à l'aide de notre modèle de simulation VOODB.

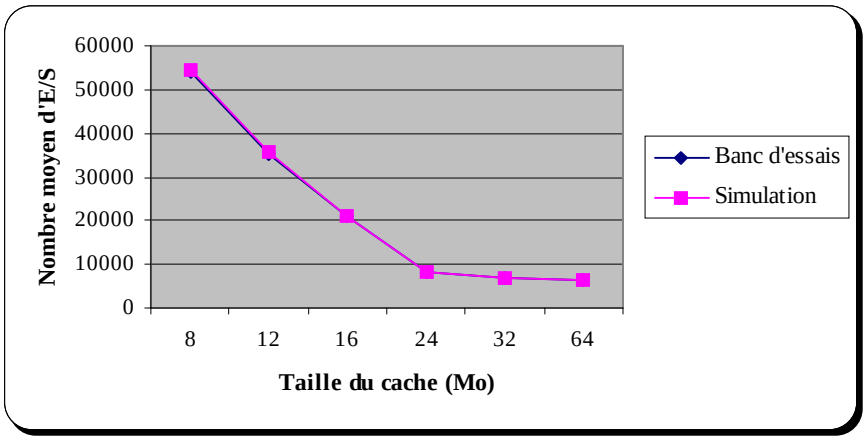

Figure 7.12 : Nombre moyen d'entrées sorties en fonction de la taille du cache
O_2 – *NC* = 50, *NO* = 20000

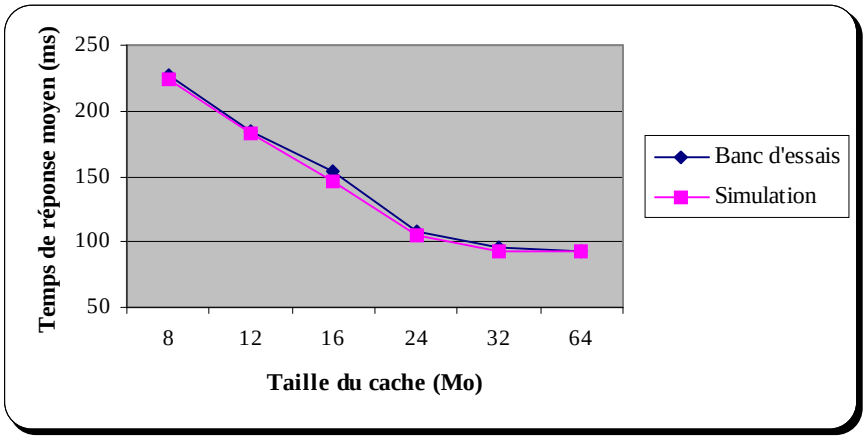

Figure 7.13 : Temps de réponse moyen en fonction de la taille du cache
O_2 – *NC* = 50, *NO* = 20000

3. Tests de performance sur Texas

3.1. Configuration matérielle

Texas est installé sur un PC Pentium-II MMX à 266 MHz disposant d'une mémoire vive de 64 Mo (SDRAM). Le système d'exploitation hôte est Linux version 2.0.30. Il dispose d'une partition de *swap* disque de 64 Mo. Les performances du disque dur utilisé (cf. Chapitre 2, Section 4.3) ont été mesurées grâce au logiciel *coretest* [CORE 88]. Elles sont récapitulées dans la Table 7.5.

Temps de recherche	ΔR = 7,4 ms
Temps de latence	ΔL = 4,3 ms
Temps de transfert	ΔT = 0,5 ms

Table 7.5: Performances du disque utilisé par Texas

La version du prototype Texas que nous utilisons est la version 0.5. Elle a été compilée grâce au compilateur GNU C++ version 2.7.2.1.

Les paramètres définissant le système Texas au sein du modèle VOODB sont présentés dans la Table 7.6. Les simulations ont été effectuées sur un PC Pentium-II 266 MMX sous Windows 95 (OSR2), disposant de 64 Mo de mémoire vive SDRAM et un PC Pentium-II 266 sous Linux version 2.0.34, disposant de 128 Mo de mémoire vive SDRAM.

Paramètre	Code	Valeur par défaut
Classe de système	SYSCLASS	Centralisé
Niveau de multiprogrammation	MULTILVL	1
Nombre d'utilisateurs	NUSERS	1
Taille d'une page disque	PGSIZE	4096 octets
Taille du cache de page	BUFFSIZE	3275 pages
Temps d'acquisition des verrous	GETLOCK	0
Temps de restitution des verrous	RELLOCK	0
Temps de recherche sur disque	DISKSEA	7,4 ms
Temps de latence disque	DISKLAT	4,3 ms
Temps de transfert disque	DISKTRA	0,5 ms
Débit réseau	NETTHRU	N/A

Table 7.6 : Paramétrage de VOODB pour le système Texas

3.2. Résultats

3.2.1. Génération de la base

3.2.1.1. Temps de génération moyen

La Figure 7.14 présente le temps de génération moyen de la base d'objets OCB, en fonction du nombre d'instances et du nombre de classes. Elle fait apparaître que ce temps de création n'augmente pas de façon linéaire. Cependant, les temps de génération les plus longs sont de l'ordre d'une dizaine de minutes, ce qui est très acceptable.

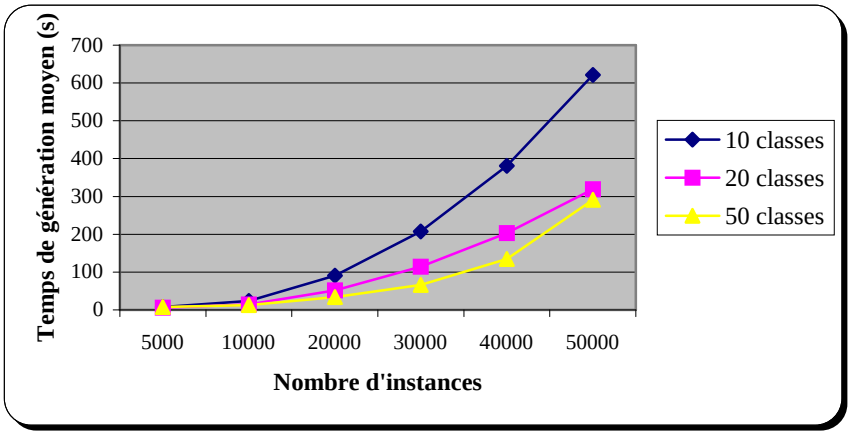

Figure 7.14 : Temps de génération en fonction de la taille de la base (Texas)

3.2.1.2. Taille effective de la base

La Figure 7.15 illustre la façon dont la taille réelle de la base d'objets générée aléatoirement évolue en fonction du nombre de classes et du nombre d'instances spécifié. Comme dans le cas d'O_2, cette évolution est linaire, ce qui permet de prévoir aisément la taille finale d'une base de données lors du choix des paramètres *NC* et *NO*.

3.2.2. Utilisation de la base

3.2.2.1. Variation de la base d'objets

Les Figures 7.16, 7.17, 7.18 et 7.19 présentent tout d'abord, pour quelques types de transaction, l'évolution du nombre d'objets accédés en moyenne en fonction du nombre d'instances dans la base, pour les quatre configurations de base que nous testons. Ce nombre reste bien constant, comme c'est nécessaire pour la validité de nos tests, lorsque la taille de la base augmente. Par ailleurs, les résultats constatés sur le système réel sont retrouvés en simulation.

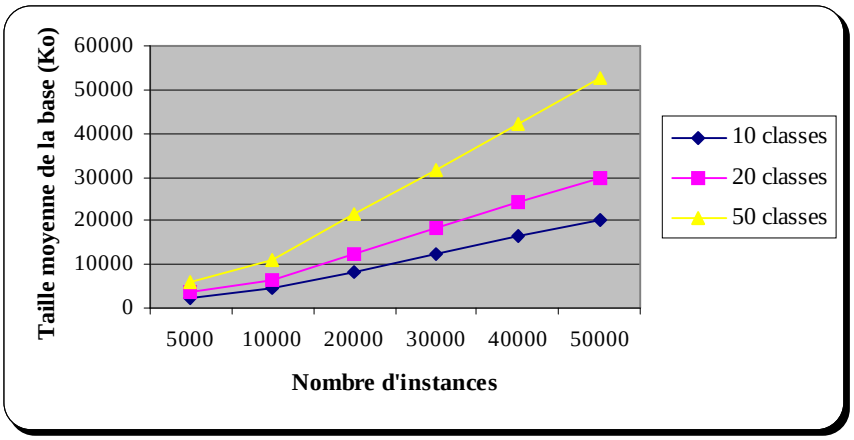

Figure 7.15 : Taille effective de la base en fonction du nombre de classes et d'instances (Texas)

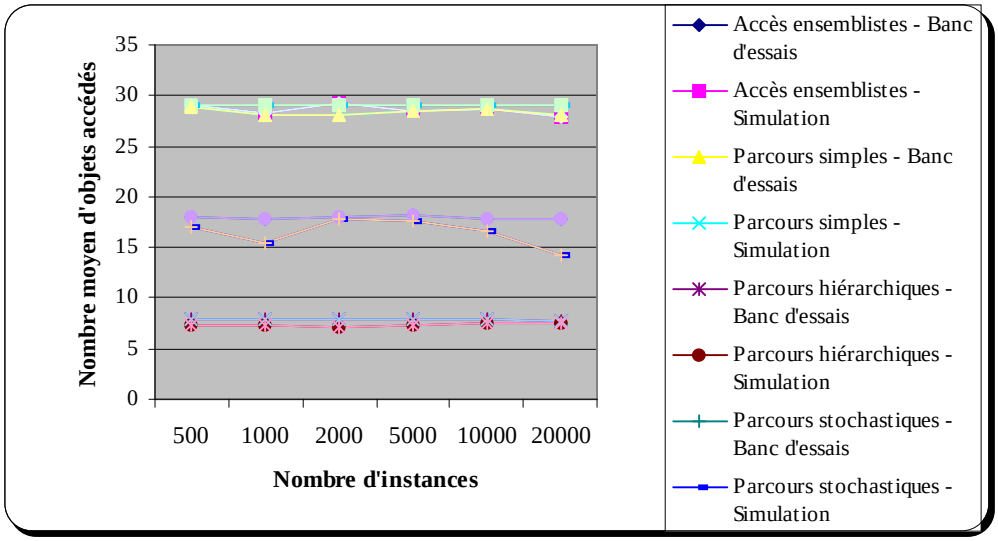

Figure 7.16: Nombre moyen d'objets accédés en fonction du nombre d'instances
Texas – *NC* = 20, *MAXNREF* = 5

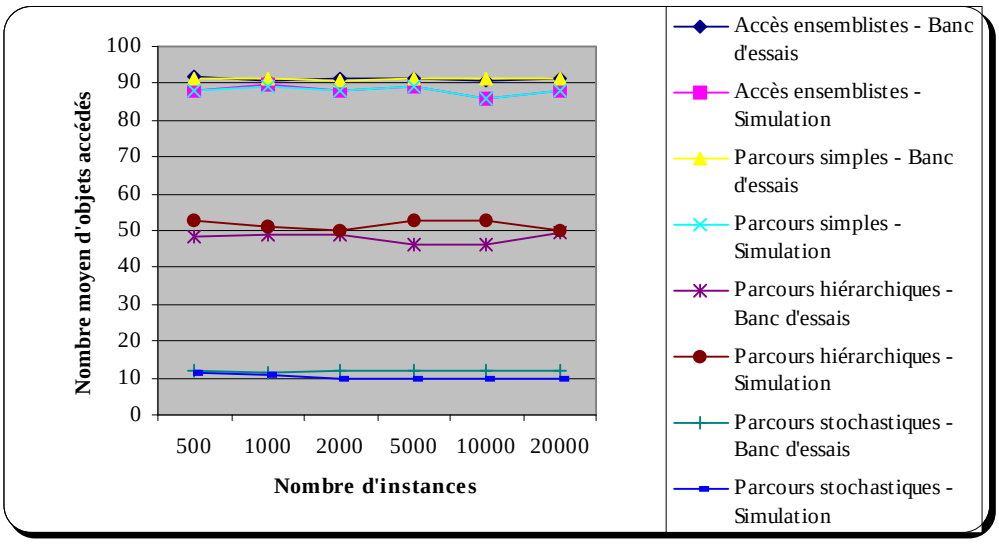

Figure 7.17 : Nombre moyen d'objets accédés en fonction du nombre d'instances
Texas – *NC* = 20, *MAXNREF* = 10

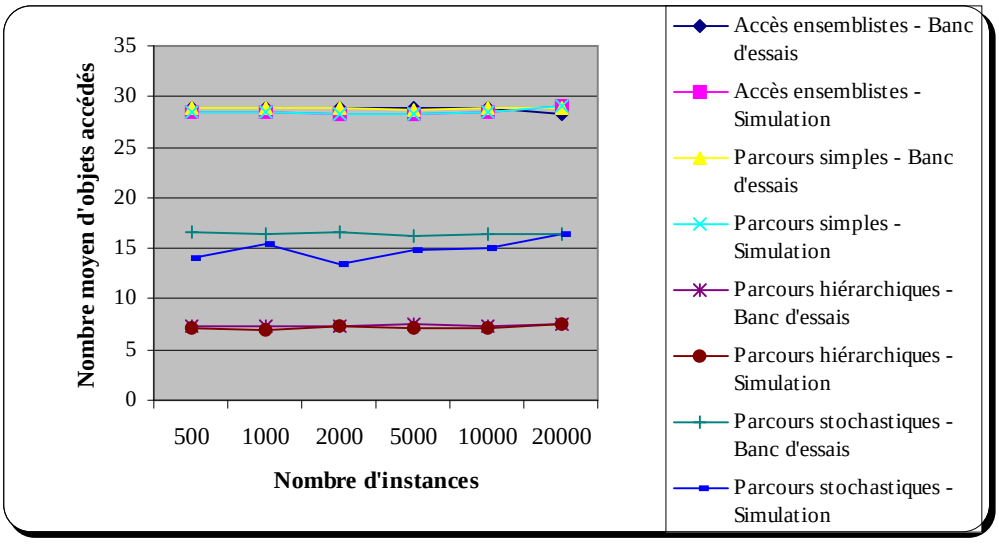

Figure 7.18 : Nombre moyen d'objets accédés en fonction du nombre d'instances
Texas – *NC* = 50, *MAXNREF* = 5

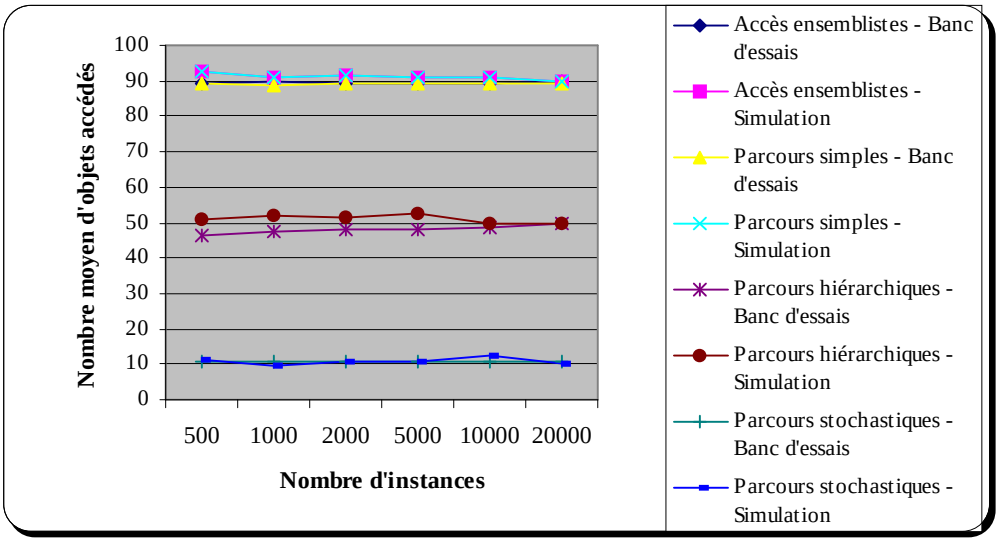

Figure 7.19 : Nombre moyen d'objets accédés en fonction du nombre d'instances
Texas – *NC* = 50, *MAXNREF* = 10

Les Figures 7.20, 7.21, 7.22 et 7.23 concernent les performances de Texas. Elles présentent respectivement le nombre d'entrées-sorties globalement nécessaires à l'exécution des transactions et le temps de réponse moyen par transaction, en fonction du nombre d'instances dans la base, pour nos quatre configurations de base. Comme dans le cas d'O_2, la corrélation entre ces deux critères de performance apparaît évidente. Nous pouvons de plus constater que, comme dans le cas d'O_2, les résultats de simulation et ceux mesurés sur le système réel diffèrent légèrement en valeur absolue, mais affichent la même tendance.

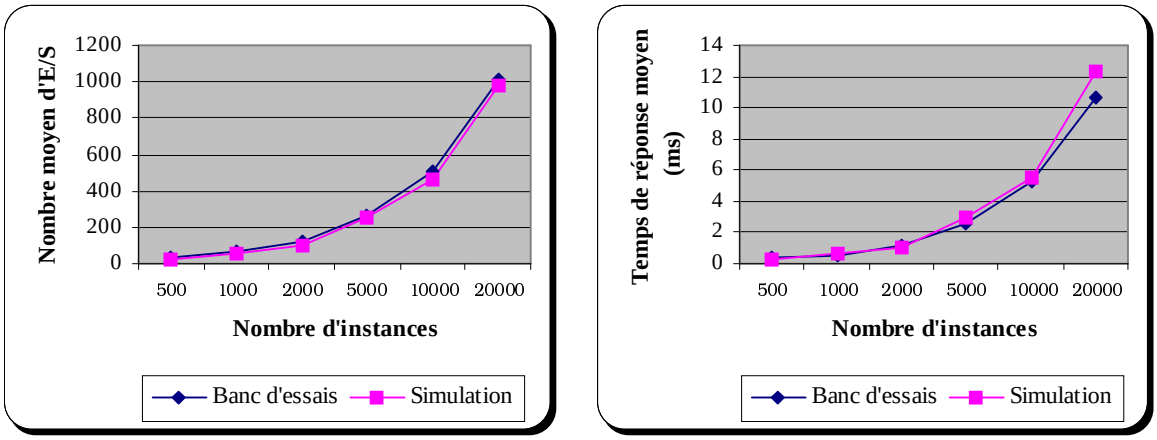

Figure 7.20 : Nombre d'entrées-sorties et temps de réponse moyens en fonction du nombre d'instances Texas – *NC* = 20, *MAXNREF* = 5

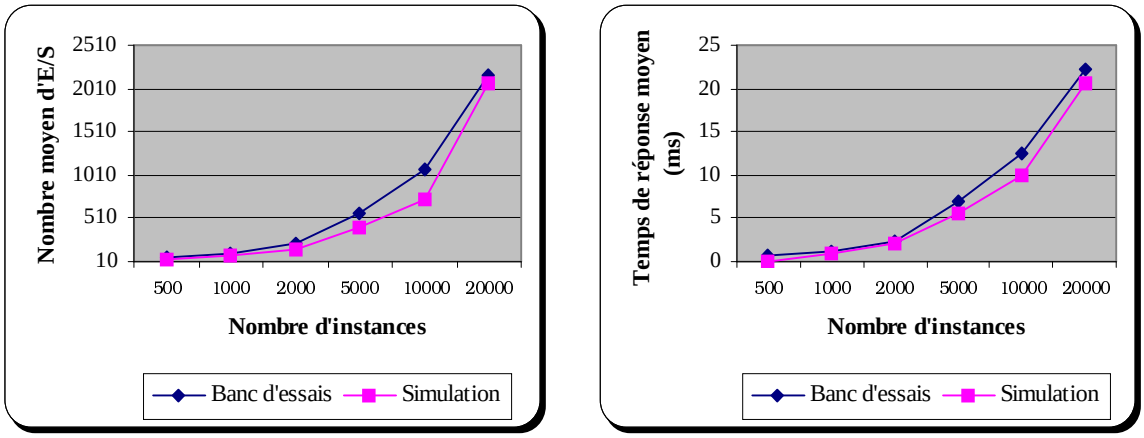

Figure 7.21 : Nombre d'entrées-sorties et temps de réponse moyens en fonction du nombre d'instances Texas – *NC* = 20, *MAXNREF* = 10

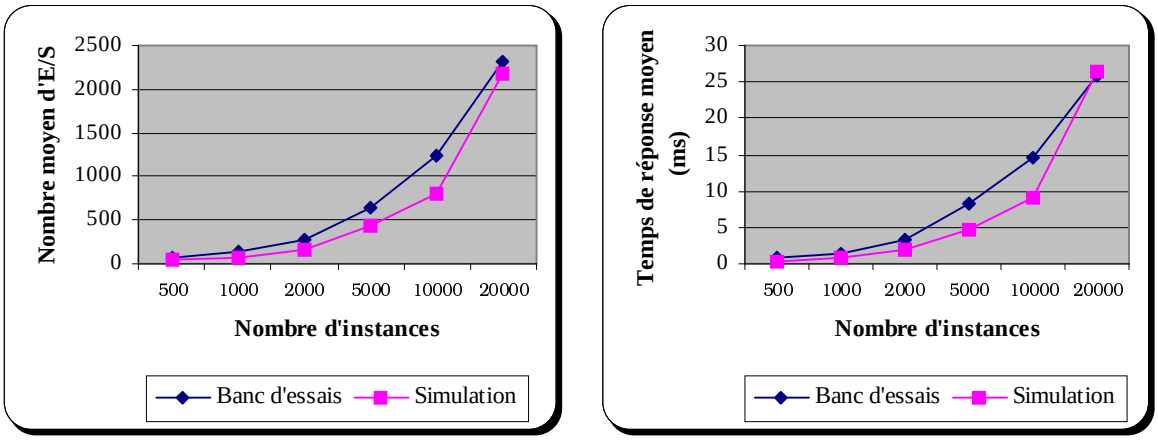

Figure 7.22 : Nombre d'entrées-sorties et temps de réponse moyens en fonction du nombre d'instances Texas – *NC* = 50, *MAXNREF* = 5

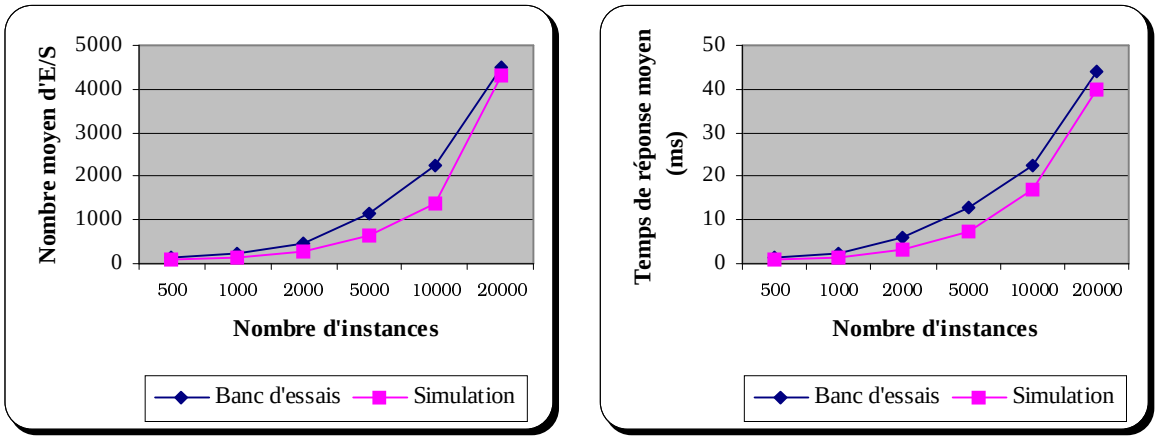

Figure 7.23 : Nombre d'entrées-sorties et temps de réponse moyens en fonction du nombre d'instances Texas – *NC* = 50, *MAXNREF* = 10

3.2.2.2. Variation de la taille de la mémoire

Pour cette expérience, la taille de la base est fixée à 50 classes et 20000 instances, ce qui correspond à une taille effective de 21 Mo en moyenne. Comme Texas fait appel aux mécanismes de mémoire virtuelle du système d'exploitation, nous avons étudié les effets d'une réduction de la taille de la mémoire centrale disponible sous Linux. Nous avons fait varier cette taille de 64 Mo à 8 Mo. C'est en jouant sur la taille de la mémoire cache dans notre modèle VOODB que nous parvenons à simuler ce phénomène. Cependant, Texas utilise les mécanismes de mémoire virtuelle du système d'exploitation hôte. Or, la quantité de mémoire vive disponible influence également directement la taille du cache disque maintenu par Linux, sur lequel nous n'avons aucun contrôle. Aussi est-il plus difficile que dans le cas d'O_2 de faire correspondre exactement la taille mémoire disponible sous Linux à la taille de la mémoire cache en simulation. La taille de la mémoire cache en simulation a varié de 1 à 3000 pages. Ceci ne correspond à aucune valeur de taille de mémoire vive sous Linux, mais permet néanmoins de retrouver fidèlement les performances de Texas. Nous avons de surcroît augmenté les performances du disque (de 12 ms par E/S en moyenne à 4 ms au mieux) afin de simuler les effets du cache disque, dont la taille augmente avec celle de la mémoire vive.

Les résultats obtenus en termes de nombre moyen d'entrées-sorties et de temps de réponse moyen sont présentés respectivement dans la Figure 7.24 et la Figure 7.25. Ces dernières montrent que les performances de Texas se dégradent rapidement lorsque la mémoire centrale diminue.

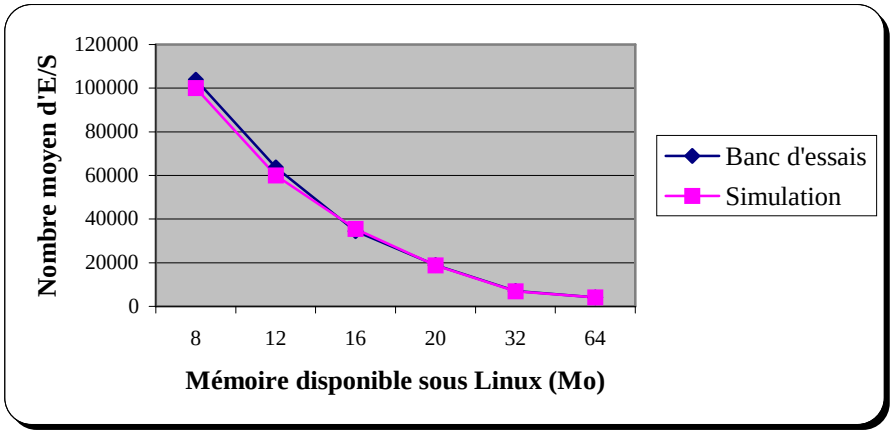

Figure 7.24 : Nombre moyen d'entrées-sorties en fonction de la taille de la mémoire (Texas)

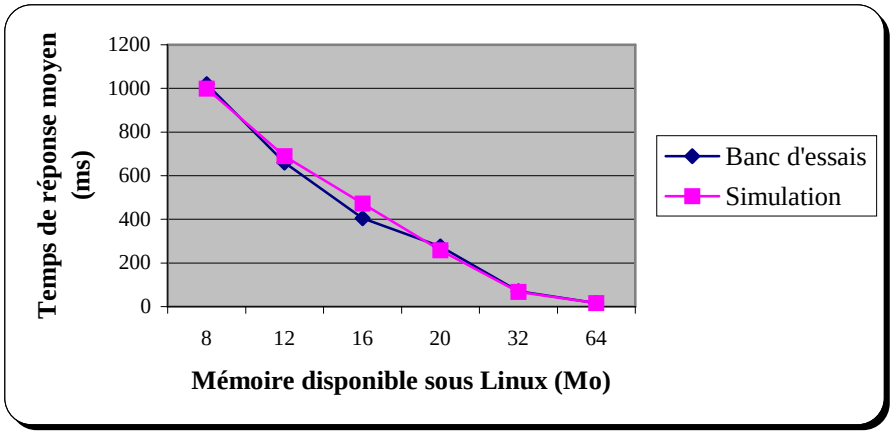

Figure 7.25 : Temps de réponse moyen en fonction de la taille de la mémoire (Texas)

4. Tests de performances sur Texas/DSTC

4.1. Configuration matérielle

DSTC est implanté dans la version 0.5 du prototype Texas (cf. Section 3.1), sous la forme de modules additionnels. Ces modules ont été, comme Texas, compilés grâce au compilateur GNU C++ version 2.7.2.1.

Les simulations ont été effectuées sur un PC Pentium-II 266 MMX sous Windows 95 (OSR2), disposant de 64 Mo de mémoire vive SDRAM et un PC Pentium-II 266 sous Linux version 2.0.34, disposant de 128 Mo de mémoire vive SDRAM.

4.2. Résultats

4.2.1. Conditions des tests

Les trois séries de tests ci-dessous concernent des tailles de base différentes. En revanche, les transactions que nous avons appliquées à ces bases sont du même type :

- *parcours hiérarchiques* de profondeur 3 selon un type de référence donné (par exemple, le lien de composition) ;
- *parcours simples* de profondeur 2.

La profondeur des parcours a été réduite par rapport aux paramétrage par défaut d'OCB, afin que les *clusters* générés ne soient pas trop gros et que les effets du regroupement soient nets. Par ailleurs, lorsque la taille de la base augmente, des problèmes techniques apparaissent avec Texas lors de la fabrication par DSTC de *clusters* de grande taille. Les parcours ont été effectués à partir de 100 objets racines prédéterminés. Chaque parcours a été exécuté 10 fois.

Afin d'obtenir le meilleur regroupement qui soit, DSTC a été paramétré de façon à prendre en compte quasiment toutes les statistiques recueillies durant la phase d'observation (Table 7.7).

Paramètre	Valeur
Tf_a	1
Tf_e	0
Tf_c	0,4

Table 7.7 : Valeur des paramètres de DSTC

4.2.2. Base d'objets de petite taille

Les tables qui suivent présentent les nombres moyens d'entrées-sorties et les temps moyens obtenus par mesures sur le système réel et en simulation, en ce qui concerne l'utilisation de la base avant et après regroupement. La Table 7.8 et la Table 7.9 correspondent à l'utilisation de parcours hiérarchiques et la Table 7.10 et la Table 7.11 à l'utilisation de parcours simples.

	Banc d'essais	Simulation	Rapport
Utilisation avant regroupement	69,25	68,82	1,0062
Consolidation	3	0	—
Réorganisation	267,87	96,70	2,7701
Surcharge due au regroupement	*270,87*	*96,70*	*2,8011*
Utilisation après regroupement	13,37	12,10	1,1053
Gain	*5,17*	*5,68*	*0,9102*

Table 7.8 : Effets de DSTC sur les performances (nombres moyens d'E/S)
Petite base — *NC* = 10, *NO* = 1000 — Parcours hiérarchiques

	Banc d'essais	**Simulation**	**Rapport**
Utilisation avant regroupement	0,625	0,60	1,0416
Consolidation	11,50	0	—
Réorganisation	1,00	0,30	3,3333
Surcharge due au regroupement	*12,50*	*0,30*	*41,6666*
Utilisation après regroupement	—	0,092	—
Gain	—	*6,52*	—

Table 7.9 : Effets de DSTC sur les performances (temps moyens en ms)
Petite base — *NC* = 10, *NO* = 1000 — Parcours hiérarchiques

Note : Sur le système réel, le temps d'utilisation de la base après regroupement est très faible. Comme nous le mesurons à l'aide d'une fonction système précise à la seconde, nous n'avons pas pu l'établir, car elle a renvoyé 0 lors de tous nos tests.

	Banc d'essais	**Simulation**	**Rapport**
Utilisation avant regroupement	59,83	61,83	0,9677
Consolidation	3	0	—
Réorganisation	277	96	2,8854
Surcharge due au regroupement	*280*	*96*	*2,9166*
Utilisation après regroupement	11,66	12,90	0,9043
Gain	*5,67*	*4,79*	*1,1830*

Table 7.10 : Effets de DSTC sur les performances (nombres moyens d'E/S)
Petite base — *NC* = 10, *NO* = 1000 — Parcours simples

	Banc d'essais	**Simulation**	**Rapport**
Utilisation avant regroupement	0,50	0,46	1,0869
Consolidation	13,16	0	—
Réorganisation	1,16	0,56	2,0833
Surcharge due au regroupement	*14,33*	*0,56*	*25,5952*
Utilisation après regroupement	—	0,085	—
Gain	—	*5,41*	—

Table 7.11 : Effets de DSTC sur les performances (temps moyens en ms)
Petite base — *NC* = 10, *NO* = 1000 — Parcours simples

Nos résultats montrent que, pour les deux types de parcours employés, la technique de regroupement DSTC permet des améliorations de performance substantielles (gain de performance de facteur supérieur à 5). Par ailleurs, les résultats de simulation sont dans l'ensemble cohérents avec les mesures de performances effectuées sur le système réel, sauf en ce qui concerne la surcharge due au regroupement des objets, qui est beaucoup moins importante en simulation qu'en réalité.

Cette incohérence flagrante n'est pas due à un bogue du modèle de simulation, mais à une particularité de Texas. En effet, après réorganisation de la base par DSTC, des objets sont déplacés au sein de la base. Ils changent donc d'OID, puisque les OID sont physiques au sein de Texas. Pour maintenir la cohérence des références inter-objets, toute la base doit être parcourue et toutes les références vers des objets déplacés doivent être mises à jour.

Cette phase très coûteuse en temps et en entrées-sorties n'a pas lieu d'être dans nos modèles de simulation, car ceux-ci utilisent nécessairement des OID logiques.

Pour simuler de façon totalement fidèle le comportement de DSTC au sein de Texas, il nous aurait été facile de prendre ce temps de conversion en compte. Néanmoins, nous avons préféré conserver nos résultats initiaux afin de souligner la difficulté d'implanter une technique de regroupement dynamique au sein d'un gestionnaire d'objets utilisant des OID physiques. Nos simulations montrent, quant à elles, qu'une telle technique dynamique est parfaitement viable dans un système à OID logiques.

Le nombre de *clusters* constitués en moyenne par la méthode DSTC ainsi que la taille moyenne de ces *clusters* (en nombre d'objets) sont présentés dans la Table 7.12 et la Table 7.13. Elles montrent qu'il y a peu de différence entre le comportement du système réel et celui de VOODB. Elles mettent également en évidence le fait que le nombre de *clusters* créés lors des parcours hiérarchiques et des parcours simples est identique. Les *clusters* générés sont légèrement plus gros dans le cas des parcours hiérarchiques (de profondeur 3), car ces derniers accèdent en moyenne à un nombre d'objets légèrement supérieur aux parcours simples (de profondeur 2).

	Banc d'essais	**Simulation**	**Rapport**
Nombre moyen de *clusters*	85,87	83,34	1,0304
Nombre moyen d'objets par *cluster*	13,07	11,40	1,1464

Table 7.12 : Comportement de DSTC
Petite base — *NC* = 10, *NO* = 1000 — Parcours hiérarchiques

	Banc d'essais	**Simulation**	**Rapport**
Nombre moyen de *clusters*	85,66	84,71	1,0112
Nombre moyen d'objets par *cluster*	9,33	9,32	1,0014

Table 7.13 : Comportement de DSTC
Petite base — *NC* = 10, *NO* = 1000 — Parcours simples

4.2.3. Base d'objets de taille moyenne

La Table 7.14 et la Table 7.15 (parcours hiérarchiques), ainsi que la Table 7.16 et la Table 7.17 (parcours simples), présentent les nombres moyens d'entrées-sorties et les temps moyens obtenus par mesures sur le système réel et en simulation, en ce qui concerne l'utilisation de la base avant et après regroupement.

Les gains constatés grâce au regroupement par DSTC sont du même ordre que ceux obtenus pour une petite base d'objets. Ceci était prévisible, étant donné que les bases de taille petite et moyenne tiennent en entier dans la mémoire centrale. Les gains réalisés le sont donc uniquement au chargement des objets. Les accès postérieurs s'effectuent uniquement en mémoire centrale. Les résultats de simulation apparaissent toujours

cohérents avec les performances constatées sur le système réel, excepté en ce qui concerne la surcharge due au regroupement.

	Banc d'essais	**Simulation**	**Rapport**
Utilisation avant regroupement	1890,70	1878,80	1,0063
Consolidation	3	0	—
Réorganisation	12796,60	354,50	36,0976
Surcharge due au regroupement	*12799,60*	*354,50*	*36,1060*
Utilisation après regroupement	330,60	350,50	0,9432
Gain	*5,71*	*5,36*	*1,0652*

Table 7.14 : Effets de DSTC sur les performances (nombres moyens d'E/S)
Base moyenne — *NC* = 50, *NO* = 20000 — Parcours hiérarchiques

	Banc d'essais	**Simulation**	**Rapport**
Utilisation avant regroupement	17,70	17,34	1,0207
Consolidation	19	0	—
Réorganisation	106,80	17,08	6,2510
Surcharge due au regroupement	*125,80*	*17,08*	7,3631
Utilisation après regroupement	3,30	3,35	0,9850
Gain	*5,36*	*5,17*	*1,0367*

Table 7.15 : Effets de DSTC sur les performances (temps moyens en ms)
Base moyenne — *NC* = 50, *NO* = 20000 — Parcours hiérarchiques

	Banc d'essais	**Simulation**	**Rapport**
Utilisation avant regroupement	1837,44	1947,60	0,9434
Consolidation	3	0	—
Réorganisation	12705,77	395,60	32,1177
Surcharge due au regroupement	*12708,77*	*395,60*	32,1253
Utilisation après regroupement	313,11	314,20	0,9965
Gain	*5,94*	*6,19*	*0,9596*

Table 7.16 : Effets de DSTC sur les performances (nombres moyens d'E/S)
Base moyenne — *NC* = 50, *NO* = 20000 — Parcours simples

	Banc d'essais	**Simulation**	**Rapport**
Utilisation avant regroupement	15,66	17,54	0,8931
Consolidation	20,11	0	—
Réorganisation	104,22	15,24	6,8387
Surcharge due au regroupement	*124,33*	*15,24*	8,1583
Utilisation après regroupement	3,00	2,84	1,0563
Gain	*5,22*	*6,17*	*0,8460*

Table 7.17 : Effets de DSTC sur les performances (temps moyens en ms)
Base moyenne — *NC* = 50, *NO* = 20000 — Parcours simples

Le nombre de *clusters* constitués en moyenne par la méthode DSTC ainsi que la taille moyenne de ces *clusters* (en nombre d'objets) sont présentés dans la Table 7.18 et la Table 7.19. Les résultats ne diffèrent pas de ceux obtenus avec une base de petite taille, car les

mêmes parcours ont été employés dans les deux cas. Le comportement de DSTC apparaît de nouveau similaire en simulation et dans le système réel.

	Banc d'essais	**Simulation**	**Rapport**
Nombre moyen de *clusters*	82,23	84,01	0,9788
Nombre moyen d'objets par *cluster*	12,83	13,73	0,9344

Table 7.18 : Comportement de DSTC
Base moyenne — *NC* = 50, *NO* = 20000 — Parcours hiérarchiques

	Banc d'essais	**Simulation**	**Rapport**
Nombre moyen de *clusters*	84,33	87,80	0,9605
Nombre moyen d'objets par *cluster*	11,20	10,20	1,0980

Table 7.19 : Comportement de DSTC
Base moyenne — *NC* = 50, *NO* = 20000 — Parcours simples

4.2.4. Base d'objets de « grande » taille

Étant dans l'impossibilité de mettre en œuvre le regroupement de bases d'objets qui soient réellement de grande taille avec DSTC, nous avons conservé la base introduite dans la Section 4.2.3 (dans sa version normale et après regroupement des objets). Par contre, nous avons diminué fortement la taille de la mémoire disponible sous Linux (de 64 Mo à 8 Mo), afin que la base soit effectivement grande par rapport à la mémoire vive disponible.

La Table 7.20 et la Table 7.21 (parcours hiérarchiques), ainsi que la

	Banc d'essais	**Simulation**	**Rapport**
Utilisation avant regroupement	12068,11	12245,53	0,9855
Utilisation après regroupement	401,33	433,90	0,9249
Gain	*30,07*	*28,22*	*1,0654*

Table 7.22 et la Table 7.23 (parcours stochastiques), présentent respectivement les temps moyens et les nombres d'entrées-sorties moyens obtenus par mesures sur le système réel et en simulation, en ce qui concerne l'utilisation de la base avant et après regroupement. Ces résultats font apparaître que le gain induit par le groupement des objets est beaucoup plus important lorsque la base de données ne peut pas être logée complètement en mémoire centrale (passage d'un facteur 5 à un facteur de l'ordre de 30). Les résultats de simulation apparaissent toujours cohérents avec les performances constatées sur le système réel.

	Banc d'essais	**Simulation**	**Rapport**
Utilisation avant regroupement	12504,60	12547,80	0,9965
Utilisation après regroupement	424,30	441,50	0,9610
Gain	*29,47*	*28,42*	*1,0369*

Table 7.20 : Effets de DSTC sur les performances (nombres moyens d'E/S)
« Grande » base – *NC* = 50, *NO* = 20000, Mémoire = 8 Mo — Parcours hiérarchiques

	Banc d'essais	Simulation	Rapport
Utilisation avant regroupement	102,10	103,70	0,9845
Utilisation après regroupement	2,90	3,15	0,9206
Gain	*35,20*	*32,92*	*1,0694*

Table 7.21 : Effets de DSTC sur les performances (temps moyens en ms)
« Grande » base – *NC* = 50, *NO* = 20000, Mémoire = 8 Mo — Parcours hiérarchiques

	Banc d'essais	Simulation	Rapport
Utilisation avant regroupement	12068,11	12245,53	0,9855
Utilisation après regroupement	401,33	433,90	0,9249
Gain	*30,07*	*28,22*	*1,0654*

Table 7.22 : Effets de DSTC sur les performances (nombres moyens d'E/S)
« Grande » base – *NC* = 50, *NO* = 20000, Mémoire = 8 Mo — Parcours simples

	Banc d'essais	Simulation	Rapport
Utilisation avant regroupement	103,11	102,60	1,0049
Utilisation après regroupement	2,66	3,09	0,8629
Gain	*38,66*	*33,20*	*1,1645*

Table 7.23 : Effets de DSTC sur les performances (temps moyens en ms)
« Grande » base – *NC* = 50, *NO* = 20000, Mémoire = 8 Mo — Parcours simples

5. Discussion

5.1. Performances de Texas et O_2

5.1.1. Avant propos

Il est délicat de comparer les performances de Texas et d'O_2, car les tests que nous avons présentés dans les Sections 2 et 3 n'ont pas été effectués dans les mêmes conditions. En effet, Texas n'est pas disponible sur matériel IBM et nous ne disposons pas d'une version d'O_2 pour Linux. Nos expériences ont donc été réalisées sur des machines différentes, de surcroît équipées de disques différents.

D'autre part, Texas et O_2 sont des systèmes très dissemblables dans leur philosophie et leurs fonctionnalités (cf. Chapitre 6). Alors qu'O_2 est un SGBDOO complet assurant l'intégrité des données et leur accès concurrent et sécurisé, Texas se positionne uniquement en tant qu'outil de persistance efficace pour le langage C++. C'est d'ailleurs un système mono-utilisateur et il ne met en œuvre que des mécanismes de reprise sur panne très simples. L'intérêt de comparer de façon détaillée ces deux systèmes est donc limité, d'autant que notre objectif initial est la validation de nos outils d'évaluation de performance.

5.1.2. Génération de la base

5.1.2.1. Temps de génération

Les deux systèmes étudiés ont des comportements légèrement différents lors de la génération de la base d'objets d'OCB. Alors que le temps de génération évolue de façon linéaire pour O_2, Texas semble plus sensible à l'augmentation de la taille de la base, ce qui est dû à la phase d'*unswizzling* de toutes les références inter-objets, qui s'ajoute à l'écriture des objets sur disque.

Néanmoins, la faisabilité de notre banc d'essais OCB est acquise, puisque sa base d'objets générée aléatoirement ne nécessite un temps de création prohibitif ni avec O_2, ni avec Texas. En effet, le cas le plus défavorable fait apparaître un temps de génération inférieur à une heure. De plus, une base d'objets donnée peut être sauvegardée pour des usages multiples, ce qui évite de la générer de nouveau à chaque fois.

Par ailleurs, il est intéressant de noter qu'avec Texas, le temps de génération est d'autant plus important que le schéma contient peu de classes, au contraire de ce qui se produit avec O_2. Ceci tient au fait que le temps de génération de la base est principalement influencé par deux phénomènes :

- la phase de vérification de la cohérence des graphes acycliques dans la génération du schéma (cf. Chapitre 3, Section 2.1) est d'autant plus complexe que le nombre de classes est faible. En effet, dans ces conditions, les références interclasses sont réparties sur un intervalle de classes réduit et forment des graphes très denses. La probabilité de cycles dans un graphe constitué de 5 classes, chacune possédant 10 références vers les autres, est par exemple plus importante que dans un graphe constitué de 20 classes, chacune possédant toujours 10 références vers les autres ;
- lorsque la base de données ne tient pas entièrement en mémoire centrale, le système doit recourir au *swap*, ce qui est coûteux en termes d'entrées-sorties et donc de temps ; ce *swap* est de toute façon plus coûteux que tout traitement en mémoire centrale.

La taille effective des bases générées avec Texas est inférieure à 60 Mo dans tous nos tests. Or, toutes ces bases tenaient en mémoire centrale (dont la taille est de 64 Mo). Par conséquent, c'est la phase de vérification de la cohérence des graphes acycliques qui a eu une importance prépondérante dans le temps de création de la base. Dans le cas d'O_2, c'est par contre le phénomène de *swap* qui était prépondérant. Nous avons vérifié cette hypothèse sur Texas en réduisant la taille mémoire vive disponible sous Linux à 16 Mo (en indiquant cette valeur en paramètre du noyau de Linux au démarrage de la machine). La Figure 7.26 présente le résultat de ces tests, qui confirment notre supposition.

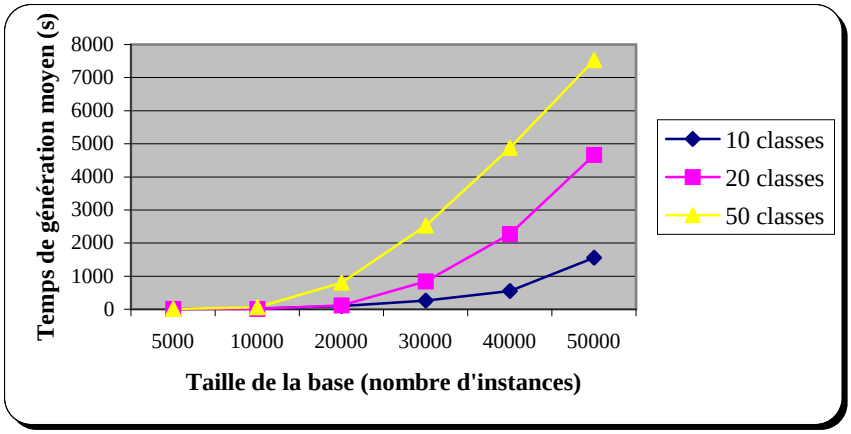

Figure 7.26 : Temps de génération en fonction de la taille de la base – Mémoire de 16 Mo (Texas)

5.1.2.2. Taille effective de la base

La base d'objets par défaut d'OCB (50 classes, 20000 instances) a une taille d'environ 30 Mo avec O_2 et d'environ 20 Mo avec Texas, ce qui peut être considérée comme une base de taille moyenne pour un banc d'essais. À titre de comparaison, la base qualifiée de « grande » dans OO1 a une taille de 40 Mo. Nous avons cependant montré que des bases de taille plus importante sont possibles avec OCB.

Les bases générées sous O_2 sont plus volumineuses que celles générées sous Texas (un tiers plus grosses, en moyenne). Ceci est dû au format de stockage des objets sur disque. Dans le cas de Texas, c'est le format mémoire qui est employé et directement écrit dans la base. Par contre, O_2 utilise les structures d'enregistrement de WiSS, qui sont plus élaborées.

5.1.3. Utilisation de la base

5.1.3.1. Variation de la base d'objets

Nous avons dans un premier temps vérifié pour chacun des systèmes étudiés que le nombre d'objets accédés en moyenne par chaque type de transaction restait stable quelle que soit la taille de la base et la valeur du paramètre *MAXNREF*. En supplément, il nous faut contrôler que le comportement stochastique d'OCB est bien similaire dans nos expériences sur Texas et O_2, dans les systèmes réels et en simulation. Pour cela, nous comparons le nombre moyen d'objets accédés par les transactions d'OCB. Les résultats concernant les types de transactions utilisés sont présentés dans la Table 7.24. Le rapport entre le nombre d'objets accédés avec Texas et avec O_2 (systèmes réels ou simulation) est comme prévu très proche de 1, étant donné que nous avons utilisé les mêmes transactions et les mêmes germes aléatoires dans toutes nos séries d'expériences. Ces résultats montrent qu'OCB permet bien des comparaisons fiables de différents systèmes.

	***MAXNREF* = 5**			***MAXNREF* = 10**		
	O_2	**Texas**	***Rapport***	**O_2**	**Texas**	***Rapport***
Accès ensemblistes (Banc d'essais)	28,920	28,895	***1,0008***	90,237	90,364	***0,9985***
Accès ensemblistes (Simulation)	28,000	28,558	***0,9804***	90,605	89,633	***1,0108***
Parcours simples (Banc d'essais)	28,932	28,914	***1,0006***	90,135	90,115	***1,0002***
Parcours simples (Simulation)	28,006	28,445	***0,9845***	90,663	89,673	***1,0110***
Parcours hiérarchiques (Banc d'essais)	7,645	7,638	***1,0008***	47,418	48,040	***0,9870***
Parcours hiérarchiques (Simulation)	7,262	7,279	***0,9977***	52,070	51,263	***1,0157***
Parcours stochastiques (Banc d'essais)	16,793	16,671	***1,0072***	11,242	11,290	***0,9957***
Parcours stochastiques (Simulation)	15,035	15,066	***0,9978***	11,181	10,545	***1,0603***

Table 7.24 : Comparaison du nombre moyen d'objets accédés

En ce qui concerne les performances, nous pouvons constater qu'O_2 et Texas ont un comportement très similaire. La corrélation entre le nombre d'entrées-sorties effectuées pour exécuter les transactions et le temps de réponse moyen du système apparaît également clairement dans les deux cas. Elle est due au fait que le gros des traitements, lors de l'exécution d'OCB, consiste à charger des objets depuis le disque.

Par ailleurs, il peut sembler illogique que les performances diminuent lorsque la taille de la base augmente, alors que le nombre d'objets accédés par les transactions, lui, ne varie pas. Cette évolution provient du fait que les objets accédés par les transactions sont distribués sur un nombre croissant de pages disque. Pour les parcourir, le système doit donc charger de plus en plus de pages.

Enfin, si les évaluations de performance obtenues par simulation diffèrent un peu des résultats mesurés sur les systèmes réels en valeur absolue, elles affichent clairement les mêmes tendances.

5.1.3.2. Variation de la taille du cache

Les résultats obtenus montrent que les performances d'O_2 se dégradent lorsque la taille de la base de données utilisée dépasse la taille de la mémoire centrale physique. Ceci était prévisible, une base d'objets dont la taille est supérieure à celle du cache forçant le système à mettre en œuvre une politique de remplacement des pages dans le cache et à les *swapper* sur disque.

Dans les mêmes conditions, les performances de Texas s'effondrent littéralement C'est le défaut de l'architecture de Texas, identifié dans le Chapitre 6 (Section 2.1.3). En effet, quelle que soit la taille de la partition de *swap*, et donc de la mémoire virtuelle globale utilisée par Texas, c'est toujours la taille de la mémoire centrale qui importe le

plus, toute opération de *swap* entraînant des entrées-sorties. Cette dégradation importante des performances est due à la politique de chargement des objets de Texas, qui provoque la réservation en mémoire centrale de nombreuses pages avant qu'elles soient effectivement chargées (cf. Chapitre 6, Section 2.1.3). Cette politique génère un *swap* d'autant plus important que la taille de la mémoire centrale est réduite.

Par ailleurs, les résultats obtenus par simulation aussi bien pour O_2 que pour Texas corroborent très bien les observations effectuées sur les systèmes réels.

5.2. Performances de Texas/DSTC

Nos expériences ont mis en évidence le fait que l'algorithme de groupement DSTC permet une amélioration significative des performances. Elles ont également confirmé un fait prévisible : les gains de performances obtenus grâce au regroupement des objets par DSTC sont beaucoup plus nets lorsque la taille de la base de données est supérieure à la taille de la mémoire disponible. En effet, plus la taille de la mémoire centrale est réduite, plus le système doit effectuer de remplacements de pages. Les pages qui ne sont pas utilisées séjournent donc normalement peu de temps en mémoire (le temps exact dépend de l'algorithme de remplacement de pages employé — LRU dans le cas de Linux). Un bon groupement des objets est donc d'autant plus utile dans ces conditions.

Cependant, nous avons également souligné que les regroupements étaient effectués au prix d'une surcharge importante pour le système. C'est d'ailleurs pourquoi la technique DSTC a justement été conçue pour que les traitements les plus lourds (réorganisation de la base) s'effectuent lorsque le système est inactif. Il faut également noter que le regroupement des objets n'est pas une opération régulière : un groupement donné peut être conservé durant plusieurs sessions d'utilisation de la base de données avant d'être remis en cause. Il est d'ailleurs important de déterminer la période après laquelle le regroupement devient rentable, c'est-à-dire lorsque la surcharge engendrée devient inférieure aux gains de performance induits.

Notre étude a finalement confirmé que DSTC se comporte bien de la façon attendue lors de la génération des *clusters*. Nous avons en effet effectué des séries de 100 transactions qui ont accédé à une dizaine d'objets chacune, en moyenne. Le nombre moyen de clusters *créés* est de l'ordre de 80 et le nombre moyen d'objets par cluster légèrement supérieur à 10. Ceci provient du fait que certains parcours ont des objets en communs et que ces objets sont par conséquent regroupés ensemble, faisant diminuer le nombre global de *clusters* et augmenter leur taille moyenne.

5.3. Bilan

Nous avons dans ce chapitre illustré la façon dont les performances de deux systèmes différents pouvaient être évaluées grâce à notre banc d'essais OCB. Nous avons à la fois démontré sa faisabilité, puisque la base d'objets d'OCB est générée (aléatoirement) dans un temps raisonnable, ainsi que sa pertinence, puisque nous avons pu confirmer le fait que les performances de Texas se dégradent fortement lorsque la taille de la base de données dépasse la taille de la mémoire vive disponible. Nous avons également montré qu'O_2 se montrait plus robuste que Texas dans ces conditions.

Nous avons aussi souligné les limites de l'approche par banc d'essais. Puisque nos expériences ont porté sur des systèmes implantés dans des environnements différents, il était très délicat de comparer les performances brutes (c'est-à-dire le temps de réponse, qui est le seul important du point de vue de l'utilisateur) de Texas et d'O_2. Ce problème peut être résolu grâce à notre modèle de simulation VOODB, où les caractéristiques d'un système ou d'un disque dur sont des paramètres qui peuvent prendre des valeurs identiques pour des simulations de systèmes différents.

Nous avons également présenté plusieurs expériences qui nous ont permis d'établir que notre modèle de simulation VOODB est un outil valide pour l'évaluation des performances des SGBDOO et plus particulièrement des méthodes dynamiques de regroupement d'objets au sein de ces systèmes. En effet, nous sommes capables, avec VOODB, de retrouver par simulation des résultats concernant les performances réelles de deux systèmes sensiblement différents l'un de l'autre (O_2 et Texas), ainsi que de simuler les effets de l'algorithme de regroupement d'objets DSTC sur les performances de Texas.

Sur la base de ces résultats préliminaires, nous pensons que notre modèle de simulation VOODB peut fournir des indications de performance quasiment aussi fiables que celles d'un banc d'essais, pour un coût moindre (il n'est pas nécessaire de disposer du système testé). Notre modèle de simulation est de plus un outil souple qui permet de simuler des systèmes existants très divers, mais aussi des prototypes non encore implémentés, sur la base de leurs spécifications. VOODB permet également de comparer entre elles différentes politiques de gestion (notamment des stratégies de regroupement des objets) au sein d'un même SGBDOO afin de déterminer quelle est la meilleure dans des conditions données.

Des campagnes de simulations intensives sont possibles pour des SGBDOO et des applications variés, existants ou futurs, prenant en compte le regroupement d'objets ou non.

Conclusion

1. Bilan et contributions

Nous nous sommes attachés dans cette thèse à proposer des outils génériques et valides pour l'évaluation des performances des SGBDOO en général et des techniques de regroupement d'objets dans les SGBDOO en particulier. Ces outils, le banc d'essais OCB et le modèle de simulation VOODB, sont génériques dans la mesure où ils permettent d'effectuer des études de performance sur n'importe quel type de SGBDOO. De plus, en les particularisant, nous sommes à même de traiter plus spécifiquement les problèmes de groupement d'objets qui nous intéressent au premier chef. Nous avons également été amenés à concevoir un petit simulateur baptisé DESP-C++, dont les principales caractéristiques sont la simplicité et la rapidité d'exécution, afin de mettre en œuvre notre modèle de simulation VOODB.

Notre travail a été guidé par une étude préalable des méthodes existantes dans le domaine de l'évaluation de performance en matière d'analyse mathématique, de bancs d'essais et de simulation. Chacune de ces techniques a été évaluée selon les critères de pertinence, fiabilité, faisabilité, indépendance. Notre analyse a mis en évidence qu'aucune des méthodes étudiées ne satisfaisait ces quatre critères à la fois. Notre contribution s'appuie donc sur cette constatation. Elle entend par ailleurs exploiter les avantages de la simulation aléatoire à événements discrets (faible coût, modélisation *a priori*, comparaison de systèmes ou d'algorithmes normalement implantés dans des environnements différents dans les mêmes conditions). Le travail effectué porte essentiellement sur :

- la *généricité* de nos solutions, qui ne devaient pas être ciblées *a priori* sur un problème donné, afin de pouvoir ultérieurement être réutilisées et/ou étendues dans un autre but que l'évaluation des performances d'algorithmes de groupement d'objets ;
- l'*adaptabilité* de nos outils, qui devaient pouvoir être spécialisés (ou instanciés, au sens « objet » du terme) grâce à un ensemble de paramètres, afin de traiter un problème donné comme le groupement d'objets ;
- la *pertinence* de nos techniques d'évaluation en ce qui concerne leur capacité à mesurer les performances (par exemple, pour déceler les bons ou les mauvais groupements d'objets) ;

- la *fiabilité* de nos propositions, ce qui passe par un processus de validation et de vérification impliquant l'implantation de nos solutions et leur utilisation sur des systèmes existants (ce qui a permis de démontrer en même temps leur *faisabilité*).

Les principales contributions concernant notre banc d'essais OCB résident dans les points suivants.

- OCB est un banc d'essais générique, qui peut être paramétré afin d'évaluer les performances de systèmes et/ou d'applications très divers. OCB peut donc être adapté à des contextes très variés, selon les besoins de ses utilisateurs. Un paramétrage adéquat permet notamment de retrouver les bancs d'essais existants.
- OCB n'utilise pas de base d'objets figée. Il permet au contraire la génération de bases de données au schéma très riche en classes et en références diverses, ce qui permet de concevoir des hiérarchies de classes complexes et entrelacées.
- OCB a dans un premier temps été conçu pour évaluer les performances d'algorithmes de groupement d'objets. Aussi est-il bien adapté à ce genre d'études, contrairement aux bancs d'essais existants.

Notre approche par simulation présente, elle, les caractéristiques novatrices suivantes.

- Comme dans le cas du banc d'essais OCB, nous avons préféré concevoir un modèle de simulation générique plutôt qu'un modèle dédié, afin de ne pas cibler notre outil sur un type de SGBDOO et/ou une méthode d'optimisation donnés. Cette particularité permet de caractériser VOODB pour une étude précise, en fonction des besoins propres à cette étude.
- Nous proposons une méthodologie de modélisation qui permet, de manière systématique, d'analyser un système, de spécifier des modèles fiables pour ce système et d'envisager une automatisation de leur traduction dans des langages de programmation ou de simulation.
- Plusieurs modèles de simulation dédiés à l'étude d'une heuristique particulière au sein des SGBDOO font usage de parties de bancs d'essais (base d'objets ou transactions). Notre modèle de simulation VOODB intègre, lui, un banc d'essais complet (OCB), ce qui assure une plus grande cohérence et nous permet de disposer d'un outil d'évaluation plus fiable.

Pour terminer, les outils que nous proposons ont été validés :

- DESP-C++, en comparant les résultats de simulation fournis avec ceux procurés par le logiciel QNAP2, qui est, lui, un simulateur validé, pour différents modèles de simulation, et en constatant que ces résultats étaient cohérents ;
- OCB et VOODB, en mesurant les performances du SGBDOO O_2, du gestionnaire d'objets persistants Texas et de la méthode de regroupement d'objets DSTC avec le banc d'essais OCB et par simulation grâce au modèle VOODB. Les résultats

mesurés sur les systèmes réels et calculés lors des simulations étaient très similaires.

Les expériences concernant DSTC ont par ailleurs permis de mettre en évidence les circonstances dans lesquelles cet algorithme est en mesure de proposer un groupement efficace, ainsi que la difficulté de fixer les paramètres de seuil (Tf_a et Tf_c) de la méthode.

2. Perspectives de recherche

Le travail réalisé dans cette thèse ouvre diverses perspectives de recherche, tant au niveau de la conception du banc d'essais OCB, de celle du modèle de simulation VOODB, que de leur exploitation dans le but de comparer différents SGBDOO et/ou méthodes de regroupement d'objets, existantes ou non.

Deux aspects concernant le banc d'essais OCB peuvent notamment faire l'objet de travaux plus approfondis.

- Nous avons seulement exposé les principes d'une version multi-utilisateurs d'OCB. Or, le passage d'une version mono-utilisateur d'un banc d'essais à une version multi-utilisateurs opérationnelle n'est pas triviale et nécessite une attention particulière. L'intérêt de cette démarche serait d'évaluer l'efficacité en terme de performance du contrôle de concurrence et le comportement des systèmes face à une charge plus importante et plus hétérogène. Le besoin en bancs d'essais multi-utilisateurs est réel, car les SGBDOO opèrent normalement en environnement concurrent et leurs performances ne sauraient être complètement testées en mode mono-utilisateur uniquement.
- OCB est actuellement conçu pour évaluer les performances brutes d'un système donné. Nous n'avons pas abordé l'aspect « qualitatif » qu'un banc d'essais complet devrait proposer afin d'apprécier l'étendue et l'efficacité des fonctionnalités offertes par un SGBDOO. En effet, ces éléments sont aussi déterminants dans le choix d'un système que les performances en elles-mêmes.

Notre modèle de simulation VOODB est également susceptible d'être amélioré et étendu.

- La base de composants logiciels de VOODB pourrait être enrichie d'éléments influençant les performances. Par exemple, elle ne comprend que des stratégies élémentaires de remplacement en mémoire cache (RANDOM, FIFO, LFU, LRU-K, CLOCK…) alors que des stratégies plus élaborées, du domaine de la recherche, pourraient tirer meilleur parti du regroupement des objets, par exemple. De même, des stratégies de préchargement des objets pourraient être intégrées à la base de composants logiciels, ainsi que des algorithmes de groupement d'objets plus

nombreux. Cela pose le problème de la gestion de cet ensemble de composants et de leur intégration dans une instance du modèle de simulation.

- VOODB pourrait être également étendu afin de prendre en compte des éléments différents de ceux que nous avons retenus en priorité, mais qui influencent aussi les performances des SGDBOO (comme le contrôle de concurrence et l'optimisation des requêtes). Une simulation plus fine des accès au disque pourrait aussi être envisagée, afin de prendre en compte la gestion de caches physiques de façon plus explicite ou les progrès réalisés dans la technologie des disques parallèles. Le couplage de VOODB avec un environnement de simulation de disque existant, comme *DiskSim* [Ganger et Worthington 98], pourrait également constituer une alternative valable.
- Afin d'être réutilisable facilement par la communauté scientifique, VOODB pourrait être reconstruit comme partie d'une librairie de modèles réutilisables, sous la forme de fragments modulaires pouvant être employés pour assembler des modèles plus gros. Or, tronçonner le modèle en fragments ne suffit pas à le rendre réutilisable. Il faut également standardiser la structure et l'interface de chaque module et fournir une documentation claire pour tous les sous-modèles [Breunese et al. 98].

Pour terminer, nos outils d'évaluation des performances s'appliquent exclusivement aux SGBDOO, qui ont démontré des limitations importantes hors de leurs domaines de prédilection (les applications d'ingénierie : CAO, AGL, etc.). Des systèmes dont l'objectif est d'allier les performances des SGBD relationnels et la puissance du modèle objet, les systèmes relationnels-objets, ont fait leur apparition pour pallier ce manque. Peu d'outils d'évaluation des performances de ces systèmes existent actuellement (un seul banc d'essais à notre connaissance : BUCKY [Carey et al. 97]). Aussi est-il envisageable d'adapter nos outils (banc d'essais OCB et modèle de simulation VOODB) au contexte relationnel-objet.

Références bibliographiques

[Adiba et Collet 93] M. Adiba, C. Collet, *Objets et bases de données: le SGBD O_2*, Traité des Nouvelles Technologies, Hermès, Paris, 1993

[AESOP 95], AESOP GmbH, *SIMPLE++ Reference Manual*, 1995

[Agrawal et Gehani 89] R. Agrawal, N.H. Gehani, *ODE (Object Database and Environment): The Language and the Data Model*, International Conference on Management of Data, ACM SIGMOD '89, Portland, Oregon, May-June 1989, pp. 36-45

[Amiel et al. 92] E. Amiel, M.-J. Bellosta-Tourtier, P. Valduriez, F. Viallet, *Étude de la Persistance dans les SGBDOO*, Rapport de Recherche INRIA No. 1592, février 1992

[Amsaleg 95] L. Amsaleg, *Conception et réalisation d'un glaneur de cellules adapté aux SGBDO client-serveur*, Thèse de Doctorat d'Université, Université Paris VI, juin 1995

[Anderson et al. 90] T.L. Anderson, A.J. Berre, M. Mallison, H.H. Porter III, B. Scheider, *The HyperModel Benchmark*, International Conference on Extending Database Technology, Venice, Italy, March 1990, pp. 317-331

[Andrews et al. 91] T. Andrews, C. Harris, K. Sinkel, *ONTOS: A Persistent Database for C++*, In "Object-Oriented Databases with Applications to CASE, Networks and VLSI CAD", Edited by R. Gupta and E. Horowitz, Prentice Hall Series in Data and Knowledge Base Systems, 1991, pp. 387-406

[Andrews et Harris 87] T. Andrews, C. Harris, *Combining Language and Database Advances in an Object-Oriented Development Environment*, International Conference on OOPSLA, Orlando, Florida, October 1987; ACM SIGPLAN, 1987, pp. 430-440

[Ardent Software 98] Ardent Software, *Site Web*, http://www.ardentsoftware.com/object/product/, 1998

[Atkinson et al. 89] M. Atkinson, F. Bancilhon, D. DeWitt, K. Dittrich, D. Maier, and S. Zdonik, *The Objected-Oriented Database System Manifesto*, 1st International Conference on Deductive and Object-Oriented Database, DOOD'89, Kyoto, Japan, December 1989

[Atkinson et al. 92] M.P. Atkinson, A. Birnie, N. Jackson, P.C. Philbrow, *Measuring Persistent Object Systems*, 5th International Workshop on Persistent Object Systems, San Miniato (Pisa), Italy, September 1992, pp. 63-85

[Atkinson et Maier 90] M.P. Atkinson, D. Maier, *Perspectives on persistent object systems*, 4th International Workshop on Persistent Object Systems (Concluding remarks on Workshop), Martha's Vineyard, USA, September 1990, pp. 425-426

[Baker 97] S. Baker, *CORBA Distributed Objects, using Orbix*, ACM Press, Addison-Wesley, 1997

[Balci et Nance 92] O. Balci, R.E. Nance, *The simulation model development environment: an overview*, 1992 Winter Simulation Conference, 1992, pp. 726-736

[Bancilhon 88] F. Bancilhon, *Object-Oriented Database Systems*, ACM International Symposium on OPDS, Austin, Texas, March 1988, pp. 152-162

[Bancilhon et al. 88] F. Bancilhon, G. Bardebette, V. Benzaken, C. Delobel, S. Gamerman, C. Lécluse, P. Pfeffer, P. Richard, F. Velez, *The Design and Implementation of O_2, an Object-Oriented Database System*, 2nd International Workshop on Object-Oriented Database Systems, Bad Münster am Stein-Ebernburg, FRG, September 1988, pp. 1-22

[Bancilhon et al. 92] F. Bancilhon, C. Delobel, P. Kanellakis (Eds.), *Building an Object-Oriented Database System: The Story of O_2,* Morgan Kaufmann Publishers, 1992

[Banerjee et al. 87] J. Banerjee, H.-T. Chou, J.F. Garza, W. Kim, D. Woelk, N. Ballou, H.-J. Kim, *Data Model Issues for Object-Oriented Applications*, ACM Transaction on Office Information Systems, Vol. 5, No. 1, January 1987, pp. 3-26

[Banerjee et al. 88] J. Banerjee, W. Kim, H.-J. Kim, J.F. Garza, *Clustering a DAG for CAD Databases*, IEEE Transactions on Software Engineering, Vol. 14, No. 11, November 1988

[Banerjee et Gardner 95] S. Banerjee, C. Gardner, *Towards An Improved Evaluation Metric For Object Database Management Systems*, OOPSLA '95 Workshop on Object Database Behavior, Benchmarks and Performance, Austin, Texas, October 1995

[Bates et al. 95] C. Bates, I. Jelly, I. Lalis, P. Menhart, *Simulating transaction processing in parallel database systems*, 7th European Simulation Symposium, ESS '95, Erlanger-Nuremberg, Germany, October 1995, pp. 193-197

[Beech et Özbütün 91] D. Beech, C. Özbütün, *Object databases as generalisations of relational databases*, Computer Standards & Interfaces, No. 13, 1991, pp. 221-230

[Bellatreche et al. 97] L. Bellatreche, K. Karpalem, A. Simonet, *Horizontal class partitioning in object oriented databases*, 8th International Conference on Database Expert Systems Applications, DEXA '97, Toulouse, France, September 1997; LNCS Vol. 1308 (Springer), pp. 1997

[Bellatreche et al. 98] L. Bellatreche, K. Karpalem, A. Simonet, *Query Optimization using Horizontal Class Partitioning in Object Oriented Databases*, XVIème Congrès Informatique des Organisations et Systèmes d'Information et de Décision, INFORSID, Montpellier, France, mai 1998

[Benzaken 90a] V. Benzaken, *Regroupement d'objets sur disque dans un système de Bases de Données Orienté-Objet*, Thèse de Doctorat d'Université, Université de Paris-sud, janvier 1990

[Benzaken 90b] V. Benzaken, *An Evaluation Model for Clustering Strategies in the O_2 Object-Oriented Database System,* 3rd International Conference on Database Theory, Paris, France, December 1990, pp. 126-140

[Benzaken et al. 88] V. Benzaken, C. Delobel, J.B. Ndala, *Gestionnaires de mémoires et d'objets*, 4èmes Journées Bases de Données Avancées, BDA 88, Benodet, 1988, pp. 233-266

[Benzaken et Delobel 90] V. Benzaken, C. Delobel, *Enhancing Performance in a Persistent Object Store: Clustering Strategies in O_2,* 4th International Workshop on Persistent Object Systems, September 1990, pp. 403-412.

[Berre et Anderson 91] A.J. Berre, T.L. Anderson, *The HyperModel Benchmark for Evaluating Object-Oriented Databases,* In "Object-Oriented Databases with Applications to CASE, Networks and VLSI CAD", Edited by R. Gupta and E. Horowitz, Prentice Hall Series in Data and Knowledge Base Systems, 1991, pp. 75-91

[Berry et Linoff 97] M.J.A. Berry, G. Linoff, *Data Mining : Techniques appliquées au marketing, à la vente et aux services clients*, InterÉditions, 1997

[Bertino et al. 94] E. Bertino, A.A. Saad, M.A. Ismail, *Clustering techniques in object bases*, Data and Knowledge Engineering, Vol. 12, No. 3, 1994, pp. 255-275

[Bertino et Martino 91] E. Bertino, L. Martino, *Object-Oriented Database Management Systems: Concepts and Issues*, IEEE Computer, April 1991, pp. 33-47

[Bonner et al. 95] A.J. Bonner, A. Shrufi, S. Rozen, *Benchmarking Object-Oriented DBMSs for Workflow Management*, OOPSLA '95 Workshop on Object Database Behavior, Benchmarks and Performance, Austin, Texas, October 1995

[Bouzeghoub et al. 94] M. Bouzeghoub, G. Gardarin, P. Valduriez, *Du C++ à Merise Objet : Objets*, Eyrolles, Paris, 1994

[Breugnot et al. 90] D. Breugnot, M. Gourgand, P. Kellert, *SIGMA: An Intelligent and Graphical System for the Modelling of Assembly Systems*, Proceedings of the European Simulation Symposium, Ghent, Belgium, November 1990, pp. 225-230

[Breunese et al. 98] A.P.J. Breunese, J.L. Top, J.F. Broenink, J.M. Akkermans, *Libraries of Reusable Models: Theory and Application*, Simulation, Vol. 41, No. 1, July 1998, pp. 7-22

[Bryan 89] O.F. Bryan Jr., *MODSIM II – An Object-Oriented Simulation Language for Sequential and Parallel Processors*, 1989 Winter Simulation Conference, Piscataway, NJ, 1989, pp. 172-177

[Bullat 95] F. Bullat, *Regroupement physique d'objets dans les bases de données*, Ingénierie des Systèmes d'Information, Vol. 3, No. 5, octobre 1995, pp. 607-638

[Bullat 96] F. Bullat, *Regroupement dynamique d'objets dans les bases de données*, Thèse de Doctorat d'Université, Université Blaise Pascal, Clermont-Ferrand II, janvier 1996

[Bullat et Schneider 96] F. Bullat, M. Schneider, *Dynamic Clustering in Object Database Exploiting Effective Use of Relationships Between Objects*, ECOOP '96, Linz, Austria, July 1996; LNCS Vol. 1098, pp. 344-365

[Butler 87] M.H. Butler, *Storage Reclamation in Object Oriented Database Systems*, International Conference on Management of Data, ACM SIGMOD '87, San Francisco, California, May 1987, pp. 410-425

[Butterworth et al. 91] P. Butterworth, A. Otis, J. Stein, *The Gemstone Object Database Management System*, Communications of the ACM, Vol. 34, No. 10, October 1991, pp. 64-77

[Carey et al. 88] M.J. Carey et al., *Storage management for objects in EXODUS*, In "Object-Oriented concepts, Databases and Applications", Edited by W. Kim and F.H. Lochovsky, ACM Press/Addisson Wesley, New York, 1988, pp. 341-369

[Carey et al. 93] M.J. Carey, D.J. Dewitt, J.F. Naughton, *The OO7 Benchmark*, ACM SIGMOD International Conference on Management of Data, Washington DC, May 1993, pp. 12-21

[Carey et al. 94a] M.J. Carey, M.J. Franklin, M. Livny, E.J. Shekita, *Fine-Grained Sharing in a Page Server OODBMS*, Technical Report CS-TR-94-1224, University of Wisconsin-Madison, April 1994

[Carey et al. 94b] M.J. Carey, D.J. Dewitt, C. Kant, J.F. Naughton, *A Status Report on the OO7 OODBMS Benchmarking Effort*, 9th Annual ACM Conference on Object-Oriented Programming Systems, Languages and Applications, OOPSLA '94, Portland, Oregon, USA, October 1994; SIGPLAN Notices Vol. 29, No. 10, pp. 414-426

[Carey et al. 97] M.J. Carey, D.J. DeWitt, J.F. Naughton, *The BUCKY Object-Relational Benchmark*, ACM SIGMOD International Conference on Management of Data, Tucson, Arizona, May 1997, pp. 135-146

[Cattell 91] R.G.G. Cattell, *An Engineering Database Benchmark*, In "The Benchmark Handbook for Database Transaction Processing Systems", Edited by Jim Gray, Morgan Kaufmann Publishers, 1991, pp. 247-281

[Cattell et Skeen 92] R.G.G. Cattell, J. Skeen, *Object Operations Benchmark*, ACM Transactions on Database Systems, Vol. 17, No. 1, March 1992, pp. 1-31

[Chabridon et al. 92] S. Chabridon, J.-C. Liao, Y. Ma, L. Gruenwald, *Storage Management Techniques for Object-Oriented Database Systems*, Technical Report, University of Oklahoma, School of Computer Science, December 1992

[Chabridon et al. 93] S. Chabridon, J.-C. Liao, Y. Ma, L. Gruenwald, *Clustering Techniques for Object-Oriented Database Systems*, 38th IEEE Computer Society International Conference, San Francisco, February 1993, pp. 232-242

[Chang 89] E.E. Chang, *Effective Clustering and Buffering in an Object-Oriented DBMS*, Ph.D. Dissertation in Computer Science, University of California, Berkeley, Computer Science Division (EECS), Technical Report No. UCB/CSD 89/515, June 1989

[Chang et Katz 89] E.E. Chang, R.H. Katz, *Exploiting Inheritance and Structure Semantics for Effective Clustering and Buffering in an Object-Oriented DBMS*, ACM SIGMOD International Conference on Management of Data, Portland, Oregon, June 1989, pp. 348-357

[Chang et Katz 90] E.E. Chang, R.H. Katz, *Inheritance in computer-aided design databases: semantics and implementation issues*, CAD, Vol. 22, No. 8, October 1990, pp. 489-499

[Chaudhri 95] A.B. Chaudhri, *An Annotated Bibliography of Benchmarks for Object Databases*, SIGMOD Record, Vol. 24, No. 1, March 1995, pp. 50-57

[Cheiney et Lanzelotte 93] J.P. Cheiney, R.S.G. Lanzelotte, *Vers une nouvelle génération d'optimiseurs pour les SGBD orientés objet*, Technique et Science Informatiques, Vol. 12, No. 4, 1993, pp. 433-461

[Cheng et Hurson 91] J.R. Cheng, A.R. Hurson, *Effective clustering of complex objects in object-oriented databases*, ACM SIGMOD International Conference on Management of Data, Denver, Colorado, May 1991, pp. 22-31

[Chou et al. 85] H.-T. Chou et al., *Design and implementation of the Wisconsin storage system*, Software Practice and Experience, Vol. 15, No. 10, October 1985

[Copeland et al. 90] G. Copeland, M. Franklin, G. Weikum, *Uniform Object Management*, International Conference on Extending Database Technology, Venice, Italy, March 1990, pp. 253-268

[Copeland et Maier 84] G. Copeland, D. Maier, *Making Smalltalk a database system*, International Conference on Management of Data, ACM SIGMOD '84, June 1984

[CORE 88] CORE International, *CORE Disk Performance Test Program Version 2.8 User Manual*, 1988

[Dahl et Nygaard 66] O.J. Dahl, K. Nygaard, *SIMULA, an algol based simulation language*, Communications of ACM, Vol. 9, No. 9, 1966

[Darmont et al. 95] J. Darmont, A. Attoui, M. Gourgand, *Performance Evaluation for Clustering Algorithms in Object-Oriented Database Systems*, 6th International Conference on Database and Expert Systems Applications, DEXA '95, London, UK, September 1995; LNCS Vol. 978 (Springer), pp. 187-196

[Darmont et al. 97] J. Darmont, A. Attoui, M. Gourgand, *Simulation of clustering algorithms in OODBs in order to evaluate their performances*, Simulation Practice and Theory, Vol. 5, No. 3, 1997, pp. 269-287

[Darmont et al. 98] J. Darmont, B. Petit, M. Schneider, *OCB: A Generic Benchmark to Evaluate the Performances of Object-Oriented Database Systems*, 6th International Conference on Extending Database Technology, EDBT '98, Valencia, Spain, March 1998; LNCS Vol. 1377 (Springer), pp. 326-340

[Darmont et Gruenwald 96a] J. Darmont, L. Gruenwald, *A Comparison Study of Object-Oriented Database Clustering Techniques*, ASME Engineering Systems Design and Analysis Conference, ESDA '96, Montpellier, France, July 1996

[Darmont et Gruenwald 96b] J. Darmont, L. Gruenwald, *A Comparison Study of Clustering Techniques for Object-Oriented Databases*, Information Sciences, Vol. 94, No. 1-4, December 1996, pp. 55-86

[De Rosnay 75] J. De Rosnay, *Le macroscope*, Le Seuil, 1975

[Delobel et al. 91] C. Delobel, C. Lécluse, P. Richard, *Bases de données : des systèmes relationnels aux systèmes à objets*, InterÉditions, Paris, 1991

[Deux et al. 90] O. Deux et al., *The Story of O_2*, IEEE Transactions on Knowledge and Data Engineering, Vol. 2, No. 1, March 1990, pp. 91-108

[Deux et al. 91] O. Deux et al., *The O_2 System*, Communications of the ACM, Vol. 34, No. 10, October 1991, pp. 34-48

[DeWitt et al. 90] D.J. DeWitt, P. Futtersack, D. Maier, F. Vélez, *A Study of three Alternative Workstation-Server Architectures for Object-Oriented Database Systems*, 16th International Conference on Very Large Data Bases, VLDB '90, Brisbane, Australia, August 1990, pp. 107-121

[Drew et al. 90] P. Drew, R. King, S. Hudson, *The Performance and Utility of the Cactis Implementation Algorithms*, 16th International Conference on Very Large Data Bases, Brisbane, Australia, August 1990, pp. 135-147

[Effelsberg et Haerder 84] W. Effelsberg, T. Haerder, *Principles of Database Buffer Management*, ACM Transactions on Database Systems, Vol. 9, No. 4, December 1984, pp. 560-595

[Eliot et Moss 90] J. Eliot, J.B. Moss, *Design of the Mneme persistent object store*, ACM Transactions on Information Systems, Vol. 8, April 1990, pp. 103-139

[Eliot et Moss 92] J. Eliot, J.B. Moss, *Working with Persistent Objects: To Swizzle or Not to Swizzle*, IEEE Transactions on Software Engineering, Vol. 18, No. 8, August 1992, pp. 657-673

[Emmerich et Kampmann 93] W. Emmerich, M. Kampmann, *The Merlin OMS benchmark: definition, implementation and results*, Internal Memo No. 65, Department of Computer Science, University of Dortmund, Germany, October 1992, revised July 1993

[Fishwick 92] P.A. Fishwick, *Simpack: Getting started with simulation programming in C and C++*, Technical Report #TR92-022, Computer and Information Sciences, University of Florida, February 1992

[Fleury et al. 98] G. Fleury, M. Gourgand, P. Lacomme, N. Tchernev, *Une méthode basée sur l'approche multi-agents pour la modélisation des systèmes industriels de production*, À paraître dans la revue TSI, 1998

[Ford et al. 88] S. Ford, J. Joseph, D.E. Langworthy, D.F. Lively, G. Pathak, E.R. Perez, R.W. Peterson, D.M. Sparacin, S.M. Thatte, D.L. Wells, S. Agarwala, *ZEITGEIST: Database Support for Object-Oriented Programming,* 2nd International Workshop on Object-Oriented Database Systems, Bad Münster am Stein-Ebernburg, FRG, September 1988, pp. 23-42

[Ganger et Worthington 98] G.R. Ganger, B.L. Worthington, *The DiskSim Simulation Environment Version 1.0 Reference Manual*, Technical Report No. CSE-TR-358-98, Computer Science and Engineering Division, Department of Electrical Engineering and Computer Science, The University of Michigan, February 1998

[Gardarin et al. 95] G. Gardarin, J.-R. Gruser, Z.-H. Tang, *A Cost Model for Clustered Object-Oriented Databases,* 21st International Conference on Very Large Data Bases, VLDB '95, Zurich, Switzerland, September 1995, pp. 323-334

[Garey et Johnson 79] M.R. Garey, D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H. Freeman and Company, 1979

[Gatziu et al. 91] S. Gatziu, A. Geppert, K.R. Dittrich, *Integrating Active Mecanisms into an Object-Oriented Database System*, 3rd International Workshop on Database Programming Languages (DBPL), Nafplio, Greece, August 1991

[Gay 96] J.-Y. Gay, *A Clustering Technique for Object-Oriented Database Management Systems,* Mémoire d'Ingénieur en Génie Informatique, ISI-CUST, Université Blaise Pascal, Clermont-Ferrand II, juin 1996

[Gay et Gruenwald 97] J.-Y. Gay, L. Gruenwald, *A Clustering Technique for Object Oriented Databases*, 8th International Conference on Database and Expert Systems Applications, DEXA '97, Toulouse, France, September 1997; L.N.C.S. Vol. 1308 (Springer), pp. 81-90

[Geppert et al. 94] A. Geppert, S. Gatziu, K.R. Dittrich, *Performance evaluation of an active database management system: OO7 meets the BEAST*, Technical Report No. IFI-94-18, Computer Science Department, University of Zurich, Switzerland, November 1994

[Gerlhof et al. 93] C. Gerlhof, A. Kemper, C. Kilger, G. Moerkotte, *Partition-Based Clustering in Object Bases: From Theory to Practice,* 4th International Conference on Foundation of Data Organization and Algorithms, FODO '93, Chicago, Illinois, 1993

[Gerlhof et al. 96] C. Gerlhof, A. Kemper, C. Kilger, G. Moerkotte, *On the Cost of Monitoring and Reorganization of Object Bases for Clustering,* SIGMOD Record, Vol. 25, No. 3, September 1996

[Goldberg et Robson 81] A. Goldberg, D. Robson, *The Smalltalk-80 System*, Byte Magazine, Vol. 6, No. 8, August 1981, pp. 36-48

[Gourgand 84] M. Gourgand, *Outils logiciels pour l'évaluation des performances des systèmes informatiques*, Thèse d'État, Université Blaise Pascal, Clermont-Ferrand II, juin 1984

[Gourgand et Kellert 92] M. Gourgand, P. Kellert, *An object-oriented methodology for manufacturing systems modelling*, 1992 Summer Computer Simulation Conference (SCSC), Reno, Nevada, 1992, pp. 1123-1128

[Gourgand et Kellert 93] M. Gourgand, P. Kellert, *Approche transaction et approche station*, Rapport Interne n° 4, Laboratoire d'Informatique, Université Blaise Pascal, Clermont-Ferrand II, décembre 1993

[Gourhant et al. 92] Y. Gourhant, S. Louboutin, V. Cahill, A. Condon, G. Starovic, B. Tangney, *Dynamic Clustering in an Object-Oriented Distributed System*, OLDA II Workshop on Objects in Large Distributed Applications, October 1992; OOPSLA '92 proceedings

[Grandi et Scalas 93] F. Grandi, M.R. Scalas, *Block Access Estimation for Clustered Data Using a Finite LRU Buffer*, IEEE Transactions on Software Engineering, Vol. 19, No. 7, July 1993, pp. 641-660

[Gray 93] J. Gray (Ed.), *The Benchmark Handbook for Database and Transaction Processing Systems*, 2nd edition, Morgan Kaufmann Publishers Inc., 1993

[Gruber 92] O. Gruber, *EOS, an Environment for Persistent and Distributed Applications over a Shared Object Space*, Thèse de Doctorat d'Université, Université Pierre et Marie Curie, Paris VI, décembre 1992

[Gruber et al. 92] O. Gruber, L. Amsaleg, L. Daynès, P. Valduriez, *EOS, an Environment for Object-Based Systems*, 25th Hawaii International Conference on System Sciences, Vol. 1, January 1992, pp. 757-768

[Gruber et Amsaleg 92] O. Gruber, L. Amsaleg, *Object Grouping in EOS*, IWDOM '92 Workshop on Distributed Object Management, University of Alberta, Canada, August 1992, pp. 117-131

[Halloran et Roth 93] T.J. Halloran, M.A. Roth, *"Magic mirror on the wall, who's the fastest database of them all?": a survey of database benchmarks*, Technical Report No. AFIT/EN-TR-93-05, US Air Force Institute of Technology (AETC), June 1993

[Harrus et al. 90] G. Harrus, V. Benzaken, C. Delobel, *Measuring performance of clustering strategies: the CluB-0 benchmark*, Technical Report, Altaïr, 1990

[He et al. 93] M. He, A.R. Hurson, L.L. Miller, D. Sheth, *An Efficient Storage Protocol for Distributed Object-Oriented Databases*, IEEE Parallel and Distributed Processing, 1993, pp. 606-610

[Healy et Kilgore 97] K.J. Healy, R.A. Kilgore, *Silk: A Java-based Process Simulation Language*, 1997 Winter Simulation Conference, Atlanta, GA, 1997, pp. 475-482

[Herscovitch et Schneider 65] H. Herscovitch, T.H. Schneider, *GPSS II – An extended general purpose simulator*, IBM System Journal, Vol. 4, No. 3, 1965

[Hill 93] D.R.C. Hill, *Enhancing the QNAP2 object-oriented simulation language*, Modeling and Simulation, ESM '93, Lyon, France, 1993, pp. 171-175

[Hill et al. 93] D.R.C. Hill, M. Gourgand, P. Kellert, *Object-oriented tools for building animations environments of simulation results*, 26th IEEE/SCS Annual Simulation Symposium, March-April 1993, pp. 237-246

[Hornick et Zdonik 87] M.F. Hornick, S.B. Zdonik, *A Shared Segmented Memory System for an Object Oriented Database*, ACM Transactions on Office Information Systems, Vol. 5, No. 1, January 1987, pp. 70-95

[Houdas 93] M. Houdas, *VERSANT : Un système de gestion de Bases de données Orientée Objet et distribué en environnement hétérogène*, Génie logiciel et systèmes experts, No. 31, Juin 1993, pp. 26-30

[Hudson et King 89] S.E. Hudson, R. King, *Cactis: A Self-Adaptive Concurrent Implementation of an Object-Oriented Database Management System*, ACM Transactions on Database Systems, Vol. 14, No. 3, September 1989, pp. 291-321

[Hurson et al. 93] A.R. Hurson, S.H. Pakzad, J.-b. Cheng, *Object-Oriented Database Management Systems: Evolution and Performance Issues*, IEEE Computer, February 1993, pp. 48-60

[Iaezolla et Mirandola 95] G. Iaezolla, R. Mirandola, *Analysis of two simulation methodologies in performance studies of distributed data bases*, 7th European Simulation Symposium, ESS '95, Erlanger-Nuremberg, Germany, October 1995, pp. 176-180

[IBM 97] IBM, *Site Web*, http://www.storage.ibm.com/hardsoft/diskdrdl/ultra/9zxdata.htm, 1997

[Jackson 83] M. Jackson, *System Development*, Prentice-Hall International, 1983

[Jarke et Koch 84] M. Jarke, J. Koch, *Query Optimization in Database Systems*, ACM Computing Surveys, Vol. 16, No. 2, June 1984

[Karedla et al. 94] R. Karedla, J.S. Love, B.G. Wherry, *Caching Strategies to Improve Disk System Performance*, IEEE Computer, March 1994, pp. 38-46

[Kellert et al. 97] P. Kellert, N. Tchernev, C. Force, *Object-oriented methodology for FMS modelling and simulation*, International Journal Computer Integrated Manufacturing, Vol. 10, No. 6, 1997, pp. 405-434

[Kempe et al. 95] J. Kempe, W. Kowarschick, W. Kießling, R. Hitzelgerger, F. Dutkowski, *Benchmarking Object-Oriented Database Systems for CAD*, 6th International Conference on Database and Expert Systems Applications, DEXA '95, London, UK, 1995; LNCS Vol. 978 (Springer), pp. 167-176

[Kemper et Moerkotte 94] A. Kemper, G. Moerkotte, *Object-Oriented Database Management: Applications in Engineering and Computer Science*, Prentice Hall International Editions, Englewood Cliffs, 1994

[Kernighan et Lin 70] B.W. Kernighan, S. Lin, *An efficient heuristic procedure for partitioning graphs*, Bell System Technical Journal, Vol. 49, No. 2, February 1970, pp. 291-307

[Khoshafian et Abnous 90] S. Khoshafian, R. Abnous, *Object Orientation*, John Wiley & Sons, 1990

[Khoshafian et Copeland 86] S.N. Khoshafian, G.P. Copeland, *Object Identity*, OOPSLA '86 International Conference, Portland, Oregon, September 1986, pp. 29-38

[Kim 90] W. Kim, *Architectural Issues in Object-Oriented Databases*, Journal of Object-Oriented Programming (JOOP), March-April 1990, pp. 29-38

[Kim et al. 87] W. Kim, J. Banerjee, H.-T. Chou, J.F. Garza and D. Woelk, *Composite Object Support in an Object-Oriented Database System*, International Conference on OOPSLA, Orlando, Florida, October 1987, pp. 118-125

[Kim et al. 88] W. Kim, N. Ballou, H.-T. Chou, J.F. Garza, D. Woelk, J. Banerjee, *Integrating an object-oriented programming system with a database system*, OOPSLA '88 International Conference, San Diego, California, September 1988, pp. 142-152

[Kim et al. 89] W. Kim, E. Bertino, J.F. Garza, *Composite Object Revisited*, ACM SIGMOD Conference, Portland, 1989, pp. 337-347

[Kim et al. 90] W. Kim, J.F. Garza, N. Ballou, D. Woelk, *Architecture of the ORION Next-Generation Database System*, IEEE Transactions on Knowledge and Data Engineering, Vol. 2, No. 1, March 1990, pp. 109-124

[Kim et al. 91] W. Kim, N. Ballou, J.F. Garza, D. Woelk, *A Distributed Object-Oriented Database System Supporting Shared and Private Databases,* ACM Transactions on Office Information Systems, Vol. 9, No. 1, January 1991, pp. 31-51

[Knadler 92] C.E. Knadler Jr., *Disk system performance,* 24th Annual Summer Computer Simulation Conference, Reno, Nevada, July 1992, pp. 274-278

[Kuno et Rundensteiner 95] H. Kuno, E.A. Rundensteiner, *Benchmarks for Object-Oriented View Mechanisms,* OOPSLA '95 Workshop on Object Database Behavior, Benchmarks and Performance, Austin, Texas, October 1995

[Lamb et al. 91] C. Lamb, G. Landis, J. Orenstein, D. Weinreb, *The ObjectStore Database System,* Communications of the ACM, Vol. 34, No. 10, October 1991, pp. 50-63

[Lanzelotte et al. 92] R.S.G. Lanzelotte, P. Valduriez, M. Zait, *Optimization of Object-Oriented Recursive Queries Using Cost-Controlled Strategies,* International Conference on Management of Data, ACM SIGMOD '92, San Diego, California, June 1992

[Law et Kelton 91] A.M. Law, W.D. Kelton, *Simulation Modeling and Analysis 2nd Edition,* McGraw-Hill Book Company, 1991

[Leroudier 80] J. Leroudier, *La simulation à événements discrets,* Monographies d'Informatique de l'AFCET, Éditions Hommes et Techniques, 1980

[Lewis et Payne 73] T.G. Lewis, W.H. Payne, *Generalized feedback shift register pseudorandom number algorithm,* Journal ACM, Vol. 20, No. 3, 1973, pp. 456-468

[Little et McCue] M.C. Little, D.L. McCue, *Construction and Use of a Simulation Package in C++,* Technical Report, Department of Computer Science, University of Newcastle upon Tyne, UK

[Loomis 92] M.E.S. Loomis, *Client-server Architecture,* Journal of Object Programming (JOOP), Vol. 4, No. 9, February 1992, pp. 40-44

[Maier et al. 86] D. Maier, J. Stein, A. Otis, A. Purdy, *Development of an Object-Oriented DBMS,* ACM OOPSLA '86, September 1986, pp. 472-482

[Mallordy 96] L. Mallordy, *Regroupement Physique d'Objets dans les SGBD. Analyse, Mise en œuvre et Validation d'une Stratégie Statistique,* Mémoire d'Ingénieur, Spécialité Informatique, CNAM, Centre associé de Clermont-Ferrand, janvier 1996

[McGlenaghan 91] C. McGlenaghan, *OODBMS Benchmark Specification,* Technical Report No. At-12/99-001523-00.01, US West Advanced Technologies, December 1991

[McIver et King 94] W.J. Mc Iver Jr., R. King, *Self-Adaptive, On-Line Reclustering of Complex Object Data*, ACM SIGMOD Conference, Minneapolis, Minnesota, 1994, pp. 407-418

[Mehlhorn et al. 95] K. Mehlhorn, S. Näher, M. Seel, C. Uhrig, *The LEDA User Manual Version 3.7.1*, 1995

[Min et al. 93] H. Min, A.R. Hurson, L.L. Miller, D. Sheth, *An Efficient Storage Protocol for Distributed Object Oriented Databases*, Parallel and Distributed Processing, IEEE CS Press, 1993, pp. 606-610

[Minsky 68] M.L. Minsky, *Matter, mind and models*, Semantic Information Processing, MIT Press, 1968

[Moller 93] P. Moller, *MATISSE*, Génie Logiciel et Systèmes Experts, No.31, juin 1993, pp. 18-25

[Nance 81] R.E. Nance, *Model representation in discrete event simulation: the conical methodology,* Technical Report CS-81003-R, Department of Computer Science, Virginia Tech, Blacksburg, Va., 1981

[NSITC 98] Information Technology Council, X3 Secretariat, *Standard – The C++ Language*, ISO/IEC:98-14882, Washington, DC, USA, 1998

[O'Neil et al 93] E.J. O'Neil, P.E. O'Neil, G. Weikum, *The LRU-K page replacement algorithm for database disk buffering*, International Conference on Management of Data, ACM SIGMOD '93, Washington D.C., May 1993, pp. 297-306

[O_2 Technology 94] O_2 Technology, *Clustering Mechanism reference manual*, Version 4.5, December 1994

[Ontologic Cie 90] Ontologic Cie, *ONTOS Client Library Reference Manual*, December 1990

[Page et al. 97] E.H. Page, R.L. Moose Jr., S.P. Griffin, *Web-Based Simulation in SimJava using Remote Method Invocation*, 1997 Winter Simulation Conference, Atlanta, GA, 1997, pp. 468-473

[Pegden et al. 90] C.D. Pegden, R.E. Shanon, P.P. Sdowski, *Introduction to simulation using SIMAN*, Mc Graw Hill, 1990

[Pritsker 86] A.A.B. Pritsker, *Introduction to Simulation and SLAM II*, Hasted Press (John Wiley & Sons), System Publishing Corporation, 1986

[Rational Software et al. 97a] Rational Software Corporation et al., *UML Semantics, version 1.1,* September 1997

[Rational Software et al. 97b] Rational Software Corporation et al., *UML Notation Guide, version 1.1*, September 1997

[Régnier 98a] S. Régnier, *Étude de DSTC, une technique de regroupement physique d'objets pour les SGBD – Mise en œuvre dans Texas 0.5*, Rapport de projet de 3ème année d'Ingénieur en Informatique, ISIMA, Clermont-Ferrand, mars 1998

[Régnier 98b] S. Régnier, *Codage d'une technique de clustering dans un système de gestion de bases de données orienté objet*, Mémoire d'Ingénieur en Informatique, ISIMA, Clermont-Ferrand, septembre 1998

[Richardson et Carey 89] J. Richardson, M. Carey, *Implementing Persistence in E Language*, International Workshop on Persistent Object Systems, New Castle, Australia, 1989

[Robert 98] C. Robert, *Étude d'une technique de regroupement physique d'objets pour les SGBDOO – Mise en œuvre dans Texas 0.5*, Mémoire de DEA Informatique – Productique – Imagerie Médicale, Université Blaise Pascal, Clermont-Ferrand II, juillet 1998

[Rubenstein et al. 87] W.B. Rubenstein, M.S. Rubicar, R.G.G. Cattell, *Benchmarking simple database operations*, ACM SIGMOD 1987

[Ruemmler et Wilkes 94] C. Ruemmler, J. Wilkes, *An Introduction to Disk Drive Modeling*, IEEE Computer, March 1994, pp. 17-28

[Sargent 79] R.G. Sargent, *Validation of simulation models*, 1979 Winter Simulation Conference, San Diego, 1979, pp. 497-503

[Sargent 91] R.G. Sargent, *Simulation model verification and validation*, 1991 Winter Simulation Conference, Phoenix, 1991, pp. 37-47

[Schreiber 94] H. Schreiber, *JUSTITIA: a generic benchmark for the OODBMS selection*, 4th International Conference on Data and Knowledge Systems in Manufacturing and Engineering, Shatin, Hong Kong, May 1994, pp. 324-331

[Schreiber 95] H. Schreiber, *Benchmarking Structure Preservation and Multi-User Behavior of Object-Oriented Database Systems*, OOPSLA '95 Workshop on Object Database Behavior, Benchmarks and Performance, Austin, Texas, October 1995

[Servio Corporation 92] Servio Corporation, *GemStone V 3.2 Reference Manual*, 1992

[Shekita et Zwiling 95] E.Shekita, M. Zwiling, *Cricket: A Mapped Persistent Object Store*, 4th International Workshop on Object System Design, Implementation and Use, 1995

[Sill 95] D. Sill, *Benchmarks FAQ Version 0.6*, http://hpwww.epfl.ch/bench/bench.FAQ.html, March 1995

[Simulog 92] Simulog, *Modline 1.0: User's guide*, 1992

[Simulog 95] Simulog, *QNAP2 Reference Manual*, 1995

[Singhal et al. 92] V. Singhal, S.V. Kakkad, P.R. Wilson, *Texas: An Efficient, Portable Persistent Store*, 5th International Workshop on Persistent Object Systems, San Miniato, Italy, 1992

[Steffen 90] H. Steffen, *Conception d'un Gérant d'objets et Analyse de performances*, Thèse de Doctorat d'Université, Université Pierre et Marie Curie, Paris VI, mars 1990

[Stepanov et Lee 95] A. Stepanov, M. Lee, *The Standard Template Library*, Technical Report, © Hewlett-Packard Company, 1995

[Stroustrup 97] B. Stroustrup, *The C++ Programming Language, Third Edition*, Addison Wesley, 1997

[Teuhola 95a] J. Teuhola, *Selection of Object Surrogates to Support Clustering*, Data & Knowledge Engineering , Vol. 15, No. 2, April 1995

[Teuhola 95b] J. Teuhola, *Effective Clustering of Objects Stored by Linear Hashing*, 4th International Conference on Information and Knowledge Management, CIKM '95, Baltimore, Maryland, November 1995

[Teuhola 97] J. Teuhola, *Heuristic Clustering of Database Objects According to Multi-Valued Attributes*, 8th International Conference on Database and Expert Systems Applications, DEXA '97, Toulouse, France, September 1997; LNCS Vol. 1308 (Springer), pp. 162-171

[Tiwary et al. 95] A. Tiwary, V.R. Narasayya, H.M. Levy, *Evaluation of OO7 as a system and an application benchmark*, OOPSLA '95 Workshop on Object Database Behavior, Benchmarks and Performance, Austin, Texas, October 1995

[TPC 95] Transaction Processing Performance Council, *TPC BENCHMARK D (Decision Support): Standard Specification*, Technical Report, December 1995

[TPC 96] Transaction Processing Performance Council, *TPC BENCHMARK C: Standard Specification, Revision 3.2*, Technical Report, August 1996

[Tsangaris 92] M.M. Tsangaris, *Principles for Static Clustering for Object Oriented Databases*, Ph.D. Dissertation in Computer Science, University of Wisconsin-Madison, Technical Report No. 1104, August 1992

[Tsangaris et Naughton 91] M.M. Tsangaris, J.F. Naughton, *A Stochastic Approach for Clustering in Object Bases,* ACM SIGMOD International Conference on Management of Data, Denver, Colorado, May 1991, pp. 12-21

[Tsangaris et Naughton 92] M.M. Tsangaris, J.F. Naughton, *On the Performance of Object Clustering Techniques,* ACM SIGMOD International Conference on Management of Data, San Diego, California, June 1992, pp. 144-153

[Valduriez et al. 86] P. Valduriez, S. Khoshofian, G. Copeland, *Implementation Techniques of Complex Objects*, 12th International Conference on Very Large Data Bases, VLDB '86, Kyoto, Japan, August 1986, pp. 101-110

[Van den Berg et Van der Hoeven 96] C.A. Van den Berg, A. Van der Hoeven, *Monet meets OO7,* Object-Oriented Database Systems Symposium, Montpellier, France, July 1996

[Vélez et al. 91] F. Vélez, V. Darnis, D. DeWitt, Ph. Futtersack, G. Harrus, D. Maier, M. Raoux, *Implementing the O_2 Object Manager: Some Lessons*, 4th International Workshop on Persistent Object Systems, San Mateo, California; Readings in Implementing Persistent Object Bases (Morgan Kaufman), 1991, pp. 131-138

[Walliser 77] B. Walliser, *Systèmes et Modèles,* Le Seuil, 1977

[White 94] S.J. White, *Pointer swizzling techniques for object-oriented Databases,* Ph.D. Dissertation in Computer Science, University of Wisconsin-Madison, 1994

[White et DeWitt 92] S.J. White, D.J. DeWitt, *A Performance Study of Alternative Object Faulting and Pointer Swizzling Strategies*, 18th International Conference on Very Large Data Bases, VLDB '92, Vancouver, Canada, August 1992

[White et DeWitt 94] S.J. White, D.J. DeWitt, *Quickstore: A High Performance Mapped Object Store,* ACM-SIGMOD Conference on Management of Data, Minneapolis, MN, May 1994

[Wierzyk 96] V.I. Wierzyk, *Performance Evaluation of New Clustering Algorithm in Object-Oriented Database Systems,* 7th International Conference on Database and Expert Systems Applications, DEXA '96, Zurich, Switzerland, September 1996, pp. 644-653

[Wiggerts 97] T.A. Wiggerts, *Using Clustering Algorithms in Legacy Systems Remodularization,* 4th IEEE Working Conference on Reverse Engineering, October 1997, Amsterdam, The Netherlands, pp. 33-43

[Wilson et Kakkad 92] P.R. Wilson, S.V. Kakkad, *Pointer Swizzling at Page Fault Time: Efficiently and Compatibly Supporting Huge Address Spaces on Standard Hardware*, International Workshop on Object Orientation in Operating Systems, Paris, France, September 1992, pp. 364-377

[Wu et Leahy 93] Z. Wu, R. Leahy, *An Optimal Graph Theoretic Approach to Data Clustering: Theory and Its Application to Image Segmentation*, IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 15, No. 11, November 1993, pp. 1101-1111

[Zeigler 76] B.P. Zeigler, *Theory of Modelling and Simulation*, John Wiley & Sons Inc., New York, 1976

[Zhang et al. 96] T. Zhang, R. Ramakrishnan, M. Livny, *BIRCH: An Efficient Data Clustering Method for Very Large Databases*, ACM SIGMOD Conference, Montreal, June 1996; SIGMOD RECORD, Vol. 25, No. 2, June 1996, pp. 103-114

[Zimmermann et Buchmann 95] J. Zimmermann, A.P. Buchmann, *Benchmarking Active Database Systems: A Requirement Analysis*, OOPSLA '95 Workshop on Object Database Behavior, Benchmarks and Performance, Austin, Texas, October 1995

Résumé

Ce travail s'inscrit dans le domaine de l'évaluation des performances des Systèmes de Gestion de Bases de Données Orientés Objets (SGBDOO). Ces systèmes rencontrent depuis leur origine des problèmes de performance pour s'imposer face aux générations antérieures de SGBD (principalement les systèmes relationnels, qui sont utilisés depuis les années 70 et parfaitement optimisés).

Diverses méthodes sont employées pour améliorer ces performances, telles que le groupement d'objets (*clustering*, dans la terminologie anglo-saxonne). Cependant, ces techniques induisent une surcharge pour le système. Il est donc important de pouvoir évaluer leur impact réel sur les performances globales.

Dans cette optique, une étude approfondie du domaine a permis de dégager les insuffisances majeures attribuables aux techniques d'évaluation de performance existantes (analyse mathématique, bancs d'essais, simulation). Nos propositions, le banc d'essais OCB (*Object Clustering Benchmark*) et le modèle de simulation VOODB (*Virtual Object-Oriented Database*), ont pour objectif de remédier à ces problèmes en se positionnant comme des outils génériques, paramétrables et adaptés à l'étude du regroupement d'objets.

Le modèle de simulation VOODB intègre le banc d'essais OCB. Il a été obtenu par l'application d'une méthodologie de modélisation ayant pour but de systématiser l'analyse des SGBDOO et de fournir des modèles fiables de ces systèmes. Son code a été spécifié dans un environnement de simulation simple et performant que nous avons conçu pour l'occasion et baptisé DESP-C++.

Une démarche de validation de nos outils a été conduite en mesurant les performances réelles du SGBDOO O_2, du gestionnaire d'objets persistants Texas et de la stratégie dynamique de regroupement d'objets DSTC, grâce au banc d'essais OCB. Des évaluations de performance concernant ces systèmes ont également été effectuées avec le modèle de simulation VOODB. Les résultats des mesures sur les systèmes réels et les résultats de simulation ont été comparés et se sont avérés cohérents.

Mots clés : Bases de données orientées objet, Groupement d'objets, Banc d'essais, Modélisation, Simulation.