Quelques fonctionnalités de bases de données avancées

Gia Toan Nguyen

THESE

Présentée à

L'UNIVERSITE SCIENTIFIQUE, TECHNOLOGIQUE ET MEDICALE DE GRENOBLE

pour obtenir le grade de
docteur ès sciences mathématiques

par

NGUYEN GIA TOAN

QUELQUES FONCTIONNALITES DES BASES DE DONNEES AVANCEES.

Thèse soutenue le 19 juin 1986 devant la commission d'examen.

J. MOSSIERE	Président
C. DELOBEL	Directeur
G. GARDARIN	Rapporteur
J.M. NICOLAS	Rapporteur
F. BANCILHON	Examinateur
C. BOITET	Examinateur

UNIVERSITE SCIENTIFIQUE ET MEDICALE DE GRENOBLE

Année universitaire 1982-1983

Président de l'Université : M. TANCHE

MEMBRES DU CORPS ENSEIGNANT DE L'U.S.M.G.

(RANG A)

SAUF ENSEIGNANTS EN MEDECINE ET PHARMACIE

PROFESSEURS DE 1ère CLASSE

ARNAUD Paul	Chimie organique
ARVIEU Robert	Physique nucléaire I.S.N.
AUBERT Guy	Physique C.N.R.S.
AYANT Yves	Physique approfondie
BARBIER Marie-Jeanne	Electrochimie
BARBIER Jean-Claude	Physique expérimentale C.N.R.S. (labo de magnétisme)
BARJON Robert	Physique nucléaire I.S.N.
BARNOUD Fernand	Biosynthèse de la cellulose-Biologie
BARRA Jean-René	Statistiques - Mathématiques appliquées
BELORISKY Elie	Physique
BENZAKEN Claude (M.)	Mathématiques pures
BERNARD Alain	Mathématiques pures
BERTRANDIAS Françoise	Mathématiques pures
BERTRANDIAS Jean-Paul	Mathématiques pures
BILLET Jean	Géographie
BONNIER Jean-Marie	Chimie générale
BOUCHEZ Robert	Physique nucléaire I.S.N.
BRAVARD Yves	Géographie
CARLIER Georges	Biologie végétale
CAUQUIS Georges	Chimie organique
CHIBON Pierre	Biologie animale
COLIN DE VERDIERE Yves	Mathématiques pures
CRABBE Pierre (détaché)	C.E.R.M.O.
CYROT Michel	Physique du solide
DAUMAS Max	Géographie
DEBELMAS Jacques	Géologie générale
DEGRANGE Charles	Zoologie
DELOBEL Claude (M.)	M.I.A.G. Mathématiques appliquées
DEPORTES Charles	Chimie minérale
DESRE Pierre	Electrochimie
DOLIQUE Jean-Michel	Physique des plasmas
DUCROS Pierre	Cristallographie
FONTAINE Jean-Marc	Mathématiques pures
GAGNAIRE Didier	Chimie physique

.../...

GASTINEL Noël	Analyse numérique - Mathématiques appliquées
GERBER Robert	Mathématiques pures
GERMAIN Jean-Pierre	Mécanique
GIRAUD Pierre	Géologie
IDELMAN Simon	Physiologie animale
JANIN Bernard	Géographie
JOLY Jean-René	Mathématiques pures
JULLIEN Pierre	Mathématiques appliquées
KAHANE André (détaché DAFCO)	Physique
KAHANE Josette	Physique
KOSZUL Jean-Louis	Mathématiques pures
KRAKOWIAK Sacha	Mathématiques appliquées
KUPTA Yvon	Mathématiques pures
LACAZE Albert	Thermodynamique
LAJZEROWICZ Jeannine	Physique
LAJZEROWICZ Joseph	Physique
LAURENT Pierre	Mathématiques appliquées
DE LEIRIS Joël	Biologie
LLIBOUTRY Louis	Géophysique
LOISEAUX Jean-Marie	Sciences nucléaires I.S.N.
LOUP Jean	Géographie
MACHE Régis	Physiologie végétale
MAYNARD Roger	Physique du solide
MICHEL Robert	Minéralogie et pétrographie (géologie)
MOZIERES Philippe	Spectrométrie - Physique
OMONT Alain	Astrophysique
OZENDA Paul	Botanique (biologie végétale)
PAYAN Jean-Jacques (détaché)	Mathématiques pures
PEBAY PEYROULA Jean-Claude	Physique
PERRIAUX Jacques	Géologie
PERRIER Guy	Géophysique
PIERRARD Jean-Marie	Mécanique
RASSAT André	Chimie systématique
RENARD Michel	Thermodynamique
RICHARD Lucien	Biologie végétale
RINAUDO Marguerite	Chimie CERMAV
SENGEL Philippe	Biologie animale
SERGERAERT Francis	Mathématiques pures
SOUTIF Michel	Physique
VAILLANT François	Zoologie
VALENTIN Jacques	Physique nucléaire I.S.N.
VAN CUTSEN Bernard	Mathématiques appliquées
VAUQUOIS Bernard	Mathématiques appliquées
VIALON Pierre	Géologie

PROFESSEURS DE 2ème CLASSE

ADIBA Michel	Mathématiques pures
ARMAND Gilbert	Géographie

.../...

AURIAULT Jean-Louis	Mécanique
BEGUIN Claude (M.)	Chimie organique
BOEHLER Jean-Paul	Mécanique
BOITET Christian	Mathématiques appliquées
BORNAREL Jean	Physique
BRUN Gilbert	Biologie
CASTAING Bernard	Physique
CHARDON Michel	Géographie
COHENADDAD Jean-Pierre	Physique
DENEUVILLE Alain	Physique
DEPASSEL Roger	Mécanique des fluides
DOUCE Roland	Physiologie végétale
DUFRESNOY Alain	Mathématiques pures
GASPARD François	Physique
GAUTRON René	Chimie
GIDON Maurice	Géologie
GIGNOUX Claude (M.)	Sciences nucléaires I.S.N.
GUITTON Jacques	Chimie
HACQUES Gérard	Mathématiques appliquées
HERBIN Jacky	Géographie
HICTER Pierre	Chimie
JOSELEAU Jean-Paul	Biochimie
KERCKOVE Claude (M.)	Géologie
LE BRETON Alain	Mathématiques appliquées
LONGEQUEUE Nicole	Sciences nucléaires I.S.N.
LUCAS Robert	Physiques
LUNA Domingo	Mathématiques pures
MASCLE Georges	Géologie
NEMOZ Alain	Thermodynamique (CNRS - CRTBT)
OUDET Bruno	Mathématiques appliquées
PELMONT Jean	Biochimie
PERRIN Claude (M.)	Sciences nucléaires I.S.N.
PFISTER Jean-Claude (détaché)	Physique du solide
PIBOULE Michel	Géologie
PIERRE Jean-Louis	Chimie organique
RAYNAUD Hervé	Mathématiques appliquées
ROBERT Gilles	Mathématiques pures
ROBERT Jean-Bernard	Chimie physique
ROSSI André	Physiologie végétale
SAKAROVITCH Michel	Mathématiques appliquées
SARROT REYNAUD Jean	Géologie
SAXOD Raymond	Biologie animale
SOUTIF Jeanne	Physique
SCHOOL Pierre-Claude	Mathématiques appliquées
STUTZ Pierre	Mécanique
SUBRA Robert	Chimie
VIDAL Michel	Chimie organique
VIVIAN Robert	Géographie

Je ne sais comment remercier Claude DELOBEL, Professeur à l'Université de Grenoble et de Paris-Sud à Orsay, et Responsable du Conseil Scientifique du Programme de Recherches Coordonnées "Bases de Données de 3e génération", pour le chemin tout au long duquel il m'a conduit depuis dix ans dans le domaine de la recherche en informatique et des bases de données. Après m'avoir acueilli dans son équipe comme chercheur débutant, son soutien indéfectible et ses encouragements permanents m'ont permis de mener à bien tous les travaux auxquels je me suis attelé. En des mots trop ternes et trop brefs, je crois qu'il est exact de dire qu'il m'a tout appris des bases de données. Je lui en serai toujours redevable. Qu'il trouve ici, si cela est possible, l'expression de ma profonde gratitude.

Georges GARDARIN, Professeur à l'Université de Paris VI, Responsable du projet SABRE à l'INRIA et du Programme de Recherches Coordonnées "Bases de Données de 3e Génération", a également été de tous temps l'aiguillon et l'interlocuteur privilégié grâce à qui mes connaissances en matière de Base de Données sont devenues ce qu'elles sont. Je le remercie pour les innombrables contacts et les échanges fructueux que nous avons eus ces dernières années sur notre sujet de recherche commun.

Je remercie également Jean-Marie NICOLAS, Directeur de Recherche à l'European Computer-industry Research Center de Munich, pour l'intérêt qu'il a manifesté très tôt pour les travaux décrits dans cette thèse. Ses critiques, ses idées et les discussions que nous avons eues sont en grande partie à la base de mes motivations pour les bases de données déductives et la logique. Qu'il en soit ici chaleureusement remercié.

Je dois rendre hommage à Jacques MOSSIERE, Directeur du Laboratoire de Génie Informatique à l'IMAG et Professeur à l'ENSIMAG, pour le cadre intellectuel, scientifique et moral dont il assume la responsabilité en fournissant un environnement de travail exceptionnel aux chercheurs du Laboratoire. Je ne saurais trop dire le plaisir et l'honneur qu'il me fait en présidant le jury de cette thèse.

Enfin, je remercie François BANCILHON, Professeur à l'Université de Paris Sud à Orsay, détaché à la Micro-electronics Computer Corporation à Austin (Texas), et prochain responsable du GIP INRIA-Intertechnique, ainsi que Christian BOITET, Professeur à l'Université de Grenoble et Directeur du Groupe d'Etudes pour la Traduction Automatique d'avoir bien voulu accepter de juger ce travail.

Je tiens également à exprimer ma reconnaissance à :

- *Jean ROHMER, Responsable du Groupe de Recherche en Intelligence Artificielle au Centre de Recherche BULL de Louveciennes,*

- *Michel SCHOLL, Chercheur à l'INRIA et*
- *François BRY, Chercheur à l'ECRC de Munich,*

pour leurs lectures attentives de ce manuscrit et pour leurs critiques et suggestions.

Enfin, parmi tous ceux et surtout toutes celles qui ont contribué humainement à l'achèvement de ce travail, je ne saurais dire combien je suis redevable à Dominique de tous les instants que nous avons passés ensemble ces derniers temps.

RESUME

On propose d'utiliser conjointement des techniques inspirées du domaine des Bases de Données et de l'Intelligence Artificielle pour mettre en

On caractérise les besoins de ces applications afin de mettre en évidence les faiblesses des SGBD classiques. On développe ensuite des techniques de :

- modélisation de données généralisées,
- de répartition des informations,
- de contrôle de contraintes sémantiques.

On utilise une méthode de représentation de l'information basée sur la logique des prédicats du premier ordre pour enrichir la représentation sémantique des données qui peuvent être stockées dans un ensemble de Bases de Données réparties.

On définit ensuite une méthode originale d'évaluation de questions sur des données distribuées basée sur une décomposition dynamique des opérations.

On propose enfin une nouvelle approche pour le contrôle des contraintes sémantiques dans une base de données. Elle est basée sur la notion de prototypes logiques d'objets formant un échantillon de la base de données.
L'échantillon est analogue à un résumé dans lequel les erreurs sur la structure des objets sont interdites, mais les erreurs et les valeurs inconnues concernant leurs propriétés sont admises.
Cette approche présente l'avantage de fournir une méthode unifiée pour le contrôle des mises à jour portant sur la structure dynamique des objets ainsi que sur les valeurs des informations qu'ils contiennent.
Elle s'appuie entre autre sur une évaluation à priori des effets des mises à jour. A ce titre, c'est une méthode pessimiste de vérification de contraintes sémantiques.

Mots-clés :

Bases de données, logique, répartition, certification, prototypes d'objets, contraintes d'intégrité, intégrité sémantique.

AVANT-PROPOS

L'ensemble des travaux présentés ici est l'aboutissement d'études menées dans le cadre de plusieurs projets du Laboratoire de Génie Informatique, à l'IMAG. Parmi ces projets, il faut citer URANUS, POLYPHEME, MICROBE, OPALE et TIGRE.

Sans faire d'élitisme ni de régionalisme excessifs, on peut mentionner qu'URANUS a été l'un des premiers systèmes de base de données relationnel développé en France, POLYPHEME l'un des premiers prototypes de base de données réparties expérimenté dans le monde et MICROBE le premier système de base de données relationnel opérationnel en France sur un micro-ordinateur. Depuis la réalisation du système SOCRATE à l'IMAG en 1972, MICROBE est le seul SGBD conçu à l'Université de Grenoble qui ait fait l'objet hors de ses murs de développements suivis d'applications opérationnelles.

Sous la responsabilité du CNET Grenoble qui l'a porté sur VAX (VMS) et APOLLO (Aegis), MICROBE a servi de base au SGBD du projet Européen CVT en CAO de VLSI. Sous la responsabilité de l'Ecole Centrale de Lyon, il a été porté sur PC compatibles IBM (MS-DOS). Il est par ailleurs opérationnel sur LSI-11 (RSX 11-M) et sur SM90 (SMX). Dans ce dernier cas, il est utilisé comme support de règles pour un démonstrateur de théorème en calcul formel réalisé au Laboratoire de Génie Informatique.

Les idées et les réalisations qui en ont résulté ont fait l'objet d'heures passionnantes et passionnées de discussions auxquelles ont participé :

- pour le projet MICROBE,
Melle F. Azrou et MM. A. Bensaid, A. Desbornes, F. Fernandez, L. Ferrat, Y.J Lee, et G. Sergeant,

- et pour le projet TIGRE,
Melles Marie-Christine Fauvet, Judith Olivares, Dominique Rieu et Pascale Winninger.

Tous ont participé d'une manière ou d'une autre à la maturation des idées présentées ici, ainsi qu'à leur expérimentation dans le cadre de leur thèse de Doctorat. Leurs efforts ont été à la base de ce travail. Qu'ils en soient ici tous chaleureusement remerciés.

Il est également nécessaire de souligner la contribution du Département Recherche en Conception Assistée du CNET Grenoble aux travaux décrits dans cette thèse. Il a fournit d'une part un ca-

dre d'expérimentation irremplaçable pour l'évaluation des prototypes qui ont été réalisés, en particulier le SGBD MICROBE, et d'autre part une occasion unique de transfert technologique et d'échanges avec des professionnels de la CAO. Parmi ceux-ci, nous remercions tout particulièrement MM. Jacques Lecourvoisier, Responsable du Département, et Christian Jullien, pour leur ouverture d'esprit et la confiance qu'ils nous ont accordées.

Plusieurs autres organismes publics de recherche ont apporté leur soutien matériel à la réalisation des projets mentionnés ici. Parmi ceux-ci figure principalement l'INRIA, qui nous compte dans ses rangs depuis plusieurs années. Enfin l'environnement scientifique et moral de l'IMAG et du Laboratoire de Génie Informatique de l'Université Scientifique Technologique et Médicale de Grenoble a été le creuset de toutes les idées émises et expérimentées durant le déroulement de ces travaux.

NOTATIONS

---

x c D : x appartient à D.

(x) c D : pour tout x appartenant à D.

(E x) c D : il existe x appartenant à D.

SOMMAIRE

CHAPITRE I

"In medio stat virtus"

INTRODUCTION

Certains ont pu dire qu'il existait entre les chercheurs en Intelligence Artificielle et les chercheurs en Bases de Données une gigantesque partie de go, dans laquelle chaque camp tentait d'"encercler" l'autre pour conquérir, avec ses propres outils et techniques, celles de l' "adversaire" [TSI85].

Cette opinion, exprimée sur un ton humoristique, reflète bien la position relative de ces deux domaines de la recherche en informatique auparavant très éloignés l'un de l'autre, où le savoir-faire des uns peut être appliqué à des domaines de plus en plus nombreux, en atteignant toutefois des limites que le savoir-faire des autres peut contribuer à faire dépasser.

On situe l'étude qui va suivre dans cette mouvance générale qui va de l'extension des systèmes utilisant les méthodes de l'Intelligence Artificielle à l'extension des Systèmes de Gestion de Bases de Données à l'aide, pour chacun d'eux, des techniques empruntées à l'autre. Il est clair que la partie de go n'est pas encore terminée, loin de là, et que les positions respectives des deux camps évoluent constamment.

La référence à un tel jeu reflète une vision très polémique des rapports entre les deux communautés. Si elle correspond effectivement à une certaine réalité et à des positions bien tranchées exprimées ici et là [STO85], on peut aussi adopter des attitudes moins guerrières. En particulier, être plus pacifique, ou réaliste, et voir les deux camps comme des spécialités très complémentaires [WIE84]. Ceci est justifié d'une part par de nombreux travaux qui ont permis de faire le lien entre les aspects théoriques qui sont le fondement des bases de données, par exemple le modèle relationnel et la logique des prédicats du premier ordre et donc certaines des techniques utilisées en Intelligence Artificielle [GAL81], et d'autre part une évolution des besoins des deux communautés [MIS84].

Pour des raisons "historiques", notre approche a pour but d'étendre les fonctionnalités des Systèmes de Gestion de Bases de Données par de nouvelles techniques. En particulier en ce qui concerne la représentation et la manipulation de la structure dynamique des données et des contraintes sémantiques.

Les bases de données sont utilisées maintenant dans un grand nom-

bre d'applications, et leur utilisation croissante a contribué à l'émergence de besoins nouveaux. De simples réceptacles passifs unifiant la représentation, le stockage et l'accès aux informations, on va par exemple irrémédiablement vers des bases de données actives, "intelligentes", ou encore "avancées" [GAR85]. Dans celles-ci, l'utilisateur ne se contente plus de décrire la structure statique des informations, mais également la sémantique des opérations qui leur sont attachées, ainsi que leur comportement dynamique.

De fait, l'histoire des Bases de Données est relativement courte puisqu'elle remonte à une vingtaine d'années, mais elle est très mouvementée. Elle a déjà connu deux révolutions, et en vit actuellement une troisième. Les deux premières sont les passages des systèmes de bases de données hiérarchiques aux systèmes réseaux, puis aux systèmes relationnels. La troisième résulte de la conjuguaison des systèmes distribués et des moyens de calcul individuels sophistiqués. Notre but est de mettre en œuvre des méthodes qui permettent de préparer la quatrième (celle des bases de données "intelligentes") en douceur.

> L'approche adoptée est d'emprunter aux méthodes de l'Intelligence Artificielle un petit nombre d'outils, et de les utiliser pour étendre les fonctionnalités des Systèmes de Gestion de Bases de Données. Elle se situe donc au confluent de ces deux domaines (Figure 1.1).

Bases de Données -----> <----- Intelligence Artificielle

Figure 1.1.

On fera un usage intensif de la programmation en logique. Bien que son essort déborde aujourd'hui largement le cadre de l'Intelligence Artificielle [HOR84], la vérité historique impose en effet de la placer parmi ces techniques, puisqu'elle a été très largement connue à travers les recherches sur le traitement de la langue naturelle [COL83].

On aboutit ainsi à des outils qui pourront être utilisées pour la mise en place de ce que l'on appelera des applications "avancées".

> De facon générale, celles-ci regroupent les applications qu'il est impossible de mettre en place aujourd'hui à l'aide de méthodes "classiques", tels que les langages de programmation procéduraux, les Systèmes de Gestion de Fichiers courants, sans un minimum d'aide à la décision, et qui font donc appel à des traitements non-déterministes et des heuristiques. La Conception Assistée par Ordinateur en est un exemple [FIN85, MIT85]. La Bureautique un autre. Le Génie Logiciel également. Cette étude puise donc dans ces domaines, et en particulier celui de la CAO, les exigences

des problèmes qu'elle entend explorer (Figure 1.2).

Notre thèse est qu'il est illusoire de vouloir intégrer dans un même ensemble des mécanismes aussi divers et complexes que des Systèmes Experts et des Systèmes de Gestion de Base de Données. Tout au moins tels qu'on les connaît et tels qu'on les pratique aujourd'hui. Bien sûr, la somme de leurs potentialités est d'un intérêt évident, que ce soit pour l'aide à la décision ou les applications de conception.

Certains points demeurent cependant non résolus à ce jour, comme le partage de grandes bases de connaissances entre plusieurs utilisateurs, le contrôle de leur cohérence ou leur utilisation dans un système de gestion répartie de l'information.

Le but est ici d'aborder la manipulation des structures de l'information et le contrôle des contraintes sémantiques.

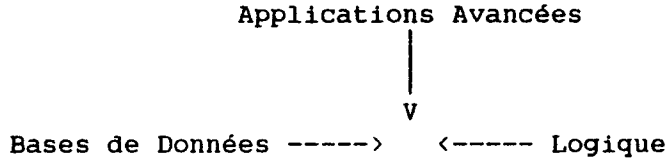


Figure 1.2.

On ne propose donc pas le n+1 ème modèle de données, mais plus modestement des méthodes qui étendent les fonctionnalités des SGBD (Figure 1.3).

On suit une démarche très pragmatique, en faisant constamment référence à des domaines d'application précis comme la Conception Assistée par Ordinateur, grâce à laquelle on a pu expérimenter concrètement nos idées [ADI84, ADI85].

On vise dans un premier temps la coopération de techniques diverses. On développe ensuite des méthodes qui permettent leur intégration.

Pour cela, on juxtapose un certain nombre de techniques qui permettent de doter les Systèmes de Gestion de Bases de Données de mécanismes de contrôle et de vérification de contraintes sémantiques évolués. On s'attache également à développer des outils qui permettent une évolution graduelle des applications existantes. C'est ainsi qu'on aborde le problème de la répartition des informations.

Une exigence reconnue des applications avancées est en effet la coopération en un ensemble homogène d'outils existant ou autonomes, dédiés à des fonctions particulières, comme les programmes de test de circuits intégrés, les simulateurs, les calculs d'éléments finis [HOW84]. Il est donc nécessaire de pouvoir

intégrer dans un même ensemble des logiciels d'origines différentes, donc utilisant des modèles de données divers et des structures de données variées. De même, les exigences de ces applications impliquent la plupart du temps la coopération entre plusieurs groupes d'utilisateurs. C'est la raison pour laquelle il faut, en plus d'une gestion de données et de traitement répartis de façon logique, être capable de gérer leur répartition physique [REH84].

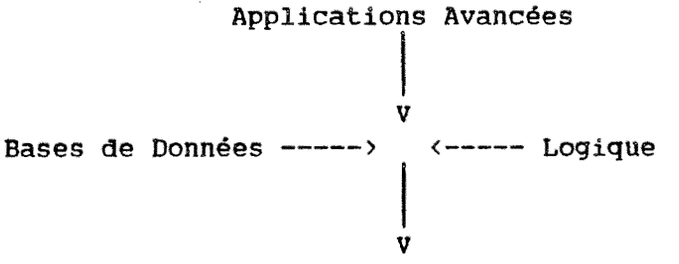


Figure 1.3.

Afin d'être aussi proche que possible des besoins des utilisateurs, on a donc adopté une démarche "utilitaire" en s'appuyant sur des exemples réels.

La diffusion des Systèmes de Gestion de Bases de Données par exemple a apporté une aide certaine à la gestion des informations dans les systèmes intégrés de CAO. Des travaux importants ont permis de définir les extensions nécessaires à ce type d'applications [LOR84, KIM84]. Pour avoir assisté de près à une telle expérience, on peut dire que leur usage a permis de trouver une solution satisfaisante à la gestion de données qui devenaient autrement inextricables [JUL86]. Elle reste toutefois fortement tributaire des fonctionnalités des SGBD classiques, relationnels par exemple, pour l'expression et l'évaluation de contraintes d'intégrité par exemple.

On a pratiquemment atteint leurs limites opérationnelles. En effet, leur pouvoir de description de l'information et d'expression des contraintes sémantiques associées est sans rapport avec les besoins ces applications.

On définit donc ici des outils qui étendent les fonctionnalités des Systèmes de Gestion de Bases de Données pour un contrôle de contraintes sémantiques plus étendu, et qui soient suffisamment généraux pour servir de support à une grande variété d'applications.

La démarche adoptée se décompose en trois étapes successives :

- le premier objectif est la spécification , à l'aide d'outils adéquats, des concepts et du comportement des ob-

jets associés aux applications avancées,
- le deuxième est ensuite d'en faire une modélisation qui permette d' en automatiser le traitement,
- le troisième enfin est de permettre ce traitement, c-à-d l' évaluation des opérations qui s'appuient sur cette modélisation, pour mettre en œuvre des applications pratiques (Figure 1.4).

On fait ici le pari que ces trois objectifs peuvent être atteints en utilisant un seul et même outil, c-à-d la logique des prédicats du premier ordre.

On cherche à éviter les deux travers les plus courants concernant le rapprochement des Bases de Données et de la logique :

- le premier qui consiste à ne considérer la logique que comme un langage d'interrogation de bases de données,
- le deuxième qui consiste à la considérer comme un épouvantail avec lequel on peut théoriquement spécifier des propriétés ou répondre à des questions, mais si lentement ou difficilement que son usage relève encore d'exemples purement académiques.

La spécification de la sémantique d'objets complexes et des opérations associées en logique du premier ordre est un domaine prometteur. Elle a été explorée dans le cadre du projet ACACIA par exemple [CET84]. Leur modélisation en logique du premier ordre quant à elle a été expérimentée dans de nombreux travaux [FIN85, ZAU83]. Restreinte aux clauses de Horn, elle présente l'avantage décisif d'être directement interprétable dans un langage de traitement symbolique comme PROLOG [COL83]. C'est une des raisons qui a motivé notre choix. La deuxième raison est le lien qui existe entre le calcul des prédicats et la théorie du modèle relationnel de données [DEM82, DEL82, GAL84], ce qui nous permet d'utiliser tout l'acquis des recherches dans ce domaine. La troisième est la parenté qui existe entre la logique et les systèmes de représentation des connaissances [MIT85, WIE84].

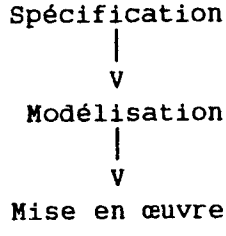


Figure 1.4. Mise en œuvre d'applications avancées.

La démarche qui est proposée ici présente deux caractéristiques principales :

- elle se situe à un niveau conceptuel,
- elle permet la représentation de la sémantique des informations.

Pour cela, elle s'appuie sur l'extension mutuelle des fonctionnalités d'un système déductif et d'un SGBD réparti. Le rôle de celui-ci est de gérer efficacement les objets complexes utilisés par les applications. Le rôle du système déductif est de contrôler la cohérence des opérations de :

- définition et modification du schéma du SGBD,
- définition et modification des contraintes liées aux informations,
- modification effectuées par les usagers par évaluation (à la demande) de ces contraintes.

L'ensemble est piloté de manière centralisée par un système logique de validation qui sera décrit au Chapitre VI. Une interface de haut niveau, orientée objet, est proposée au Chapitre IV pour simplifier l'utilisation de l'ensemble (Figure 1.5).

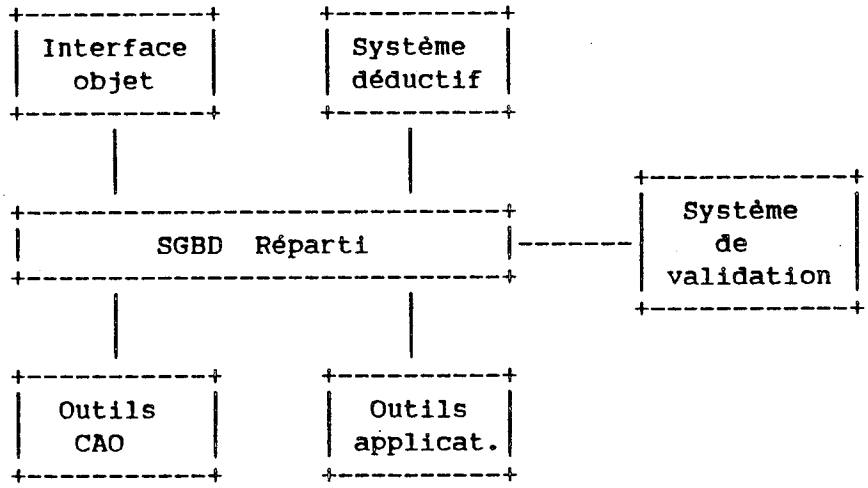


Figure 1.5 Réalisation.

Dans un premier temps (Figure 1.6), on fait un certain nombre de rappels sur les principes de base de la logique mathématique et sur ses rapports avec la théorie des bases de données (Chapitre II).

On caractérise ensuite les besoins de ce qu'on appelle les applications avancées : c'est le premier objectif de "spécification" des connaissances et des données (Chapitre III).

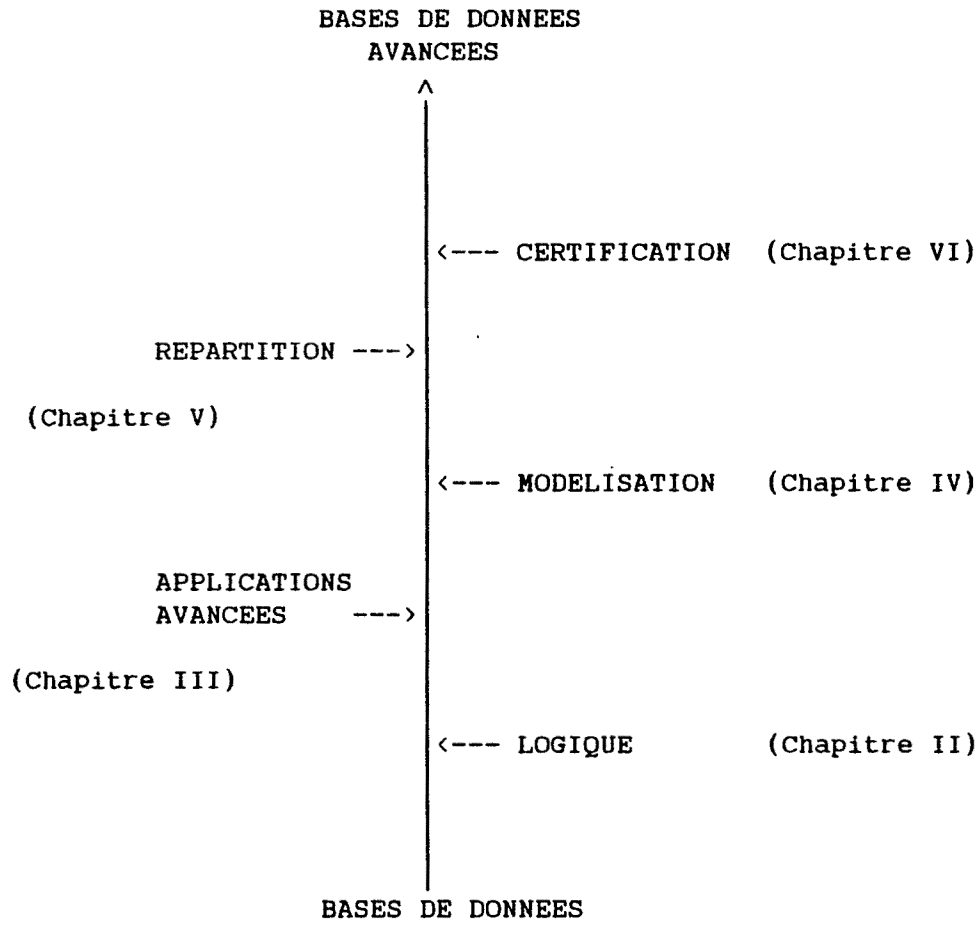


Figure 1.6. Démarche adoptée.

Le chapitre IV présente quelques solutions pour la modélisation des informations dans les Systèmes de Gestion de Bases de Données Avancées.

On détaille ensuite les fonctionnalités nécessaires à la gestion d'informations réparties : c'est le troisième objectif, celui du traitement ou de l'"évaluation" des opérations (Chapitre V).

Le Chapitre VI est consacré à la gestion et à la vérification de contraintes sémantiques dans les applications avancées à l'aide d'une méthode originale, dite de certification.

On tire enfin quelques conclusions et l'on ouvre les perspectives offertes par cette étude (Chapitre VII).

Une expérimentation de ces propositions concernant le SGBD multi-média TIGRE est décrite dans [OLI86]. Par ailleurs, une application à la Conception Assistée par Ordinateur est présentée dans [RIE85].

CHAPITRE II

LOGIQUE ET BASES DE DONNEES

2.1 Concepts de base.

- 2.1.1 Modèles de données et représentation des connaissances.
- 2.1.2 Systèmes de gestion de bases de données.
- 2.1.3 Intelligence artificielle et logique.

2.2 Fondements de la logique.

- 2.2.1 Le calcul des prédicats du premier ordre.
- 2.2.2 Déduction et résolution.

2.3 Logique et bases de données.

- 2.3.1 Deux théories.
- 2.3.2 Accès aux informations.
- 2.3.3 Mise en œuvre des traitements.

CHAPITRE II

"Ab origine"

LOGIQUE ET BASES DE DONNEES

A l'origine considérées comme deux disciplines indépendantes, la Logique appliquée à l'informatique et les Bases de Données sont deux domaines dont les relations ont émergé petit à petit et tendent à s'accroître de façon inéluctable [GAL84].

Elles sont apparues en poursuivant des chemins spécifiques. Dans un cas par exemple pour le traitement automatisé du langage naturel, dans l'autre pour la gestion de quantités importantes d'informations structurées.

Des travaux théoriques ont jeté les premiers liens entre elles une dizaine d'années plus tard. Ainsi les recherches effectuées sur les mécanismes de résolution automatique de problèmes ont été formalisées par certains chercheurs en logique mathématique [KOW79]. L'utilisation de la logique des prédicats pour spécifier les concepts fondamentaux des bases de données a par ailleurs ouvert la voie à ce que l'on peut qualifier d'étape majeure dans leur évolution [GAL81, NIC79].

On assiste depuis lors à une convergence qui est amplifiée par les besoins manifestés par les utilisateurs de ces deux technologies.

L'usage opérationnel des Systèmes de Gestion de Bases de Données (SGBD) qui s'est largement développé durant la décennie 1975-1985 a en effet démontré qu'une de leurs lacunes résidait dans l'impossibilité de définir et de mettre en œuvre les spécifications du comportement dynamique des informations. Parallèlement, ce type de fonctionnalité a été développé et utilisé de manière systématique par les chercheurs en Intelligence Artificielle, par exemple dans le cadre des modèles de représentation des connaissances [ISR84].

Cette conjonction de travaux théoriques et de besoins pratiques est à la base de recherches auxquelles participe aujourd'hui une grande partie de la communauté scientifique et industrielle internationale. Ils visent d'une part à approfondir les liens entre les aspects formels des travaux en Intelligence Artificielle et, dans le domaine des bases de données, la spécification des infor-

mations ainsi que la sémantique des opérations qui leur sont attachées.

Ils ont également pour objectif de concevoir des outils informatiques plus évolués en matière de gestion d'informations et de résolution de problèmes.

Le but de ce chapitre est de rappeler les éléments significatifs de cette convergence, d'une part sur le plan théorique, d'autre part sur ses retombées pratiques. On fait de brefs rappels sur les aspects formels qui lient les domaines de la logique et des Bases de Données. On insiste sur l'utilisation commune de la logique pour la résolution de problèmes et la modélisation des informations. On aborde ensuite des aspects plus concrets sur les interactions qu'elle rend possible entre les techniques utilisées dans ces deux domaines.

2.1 Concepts de base.

A la base de l'Intelligence Artificielle et des SGBD se trouve l'objectif commun de modéliser, stocker et manipuler de l'information. Ce terme générique désigne dans chacune de ces disciplines des concepts qui les différencient fondamentalement.

L'intérêt de la première s'est concentré sur le raisonnement et les connaissances nécessaires à la résolution de problèmes. L'intérêt de la seconde sur les données factuelles.

On donne brièvement les éléments significatifs de ces deux approches.

2.1.1 Modèles de données et représentation des connaissances.

Représenter, stocker et manipuler de l'information par des moyens électroniques nécessite un codage approprié aux calculateurs. Le but d'un modèle de données est d'en spécifier la syntaxe et la sémantique afin de le rendre utilisable par les usagers. L'histoire et l'évolution des SGBD conduit à classer les modèles de données en quatre familles :

- les modèles de données hiérarchiques,
- les modèles réseaux,
- les modèles relationnels,
- les modèles sémantiques [DEL82].

Du point de vue des spécialistes en Intelligence Artificielle, les divers modes de représentation de l'information peuvent être classés en :

- réseaux sémantiques,
- frames,
- règles de production,
- logique des prédicats [COM85].

o Modèle de données hiérarchique.

Schématiquement, les modèles de données hiérarchiques permettent de modéliser l'information selon un ensemble d'enregistrements structurés et liés entre eux par des liens arborescents. Ces enregistrements et ces liens sont définis de manière statique, et imposent aux occurences des données contenues dans une base un type pour chacun de leurs champs. Une base de données est de ce fait une forêt d'arbres valués. Ces systèmes ont fait leur apparition sur le marché industriel dans les années soixante.

o Modèle de données réseau.

Les modèles de données en réseau constituent une extension des modèles hiérarchiques. Ils ont été commercialisés dans les années soixante-dix. En plus de la notion de hiérarchie de liens, les enregistrements peuvent ici être liés par des associations quelconques, c-à-d que deux arborescences valuées peuvent être liées entre elles, et deux enregistrements appartenant à la même arborescence valuée peuvent également être liés entre eux. Les travaux de normalisation du groupe CODASYL ont donné lieu à la définition d'une norme internationale concernant ce type de modèlisation qui a été suivie pour de nombreux SGBD disponibles sur le marché. Dans ce type de système, la définition structurelle de l'information demeure également statique.

o Modèle de données relationnel.

Les modèles de données relationnels sont basées sur la définition d' ensembles de propriétés communes aux informations qui peuvent être regroupées dans des aggrégats ou relations. Ces propriétés peuvent être définies à l'aide de prédicats qui doivent être vérifiés par tout élément ou n-uplet d'une relation. Les SGBD qui implantent le modèle de données relationnel ont permis d' effectuer un saut technologique majeur dans la mise en œuvre des logiciels de gestion d'informations structurées pour plusieurs raisons. Ce sont d'une part les premiers à être basés sur des notions mathématiques formelles comme le calcul des relations et la théorie des ensembles. D'autre part la définition explicite des liens entre les informations n'est plus nécessaire au niveau physique d'implantation des données. Elle peut être

faite à un niveau purement logique, par exemple par la notion de vues. Enfin des langages de manipulation ensemblistes ont pu être définis et ont permis de s'affranchir de la gestion explicite par l'utilisateur des enregistrements et méthodes d'accès, ce qui n'était pas le cas auparavant.

Paradoxalement, cette simplicité a fait l'objet d'un certain nombre de critiques, en particulier du fait de l'impossibilité de définir et d'exploiter les liens sémantiques entre les informations, qui étaient dans les modèles hiérarchiques et en réseau directement implantés au niveau des enregistrements sous la forme de pointeurs. C'est la raison pour laquelle le modèle relationnel initial a été étendu et a donné lieu au développement d'une nouvelle classe de modèles appelés modèles sémantiques [HUL85].

o Modèles de données sémantiques.

En résumé, ceux-ci permettent de définir au niveau conceptuel les liens entre les informations. Ceci inclut à la fois les liens de nature intrinsèques comme l'héritage de propriétés entre données, et les liens de nature extrinsèque ou opérationnelle, comme les dépendances entre données. Cette possibilité constitue un avantage essentiel dans la mesure où les liens sémantiques entre les informations peuvent être exprimés fonctionnellement, c-à-d par le rôle qu'ils jouent dans les relations entre les informations, et non plus par leur implantation physique dans une structure de donnée.

A ce jour, peu de SGBD réellement opérationnels ont implanté ce type de modèle. Plusieurs projets de recherche ont cependant adopté cette philosophie dans les années quatre-vingt [TIG83, LIE81], tout en étant l'objet de critiques inverses aux précédentes, comme par exemple : la difficulté de compréhension et d'utilisation d'un modèle de données croît plus qu'exponentiellement avec le nombre de concepts qu'il permet de mettre en œuvre [STO85].

Ce souci de représenter l'information factuelle de manière sûre, partageable et efficace a contribué a faire connaître la technologie des SGBD dans tous les domaines scientifiques, industriels et commerciaux.

o Représentation des connaissances.

Parallèlement, les travaux liés à l'Intelligence Artificielle ont eu pour objectifs la représentation des propriétés de l'information et la résolution de problèmes.

De ce fait, la représentation de l'information a ici fait l'objet de techniques particulières, comme les réseaux sémantiques, où les liens opérationnels entre données sont particulièrement mis en avant [ISR84].

Ceci les distingue des modèles utilisés par les SGBD de manière fondamentale, car on met en avant ici les aspects dynamiques de l'information, alors que les modèles de données précédents ne permettent qu'une description statique.

Les concepts que l'on peut définir dans les modèles sémantiques des bases de données et les réseaux sémantiques en Intelligence Artificielle sont si proches qu'il est parfois difficile de définir la frontière entre les deux (ce qui ne contribue pas a apaiser la polémique sur le jeu de go ...). Les notions d'héritage de propriétés, de hiérarchie de types ou de spécialisation sont en effet présentes dans les deux.

La différence fondamentale réside à ce niveau dans les caractéristiques opérationnelles, plus que dans les concepts. Les modèles sémantiques sont en effet des outils de modélisation. Ce sont les parties visibles de l'iceberg dont les éléments cachés se nomment procédures d'accès et de recherche d'informations. Ces aspects sont occultés dans les systèmes de représentation des connaissances en Intelligence Artificielle au profit de capacités de déduction, de résolution de problème et d'attachement procédural plus élaborées.

Une étape supplémentaire a été franchie dans les années 1975 avec la notion de squelette ou "frame", dans laquelle le comportement dynamique des objets peut être spécifié de manière procédurale pour chacune de ses caractéristiques. Dans ce cas cependant, ce comportement est spécifique à une classe d'informations bien définie.

Afin de lever cette restriction, des méthodes plus générales de spécification ont été définies et implantées, dans lesquelles la résolution d'un certain type de problème peut être spécifiée, indépendamment de l'application étudiée. Les systèmes de règles de production par exemple sont utilisés pour spécifier des méthodes générales de traitement d'informations applicables à une grande variété d'applications.

Ces méthodes peuvent être formalisées en faisant appel à la théorie du calcul des prédicats, ce qui a également contribué à allumer une polémique entre les logiciens et les adeptes des réseaux sémantiques sur la puissance respective de leurs outils préférés.

On rejoint ici le souci de formalisation qui a été à la base des travaux sur les bases de données et la logique [GAL81]. Leur importance est fondamentale dans la mesure où ils ont permis de développer toutes les recherches sur les bases de données déductives, les systèmes de bases de données experts et les SGBD avancés [ADI86].

2.1.2 Systèmes de Gestion de Bases de Données.

Un Système de Gestion de Bases de Données est un logiciel intégré permettant l'accès partagé, la modification cohérente et le stockage sûr de quantités importantes d'informations structurées.

A chaque modèle de données correspond une génération de SGBD qui en réalise les concepts de manière opérationnelle. Une étape supplémentaire a été franchie durant la période 1985 avec l'apparition de SGBD multi-média dans lesquels on peut également manipuler de l'information moins structurée, par exemple du texte, de la voix ou des images [TIG83].

De logiciels importants en termes de volume d'instructions de programmation, et destinés essentiellement à de gros calculateurs, on est passé à la prolifération de SGBD destinés à des micro-ordinateurs pour des prix commerciaux tout à fait modiques [MIC83].

Cette large diffusion, tout au moins dans certains milieux d'études et de recherche, a largement contribué au succès et à la notoriété de certains logiciels de SGBD. Cela s'est confirmé dans tous les domaines, aussi bien de la CAO que de la gestion [JUL86].

Parallèlement, son utilisation dans certains domaines de la recherche comme le traitement de la langue naturelle, mais aussi industriels, comme la prospection pétrolière, a contribué à la diffusion des techniques de l'Intelligence Artificielle.

2.1.3 Intelligence Artificielle et logique.

L'Intelligence Artificielle est une discipline de l'informatique qui a pour objectif l'étude et la mise en œuvre de mécanismes de représentation de connaissances, d'interprétation automatisée de celles-ci, et enfin de résolution automatisée de problèmes.

Le fondement de l'interprétation réside dans des méthodes de déduction logique. Le fondement de l'automatisation est quant à

lui la spécification des stratégies guidant les déductions et résolutions.

Les applications de l'Intelligence Artificielle sont nombreuses, et leurs domaines d'application immenses. Ils vont de la résolution de problèmes non-déterministes comme le diagnostic médical à la robotique, en passant par la traduction automatisée des langues naturelles et la vision par ordinateur.

Une des retombées concrètes de l'Intelligence Artificielle est constituée des Systèmes Experts. Ce sont des logiciels permettant de spécifier des méthodes de résolution de problèmes dans des domaines très précis (détection de pannes dans les réacteurs nucléaires, configuration de matériels informatiques, aide à la décision dans le domaine financier ...).

Ils sont basés sur des connaissances déclaratives, généralement des règles de production, dont l'activation sous certaines hypothèses, entraîne des conclusions. Elles sont spécifiées au préalable par des "experts" dans un domaine particulier d'application. Elles constituent ce que l'on a l'habitude d'appeler une base de connaissances. Ces règles sont interprétées afin de déduire des conclusions liées aux hypothèses ou aux observations concernant le problème à résoudre.

Les méthodes d'interprétation des connaissances font appel en général à des algorithmes non-déterministes qui permettent de prendre en compte l'imprécision, donc le degré de confiance, que l'on peut accorder à certaines conclusions. Les méthodes de résolution font largement appel à la logique mathématique. Elle doit cependant être adaptée à la nature instable et évolutive des connaissances que l'on peut associer à un domaine d'application particulier. Il faut savoir d'une part coder les connaissances. Mais il faut également savoir les faire évoluer. Par ailleurs, la plupart des problèmes non mathématiques peuvent avoir de multiples solutions, et il devient indispensable de savoir effectuer des choix parmi celles-ci. Il faut aussi pouvoir les justifier.

L'intérêt des méthodes de résolution de problèmes utilisant la logique vient de leur généralité. Elles sont applicables à tous les types particuliers de connaissances. Il est aussi possible d'appliquer des heuristiques différentes à des bases de connaissances différentes sans en connaître ni en altérer le contenu ou la sémantique.

Schématiquement, on explore un espace de recherche qui est souvent modélisé à l'aide d'un graphe. Partant d'un état initial et d'une base de connaissances qui contient les règles et l'expertise relatives au problème, le graphe permet d'aboutir à un état final matérialisant le but à atteindre, c-à-d une solution au problème à résoudre.

Le cheminement dans ce graphe, c-à-d les étapes successives de la résolution d'un problème, peut être spécifié avec des méthodes de déduction logique.

Le langage du calcul des prédicats du premier ordre a une puissance d'expression suffisante pour exprimer les problèmes courants et des propriétés formelles intéressantes.

2.2 Fondements de la logique.

On se contentera de rappeler brièvement les éléments significatifs de la logique mathématique et des méthodes de déduction dans le domaine qui nous intéresse. On insiste particulièrement sur le calcul des prédicats du premier ordre et sur les méthodes de résolution qu'il permet de spécifier, en particulier sur la résolution par réfutation et la programmation en logique [KLE71, KOW79].

2.2.1 Le calcul des prédicats du premier ordre.

Le calcul des prédicats du premier ordre auquel on se cantonnera ici est un cas particulier du calcul des prédicats qui comprend également :

- le calcul des propositions,
- le calcul des propositions avec quantification,
- le calcul des prédicats avec égalité.

Le calcul des prédicats du premier ordre est un système formel basé sur des axiomes et des règles de déduction qui permettent d'évaluer des formules.

Les formules sont construites à partir d'un langage L constitué de :

- variables notées x, y, z, ...
- connecteurs logiques : $\wedge$ (et), V (ou), ~ (non), -> (implique),
- les quantificateurs : (x) pour tout x et (E x) il existe x,
- parenthèses et symbole d'égalité (=),
- constantes notées : a, b, c, ...
- symboles de fonctions d'arité supérieure ou égale à 1 : f, g, h,...
- symboles de relations P, Q, R, ...

Un terme du langage est alors :

- une variable,
- une constante,
- f(t1, t2, .. tn) si t1, t2, ... tn sont des termes et f un symbole de fonction. f(t1, t2, .. tn)

Les formules sont construites à partir de termes et de formules atomiques.

Les formules atomiques sont de la forme :

- t1=t2 si t1 et t2 sont des termes,
- P(t1,...,tn) si P est un symbole de prédicat et les ti des termes.

Les formules sont soit :

- des formules atomiques,
- des formules entre parenthèses : (F),
- des formules composées : F1∧F2, F1∨F2, F1→F2 et ~F1,
- des formules avec quantificateur : (x) F(x), (E x)F(x)

où F, F1 et F2 sont des formules.

Afin de pouvoir utiliser le calcul des prédicats pour la résolution de problèmes, il est nécessaire d'associer au langage du premier ordre L une signification ou "interprétation".

Formellement, une interprétation I est une fonction qui :

- à chaque symbole de constante du langage associe un individu d'un domaine du discours non vide D,
- à chaque symbole de fonction associe une application du produit cartésien Dn dans D,
- à chaque symbole de prédicat une relation dans Dn.

L'évaluation d'une formule se fait alors dans le cadre de cette interprétation par "assignation" d'individus de D aux symboles de variables de la formule.

Une formule F est satisfaite dans une interprétation s'il existe au moins une assignation des variables qui permette de l'évaluer à vraie, ce qui est noté : |= F dans I.

Un modèle d'un ensemble de formules G est une interprétation I dans laquelle toutes les formules F de G sont satisfaites.

Une formule F est satisfaisable si et seulement si elle possède une modèle.

Une formule F est une "conséquence logique" d'un ensemble de formules G si tout modèle de G est un modèle de F, ce qui est noté : G |= F.

Si F est satisfaite dans toute interprétation du langage, alors F est dite "valide".

2.2.2 Déduction et résolution.

L'évaluation des formules se fait à l'aide de systèmes de déduction.

Un système de déduction est défini à partir d'un ensemble d'axiomes et des règles d'inférence.

Les axiomes sont :
- les tautologies,
- les axiomes d'égalité,

    - x=x,
    - x=y -> F(x1,x2,..,x,xi,..xj) -> F(x1,x2,..,y,xi,..xj),

- les axiomes de particularisation : (x) F -> F(t/x) où t/x est la substitution de t aux occurrences libres de la variable x.

La règle d'inférence la plus connue est sans doute celle de "modus ponens", telle que de A et A -> B on peut déduire B.

On a également besoin de la régle de "généralisation" qui dit que si x est non libre dans F alors de : F -> F' on peut déduire : F -> (x) F'.

Une formule F est dérivable s'il existe une séquence finie d'application des règles d'inférence qui permet, partant des axiomes, d'atteindre F : |- F. F est alors un théorème dans le système de déduction formé des axiomes précédents et des règles d'inférence.

La déduction peut être appliquée de manière opérationnelle à des formules mises sous forme clausale, c-à-d à des disjonctions de littéraux. Un littéral est une formule atomique ou sa négation.

La résolution par exemple s'applique en cherchant le plus grand unificateur entre deux littéraux et en substituant itérativement dans chaque littéral les variables unifiées.

Une méthode de résolution couramment utilisée s'appuie sur la "réfutation", c-à-d qu'on utilise la méthode d'unification-substitution pour prouver qu'une formule saine T est une conséquence logique d'un ensemble de formules F si la conjonction de F et de non T est contradictoire. Cette méthode est à la base des interpréteurs du langage Prolog.

2.3 Logique et bases de données.

2.3.1 Deux théories.

Des travaux théoriques importants ont été menés dans les années soixante quinze sur la formalisation des concepts des bases de données en logique du premier ordre [GAL84].

o Théorie du modèle.

Dans l'approche théorie du modèle, la base de données est considérée comme une interprétation (au sens défini en 2.2.1) de la théorie formée du schéma de la base et des contraintes d'intégrité.

Les faits de la base de données sont les individus du domaine associé à l'interpétation.

Les contraintes d'intégrité sont des propriétés générales qui constituent des axiomes de cette théorie.

La plupart des premiers travaux concernant l'évaluation de questions et de contraintes d'intégrité dans les bases de données sont basés sur cette approche [NIC82].

La base de données est considérée comme un modèle (au sens de 2.2.1) de la théorie.

On voit immédiatement que la vérification de contraintes d'intégrité dans ce cas est liée au processus de déduction et aux axiomes de la théorie. Il est donc nécessaire d'étendre la théorie si l'on veut tenir compte de valeurs nulles ou incomplètes [DEM85, REI84].

Une question adressée à la base de données est dans ce cas une formule ouverte, c-à-d qui contient des variables non quantifiées ou libres. La réponse à une question est l'ensemble des individus de la base qui rendent la formule vraie lorsqu'ils sont substitués aux variables libres.

o Théorie de la preuve.

La théorie du modèle est différente de l'approche adoptée par les logiciens, qui considèrent les données et les contraintes dans leur ensemble comme les axiomes propres d'une théorie du premier ordre.

Dans ce contexte, toute question adressée à un SGBD est un théorème que l'on doit prouver à partir des règles d'inférence et des axiomes de la théorie. C'est la théorie de la preuve.

Elle est généralisable aux valeurs nulles et incomplètes. Elle permet également de prendre en compte les notions d'héritage de propriétés et d'aggregation [REI84].

Si L est un langage défini selon les règles syntaxiques du paragraphe 2.2.1, et si G est un ensemble de formules (faits atomiques ou contraintes d'intégrité), une théorie du premier ordre T est constituée des formules suivantes :

- l'axiome de fermeture du domaine :
  si a1, a2, ... an sont les constantes de L, on a :
  (x) (x=a1) V (x=a2) V ... V (x=an)
- l'axiome des noms uniques, c-à-d (i,j) ai =/= aj,
- les axiomes définissant l'égalité (réflexivité, commutativité et transitivité) et la substitution des termes égaux,
- les axiomes de complétude pour chaque prédicat de L :
  pour tout prédicat n-aire P de L, si P(x1, x2, ..., xn) c W, alors :
  (xi) 1<=i<=n P(x1, x2, ..., xn) => (x1=a1) ∧ (x2=a2) ... ∧ (xn=an) V (x1=a12) ∧ (x2=a13) ... ∧ (xn=anp)
- les formules de G.

Le système de déduction est formé des axiomes logiques définis précédemment, des axiomes propres de la théorie T (faits élémentaires et contraintes d'intégrité) et des règles d'inférence modus-ponens et généralisation.

Une déduction dans cette théorie est alors une séquence finie d'application des règles d'inférence à partir des axiomes logiques et de G qui permet de prouver une formule F : G |- F.

On peut montrer que si F est un théorème dans une théorie T, alors F est vraie dans tout modèle de T : T |- F => T |= F.

Cette théorie a un modèle unique qui correspond précisément à l'interprétation du schéma de la base et des contraintes par la base de données dans l'approche théorie du modèle.

Elle peut être étendue aux valeurs nulles ou inconnues, en éliminant les axiomes des noms uniques liés aux constantes de Skolem et en ajoutant les axiomes de fermeture et de complétude qui les concernent [REI84].

Dans ce qui suit, on utilise cette approche pour contrôler

de manière unifiée les opérations de mise à jour sur les informations et sur les schémas d'une base de données (Chapitre VI). On cherche à caractériser à tout instant l'ensemble des formules vraies de la théorie et on se limite à une description incomplète des objets. Ceci évite le recours à des techniques plus complexes comme la logique modale ou la théorie du modèle étendue aux valeurs nulles pour contrôler la cohérence dans les bases de données.

2.3.2 Accès aux informations.

L'accès aux informations à l'aide de la logique peut être utilisé de différentes manières :

- pour déduire des informations implicites,
- pour simplifier et contrôler des contraintes d'intégrité [NIC79],
- pour optimiser les accès aux données [WAR81],
- pour étendre les fonctions d'un système de déduction [VAS83].

Une première étape concrète a été franchie avec l'implantation de mécanismes de coopération entre les SGBD et des mécanismes de déduction [NGU83, NGU84]. Cette approche est la plus simple à mettre en œuvre. Elle a débouché sur un grand nombre d'applications opérationnelles [NGU84b, PAR83, VAS83, WAL83]. Elle s'appuie sur deux constatations :

- d'une part les techniques d'accès aux informations dans les systèmes déductifs sont extrêmement pauvres,
- d'autre part les mécanismes correspondant sont en général très efficaces dans les SGBD, mais les possibilités de déduction inexistantes ou très limitées.

On a donc naturellement cherché à coupler les deux ensembles pour tirer parti de leurs avantages respectifs. Ce couplage présente des avantages en ce qui concerne la représentation d'informations déduites, le calcul d' opérateurs transitifs et/ou récursifs comme la fermeture d'un ensemble répondant à un critère de filtrage, et l'évaluation de contraintes d' intégrité.

Pour suivre une démarche traditionnelle en matière de SGBD, elle peut être implantée selon trois niveaux : interne, externe et conceptuel.

o Niveau interne.

Le premier niveau concerne l'accès direct aux informations contenues dans une base de données par un système déductif

extérieur au SGBD. On considère alors les faits élémentaires stockés dans la base de données comme des assertions. Il convient dans ce cas d'optimiser les accès physiques afin d'éviter des allers-retours incessants entre le SGBD et le système déductif [WAR81]. C'est par exemple l'objectif des méthodes dites compilées de requêtes exprimées en logique des prédicats, dans lesquelles l'accès proprement dit aux faits élémentaires est repoussé aussi tard que possible dans l'évaluation de la requête. C'est par ailleurs une nécessité dans les méthodes dites de couplage fort [VAS83].

Dans les méthodes de couplage faible, l'efficacité est obtenue au prix de l'extraction des données avant évaluation de la requête déductive. Ceci n'est possible malheureusement que si l'on connait a priori les données dont on aura besoin. Cette approche est donc reservée à des problèmes particuliers et son inconvénient majeur est d'interdire toute évaluation dynamique d'informations. Elle interdit par exemple la propagation de contraintes après mise à jour d'informations dépendantes. C'est pourquoi l'implantation d'un accès au niveau interne a été reservée ici à la réalisation d'une interface logique sur des bases de données réparties (Chapitre IV).

o Niveau externe.

La coopération au niveau externe, c-à-d au niveau de l'utilisateur ou des applications, nécessite la réalisation d'interfaces spécifiques. Dans le cas où l'on veut intégrer des outils existants et les coupler à une base de données, par exemple un système expert en modélisation, il convient de s'appuyer sur les méthodes de couplage faible car, malgré leurs limitations, ce sont les seules a permettre une certaine indépendance logique entre les données et les applications.

o Niveau conceptuel.

La coopération au niveau conceptuel vise à s'affranchir des problèmes d'implantation pour s'attacher à la représentation de la sémantique des traitements [NGU83, NGU84, NGU85].

On entend ici par sémantique des traitements la spécification :

- des objets manipulés,
- de leurs propriétés statiques,
- des traitements appliqués,
- des contraintes dynamiques qu'ils doivent respecter.

Elle nécessite des moyens de représentation et de traitement évolués d'informations complexes [ADI84, ADI85, ADI86]. Ses applications opérationnelles sont nombreuses. La CAO en fait partie. Elle nécessite en effet des moyens de calcul puissants, par exemple pour la simulation électrique de circuits intégrés ou le calcul de profils aérodynamiques. Elle nécessite également des moyens de représentation et de stockage adaptés à des objets complexes comme les circuits VLSI. Leur représentation instantanée comporte en effet des versions multiples correspondants à des niveaux de spécification différents (fonctionnel, logique, électrique, etc).

Dans ce contexte, le rôle du SGBD est de fournir des outils de stockage adaptés aux objets complexes. Le rôle du système déductif est de pourvoir aux méthodes de vérification de contraintes, par exemple :

- d'équivalences entre représentations,
- d'équivalences entre versions,
- de respect des règles de conception propres à chaque représentation du circuit.
- à plus long terme, génération automatisée d'une représentation à partir d'une autre, par exemple logique à partir de fonctionnelle.

Dans le cadre de cette étude, une expérimentation a été entreprise en collaboration avec le CNET pour la CAO de circuits VLSI. Contrairement à d'autres approches, on n'y vise pas l'extension de mécanismes de déduction avec les fonctionnalités des SGBD.

On s'est putôt attaché à étendre les fonctionnalités d'un SGBD avec des outils de définition, manipulation et évaluation de la sémantique d'applications avancées qui mettent en jeu des données complexes [NGU86a, NGU86b].

Le tableau de la Figure 2.1 résume les trois niveaux d'accès possibles entre un système déductif et un SGBD relationnel. Une réalisation du niveau interne en a été faite par Pascale Winninger sur le SGBD MICROBE [NGU84b]. La spécification et l'implantation du niveau conceptuel est l'objet de la Thèse de Judith Olivares [OLI86]. L'aspect externe fait l'objet d'un projet mené conjointement par le Laboratoire de Génie Informatique à l'IMAG et le Département Recherche en Conception Assistée du CNET Grenoble.

	système déductif	SGBD relationnel	
niveau interne	Assertions	relations n-uplets	accès informations
niveau conceptuel	Définitions et règles déducti.	modèle de données	représentation des concepts
niveau externe	Contraintes de l'application	traitements et requêtes	programmes d'application

Figure 2.1 Liens entre système déductif et SGBD.

2.3.3 Mise en œuvre.

A chacun des trois niveaux d'accès précédents correspond une méthode d'implantation différente des traitements.

- Niveau interne.

Au niveau interne correspond l'accès aux données physiques à travers leurs structures de représentation interne. Les traitements sont alors exprimés dans la logique des accès à ces structures de données : par exemple la notion de "balayage" d'une relation dans MICROBE. C'est ce qui a été réalisé son interface avec le langage Prolog [NGU84b]. Dans ce cas, le schéma de la base de données est traduit en Prolog. Il sert à réécrire les requêtes déductives en termes d'opérateurs de l'algèbre relationnelle. Ceux-ci sont directement interprétés par le SGBD sous la forme d'une arborescence algébrique (Figure 2.2). Dans cette réalisation toutefois les requêtes récursives ne sont pas interprétables [BAN85, DEL86, GAR86].

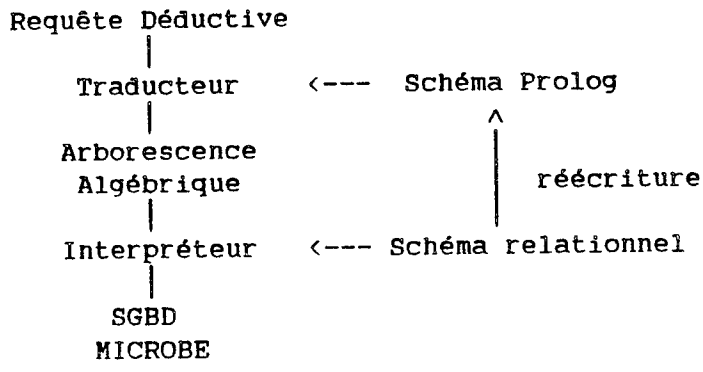


Figure 2.2 Interface Prolog-Microbe.

L'ensemble des fonctions classiques d'un SGBD y est réalisé à l'aide des fonctions suivantes, qui sont réalisées par des clauses Prolog :

- creer_base(B) pour créer une base dont le nom est contenu dans la variable B,
- ouvrir_base(B) pour l'ouvrir,
- car_base, qui permet d'obtenir la liste des relations contenues dans la base de données que l'on manipule,
- fermer_base(B) pour la fermer,
- détruire_base pour la supprimer,
- creer_rel(R) pour créer une relation dont le nom est contenu dans la variable R,
- creer_relvirt(R) pour créer une relation virtuelle, c-à-d définie à partir de relations existantes par un ensemble d'opérations relationnelles,
- dest_rel(R) pour détruire une relation,
- creer_att(R) pour créer un constituant de relation,
- dest_att(R, [liste_de_constituants]) pour détruire une liste de constituants d'une relation,
- att_rel(R) pour obtenir les caractéristiques des constituants d'une relation,
- insérer(R) pour insérer des n-uplets dans une relation,
- supprimer(R,[liste_de_conditions]) pour détruire des n-uplets dans une relation,
- modifier(R, [liste_de_constituants] , [liste_de_valeurs] , [conditions]) pour mettre à jour des constituants de relations,
- seprojo([constituants],[conditions],[résultat]) qui est un opérateur combiné de sélection-projection-jointure. Il évalue les conditions spécifiées sur des relations, en extrait les constituants précisés en paramètres et produit un résultat. Il permet avec une syntaxe unique d' exprimer la plupart des requêtes simples. Il inclut les fonctionnalités de l'opérateur "set_of" qui permet à Prolog de travailler de manière ensembliste. Il interprète les arguments à l'instar d'une arborescence d'opérateurs de l'algèbre relationnelle.

Supposons par exemple que la base de données soit construite sur le schéma suivant :

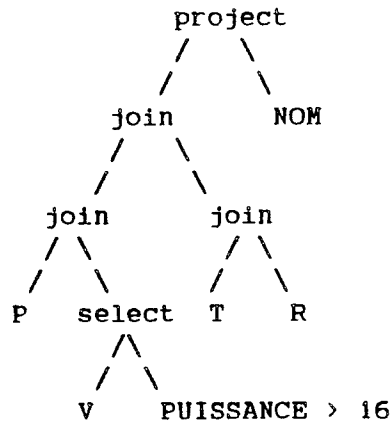
```
PERSONNE : relation P(NP, NOM),
VOITURE  : relation V(NV, NP, PUISSANCE),
TRAVAIL  : relation T(NP, QUALIF, LIEU),
RESIDENT : relation R(NP, VILLE, RUE).
```

où NP est le numéro d'une PERSONNE ayant un ceratin NOM, NV le numéro de sa voiture, dotée d'une certaine PUISSANCE. Cette personne travaille en un certain LIEU, a une certaine QUALIFication, et habite une certaine RUE d'une certaine VILLE.

La question : "Quel sont les noms des personnes qui travaillent sur leur lieu de résidence et qui sont propriétaires d'une voiture de plus de 16 chevaux ?" est évaluée grâce à la clause suivante :

seprojo([NOM], [P(NP, NOM), R(NP, VILLE,_), T(NP,_, VILLE), V(_, NP, PUISSANCE), PUISSANCE>16], [RESULTAT]).

où [RESULTAT] est la liste des NOMs qui répondent à la question. Elle correspond à l'arborescence ci-dessous :



Une approche similaire est possible dans les systèmes de représentation des "connaissances". On peut y intégrer au niveau interne les opérateurs de l'algèbre relationnelle pour le stockage et l'accès aux informations [BEN85].

- Niveau externe.

La mise en œuvre au niveau externe, c-à-d de l'application ou de l'utilisateur, nécessite la formalisation des concepts du modèle de données utilisé. Les contraintes qui lui sont liées, par exemple les règles de conception de schémas, peuvent alors être implantées dans ce même formalisme du premier ordre. Cette approche est à la base des systèmes experts d'aide à la conception de bases de données comme SECSI [BOU85]. Elle repose donc sur le niveau conceptuel.

Une approche plus élémentaire consiste a fournir à l'utilisateur des opérateurs étendus, comme la fermeture transitive, pour augmenter les fonctionnalités d'un SGBD classique [STO85].

- Niveau conceptuel.

En dehors de la conception de schémas, la mise en œuvre de la coopération SGBD-Système déductif au niveau conceptuel peut servir à la définition et manipulation de la sémantique des informa-

tions pour une application particulière.

C'est l'approche qui a été utilisée pour l'extension du SGBD généralisé TIGRE avec comme objectif initial la vérification de contraintes dans des applications bureautiques ou de CAO.

Ceci implique la résolution de cinq problèmes différents :

- la formalisation des concepts du modèle de données utilisé,
- la traduction du schéma de la base de données dans ce formalisme,
- la spécification des contraintes associées aux informations dans ce formalisme,
- la traduction des requêtes.
- l'évaluation de ces requêtes.

o Formalisation des concepts du modèle de données.

Pour rester cohérent avec la démarche unifiée qui a été adoptée, on formalise les concepts du modèle de données utilisé en logique des prédicats du premier ordre.

Ceci veut dire que l'on définit un méta-modèle qui contient toutes les spécifications des objets que l'on peut définir (entités, associations, rôle, aggrégation, etc), ainsi que toutes les opérations licites sur ces objets (instanciations, mise à jour, suppression d'entités, d'associations, règles d'héritage de propriétés, etc). Ceci est évidemment un préalable à toute étape ultérieure. Une réalisation pour le modèle de données généralisées du SGBD TIGRE est étudiée de manière détaillée dans [OLI86]. On en rappelle ici quelques éléments de base en utilisant des clauses de Horn.

```
type (X) :-              type_de_base(X) ; type_dérivé(X) ;
                         type_construit(X) ; classe(X).
type_de_base (X):-       entier(X) ; chaîne(X) ; booléen(X) ;
                         intervalle(X).
type_dérivé (X) :-       type_de_base(X).
type_construit(X):-      tableau(X) ; enregistrement(X) ; document(X).
classe(X) :-             entité(X) ; association(X) ;
                         généralisation(X) ; aggrégat(X).
généralisation(X):-      spécialisation(X) ; union(X).
aggrégat([]).
aggrégat(X) :-           X is [Y|L], not (aggrégat(Y)), type(Y),
                         aggrégat(L).
```

Une description plus complète en est donnée au Chapitre IV : Modélisation.

o Traduction du schéma de la base de données.

Cette formalisation permet de contrôler la définition et toute modification d'un schéma de la base de données. Toute définition de données est traduite en une requête adressée à un sous-système de validation (Chapitre VI).

Par une déduction appropriée, cette requête est validée ou non selon qu'elle est satisfaisable à l'aide du méta-modèle ou non. Si elle est validée, son résultat est ajouté (sous forme d'une nouvelle définition ou d'une modification du schéma existant) à une base de connaissance qui contient la formalisation du schéma de données utilisé par l'application. Cette démarche est illustrée Figure 2.3.

La définition du schéma est contrôlée à l'aide d'un analyseur syntaxique qui est généré automatiquement à partir de la grammaire du langage de définition (Figure 2.4).

On sait que si cette grammaire est de type LL(1), c-à-d hors contexte et déterministe, on peut décrire toute phrase du langage à l'aide d'un ensemble de règles non récursives. C'est le cas de la grammaire du modèle TIGRE. On a donc réalisé un analyseur syntaxique à partir de la définition formelle de ce langage sous forme de Backus (forme BNF, ou Backus-Naur Form). Ceci étant classique dans la théorie des langages, on peut ajouter que cette génération est immédiate lorsque les règles sont écrites sous forme de clauses de Horn [NGU83].

Une entité "employé" définie dans le modèle TIGRE par

```
Define entity employé
   Key numéro     : integer end
       nom        : string (20)
       salaire    : montant
       catégorie  : (ingénieur, secrétaire)
       adresse    : type_adresse
       contrat    : type_contrat.
```

est traduite sous la forme :

```
entité(employé).
constructeur (employé, NUM, NOM, SAL, CAT, ADR, CONTR) :-
    type          (employé),
    key           (employé, NUM),
    integer       (NUM),
    string        (NOM),
    montant       (SAL),
    catégorie     (CAT),
    type_adresse  (ADR),
    type_contrat  (CONTR).
```

où les types integer, string, ..., type_adresse et type_contrat sont prédéfinis. Par exemple :

```
type_adresse (X) :- X is [No|L], integer(No), lieu(L).
lieu(L)          :- L is [Voie|Ville], string(Voie),
                    string(Ville).
```

et :

```
montant(X)  :- integer(X), X >= smic, X <= plafond.
```

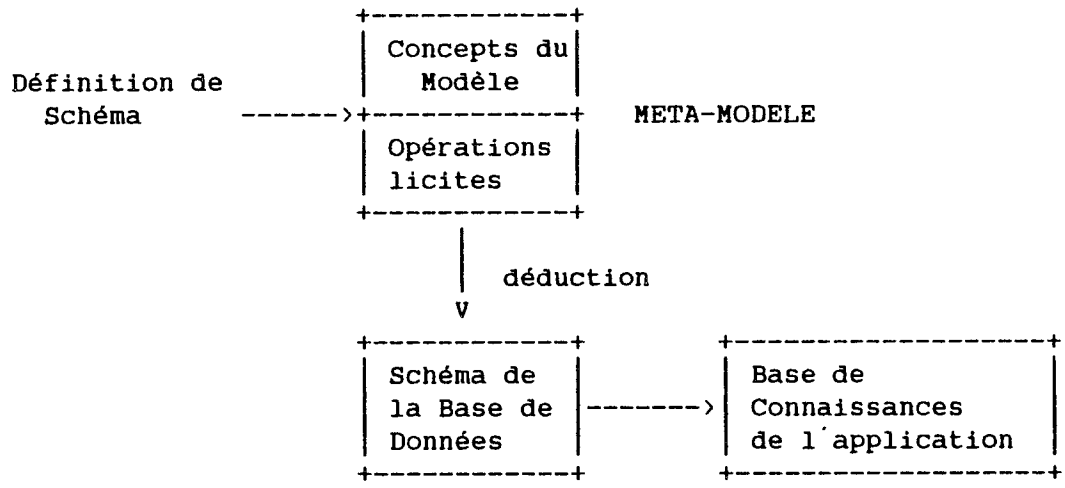


Figure 2.3 Méta-modèle et construction de schéma.

o Spécification des contraintes associées aux informations.

La spécification des contraintes est dans cette étape restreinte aux dépendances entre données (fonctionnelles, multi-valuées, d'inclusion), et aux cardinalités des rôles des associations qu'elles expriment dans le cas du modèle de données généralisées TIGRE. Ce sont les contraintes existentielles. Elles sont complétées par les contraintes qui spécifient la propagation des contraintes de mise à jour. Ce sont les contraintes opérationnelles.

Toutes doivent être également spécifiées en logique des prédicats du premier ordre et constituent une partie de la base de connaissances de l'application.

Si le schéma "employé" est complété par la définition de son lieu de travail par :

```
Define lieu_de_travail between
       EMPLOYE : poste (1,*)
       SITE    : bureau(1,1)
       DEPUIS  : date
```

qui est transformé en :

```
association (lieu_de_travail,employé,site).
rôle        (lieu_de_travail,employé,poste,1,*).
rôle        (lieu_de_travail,employé,bureau,1,1).
constructeur(lieu_de_travail(EMP,SITE,DATE)) :-
             association(lieu_de_travail),
             employé(EMP),
             bureau(SITE),
             date(DATE).
```

la définition de la contrainte : "Tout ingénieur doit résider sur son lieu de travail", est définie par :

```
Define constraint localisation between
       employé where catégorie = 'ingénieur'
       and
       lieu_de_travail where site is bureau of employé
       enforce
       site = adresse.
```

qui est transformé en (les variables anonymes sont notées "_") :

```
constraint(localisation, employé, lieu_de_travail) :-
           employé(NUM, _, _, ingénieur, ADR, _),
           lieu_de_travail(NUM, ADR, _).
```

Dans la mesure où les contraintes sémantiques de l'application peuvent aussi être exprimées sous forme de règles dans le calcul des prédicats du premier ordre, elles sont également incluses dans cette base de connaissances. Ce sont par exemple les règles de conception de circuits intégrés en fonction de la technologie utilisée (MD-MOS, etc).

o Traduction des requêtes.

Les contraintes sémantiques et en particulier les règles de propagation des mises à jour servent à contrôler l'exécution des requêtes. Elles sont analysées et modifiées en conséquence avant d'être évaluées. Ceci est réalisé par l'adjonction d'un préfixe et d'un suffixe à chaque requête. Ils contiennent les appels nécessaires à l'évaluation de préconditions et à l'activation des procédures de propagation de contraintes.

Supposons que l'on veuille modifier le lieu de travail de l'ingénieur Martin. Ceci est exprimé sous la forme :

```
replace  bureau := 'Paris'
  from   lieu_de_travail
  where  nom of employé of poste = 'Martin'.
```

qui est transformé en :

```
assert(lieu_de_travail(E, S, _) :- association(lieu_de_travail),
                                   employé(E, Martin,_,_,_,_,_),
                                   bureau(Paris),
                                   date(_)).
```

L'analyse des contraintes définies dans le schéma montre l'existence de la contrainte de localisation entre les employés et les lieux de travail. La requête est donc suffixée par l'évaluation du prédicat "constraint(localisation, employé, lieu_de_travail)" :

```
assert(lieu_de_travail(E, S, _) :- association(lieu_de_travail),
                                   employé(E, Martin,_,_,_,_,_),
                                   bureau(Paris),
                                   date(_).
localisation, employé,lieu_de_travail.
```

Par renommage des variables et substitutions, ceci devient :

```
assert(lieu_de_travail(E, S, _) :- association(lieu_de_travail),
                                   employé(E, Martin,_,_,_,_,_),
                                   bureau(Paris),
                                   date(_)).
constraint(localisation,employé,lieu_de_travail) :-
                                   employé(E,Martin,_,ingénieur,S,_),
                                   lieu_de_travail(E, S,_).
```

L'évaluation à postériori de la contrainte permet de contrôler que le lieu de résidence de l'ingénieur Martin est toujours identique à son nouveau lieu de travail. Si ce n'est pas le cas, la contrainte est évaluée à 'faux'. Cette évaluation étant immédiate, les actions correctives peuvent être entreprises sans délai.

Il faut noter que les transformations effectuées sont de nature purement synataxiques : préfixage, suffixage, renommage et substitutions de variables. Ce procédé ne dépend donc ni de la nature des requêtes, ni du schéma de la base de données, encore moins des valeurs des données.

Une approche plus "manuelle" peut être obtenue en précisant explicitement dans la définition des contraintes les conditions dans lesquelles elles s'appliquent. Par exemple :

- les types d'opérations qui les concernent (insertion, suppression, mises à jour),
- la nature de la contrainte (pré ou post conditions).

Cette approche mise en œuvre par une interface ad-hoc entre le sous-système de validation et l'application est décrite dans [OLI86]. Une automatisation complète des transformations doit faire appel à une analyse sémantique plus évoluée qui n'est pas abordée ici.

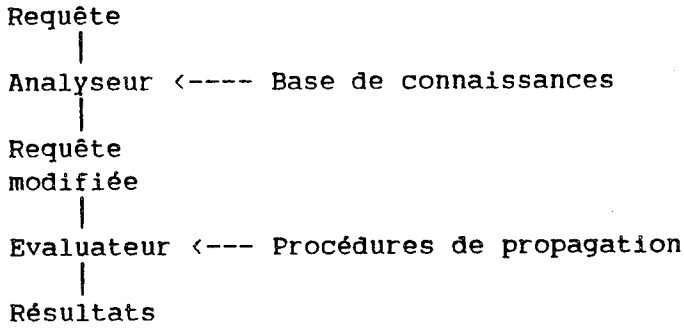


Figure 2.4 Modification de requête.

o Evaluation des requêtes.

L'évaluation des requêtes se fait en quatre étapes :

- transformation par préfixage et suffixage de contraintes,
- évaluation des contraintes préfixées,
- exécution de la requête initiale,
- évaluation des contraintes suffixées.

L'accès à des objets complexes tels que des documents, des graphiques ou de l'information vocale, se fait par l'intermédiaire de procédures d'accès ad-hoc. Leur gestion et leur stockage matériel ne sont pas pris en compte ici. Seule sont formalisées la structure des informations et leur contrintes, c-à-d leur description et leur dynamicité.

L'évaluation des contraintes nécessite l'implantation d'un mécanisme adéquat d'accès aux données par les procédures de vérification qui les réalisent. On ne détaillera pas ici ce type de communication. On peut mentionner simplement que contrairement à une opinion largement répandue, il existe des systèmes déductifs qui permettent cette communication dans de bonnes conditions.

CHAPITRE III

APPLICATIONS AVANCEES

3.1 Caractérisation et besoins.

3.1.1 Gestion intégrée de données.
3.1.2 Gestion de données réparties.

3.2 Exemple : la Conception assistée par ordinateur.

3.2.1 Systèmes intégrés de CAO.
3.2.2 Représentations multiples.
3.2.3 Versions multiples.
3.2.4 Cohérence et complétude.
3.2.5 Répartition.

3.3 Conclusion.

CHAPITRE III

"Ad augusta per angusta"

APPLICATIONS AVANCEES

Parmi les applications de l'informatique, on commence à distinguer celles qui se contentent de traitements déterministes, comme les applications de gestion, et celles qui nécessitent des traitements plus évolués.

Parmi ces dernières, on peut citer la bureautique, le génie logiciel ou la conception assistée par ordinateur. Elles se différencient fondamentalement des premières par leur démarche. A un type de problème donné correspond dans ces domaines une multitude de solutions possibles que l'on doit déterminer au moins partiellement et parmi lesquelles on doit opérer un choix.

C'est le cas de la construction de logiciel par exemple dans laquelle le découpage et la connexion de modules entre eux peuvent être définis de manière a priori très variée. C'est également le cas de la conception assistée par ordinateur où l'agencement de puces sur un circuit intégré peut être fixé de façon à minimiser la consommation électrique, la complexité ou la place.

On caractérise dans ce chapitre les spécificités de ces applications en prenant comme exemple la conception de circuits VLSI. Ce choix est motivé par le fait qu'on a pu expérimenter de façon concrète la mise en œuvre d'un SGBD relationnel et son couplage avec des techniques de l'Intelligence Artificielle [ADI84, ADI85, RIE86].

3.1 Caractérisation et besoins.

A titre d'exemple, on considère ici trois domaines d'application où les problèmes rencontrés et les solutions relèvent de moyens non déterministes.

Ce sont d'une part la bureautique, d'autre part les applications de Conception. Ce terme recouvre à la fois la Conception Assistée par Ordinateur et le Génie Logiciel, dont une partie des exigences recoupe celles de la CAO.

Pour fixer les idées, on prendra comme exemple les travaux liés aux projets TIGRE [TIG83], CVT [JUL86], CONCERTO [CAZ83] et ADELE [EST84].

3.1.1 Gestion intégrée de données.

Dans toutes ces applications, on retrouve un ensemble de besoins communs en :

- modélisation des objets manipulés,
- modélisation de l'environnement de travail,
- spécification des protocoles de communication,
- spécification des protocoles du dialogue individu-système,
- gestion de représentations multiples des objets,
- répartition logique et physique des outils et des informations.

A des degrés divers et avec une terminologie propre, ces problèmes sont traités dans chacun de ces projets.

o Modélisation des objets.

Le terme objet est utilisé ici comme synonyme de "donnée". Ce peut être un document en bureautique, un programme source en génie logiciel ou encore un circuit intégré en CAO.

Cette notion est peu habituelle dans le domaine des bases de données où l'on s'est concentré dans le passé sur la modélisation de données factuelles structurées et simples, par opposition à des objets "complexes" ou "volumineux" : un circuit intégré ou un document multi-média.

En masquant la gestion des informations en mémoire secondaire et en fournissant des langages de haut niveau, les SGBD ont contribué à la simplification des applications qui mettent en œuvre de tels objets. La complexité et la variété de leurs structures a cependant mis en lumière leurs limites.

Tout d'abord, il est impératif de pouvoir modéliser des structures logiques qui sont hiérarchiques. Que ce soit la structure interne d'un document en chapitres, sections et paragraphes, d'un logiciel en modules, interfaces et corps de procédures, ou d'un circuit intégré en représentations, alternatives, versions et composants. Ceci est un minimum, car les besoins réels requièrent parfois des structures en réseau (en particulier pour tout ce qui a trait à l'héritage

de propriétés).

Ceci a immédiatement mis en défaut tous les SGBD relationnels. Pour répondre à ce besoin, des extensions du modèle de données relationnel ont été proposées [JUL86, LOR84]. D'autres solutions sont réalisées ou envisagées :

- Le projet CONCERTO utilise une représentation à base de frame.
- Le projet TIGRE utilise une représentation typée des objets bâtie sur le modèle relationnel classique. Les documents y ont une structure interne arborescente.
- Le projet ADELE utilise une représentation interne en réseau spécifique.
- D'autres approches conceptuelles sont proposées par ailleurs sans éléments de réalisation [KIM85, LEO83].
- Le projet CVT utilise une représentation interne en réseau bâtie elle aussi sur le modèle relationnel (Figure 3.1). Les informations liées entre elles sont associées à l'aide d'un lien matérialisé par les adresses internes des objets correspondant. Ces liens ont une structure d'anneau. On peut définir un nombre quelconque de liens sur les relations. Ceci permet d'évaluer très rapidement des questions du type

    - "Quels sont tous les éléments connectés à tel objet?"
    - "Quels sont tous les composants de l'objet untel?"
    - "selectionner parmi les composants de tel objet ceux qui répondent à des critères de selection particuliers".

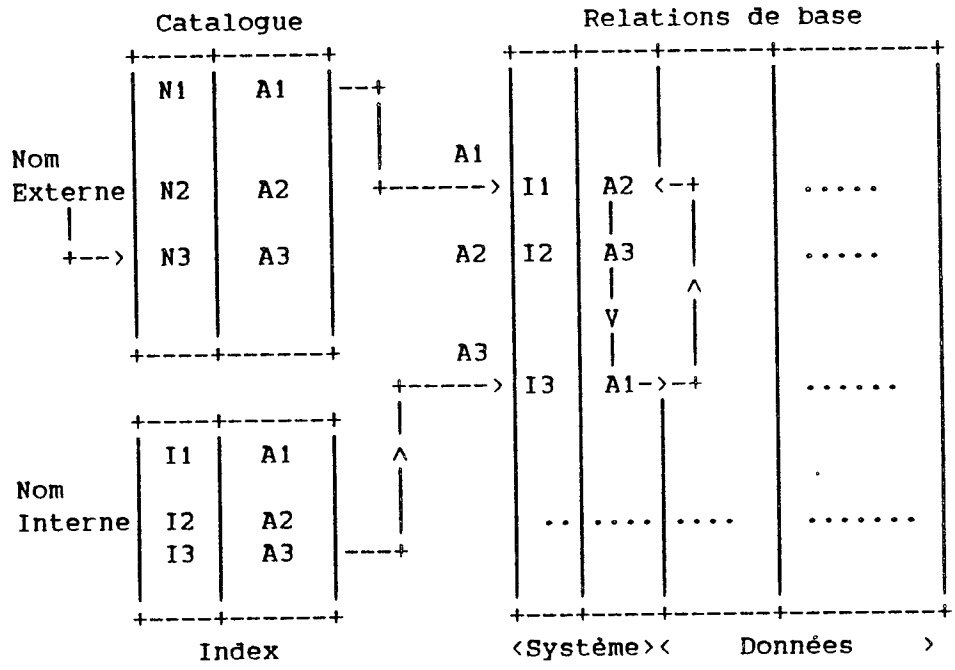


Figure 3.1 Structure des données CVT.

Dans l'exemple de la figure 3.1, on suppose que le composant de nom externe N1 utilise les composants N2 et N3 (Figure 3.2). Les noms internes I1 , I2 et I3 leur sont alloués. Ils sont utilisés pour la création d'index. La relation père-fils est matérialisée de façon interne par chaînage entre les données correspondantes. Ce sont ici les n-uplets d'adresse physique A1, A2 et A3. Ils peuvent appartenir à des relations différentes. Ils sont constitués de données propres à l'application et d'informations propres au système comme les identificateurs internes et les liens.

Figure 3.2 Relation de composition.

3.1.2 Gestion de données réparties.

Indépendamment des problèmes de réalisation, la conception des systèmes modernes de traitement de l'information repose sur la coopération de multiples acteurs. Elle fait appel à la répartition à tous les niveaux de communication. On distingue ici trois types de répartition qui se superposent. Ce sont la :

- répartition logique,
- répartition physique,
- intégration d'outils.

Dans un système comme CONCERTO, on trouve de façon omniprésente cette distribution des fonctions [CAZ83]. L'architecture globale comprend :

- un système de communication,
- un système de commande,
- un système d'information,
- un système de dialogue.

Le système de communication assure les échanges entre acteurs (Figure 3.3). Le système de commande est chargé du contrôle du bon déroulement des opérations et du comportement des acteurs. Il assure aussi des fonctions de décision propres à l'application, comme le lancement de programmes particuliers à la suite d'évènements prédéfinis. Le système de dialogue contrôle les échanges individu-système. C'est lui qui met en œuvre les interfaces de dialogue homme-machine ou de "médiateur" dans le projet ADELE. Enfin le système d'information gère les données de base, les documents, les programmes et les ressources du système.

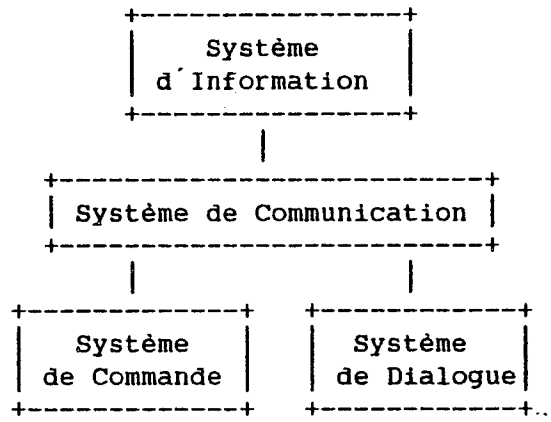


Figure 3.3 Architecture générale CONCERTO.

3.2 Exemple : la Conception Assistée par Ordinateur.

La Conception Assistée par Ordinateur repose aujourd'hui sur la solution de trois problèmes distincts. Ce sont :

- la définition d'un processus de conception, c-à-d d'un enchaînement correct des opérations de construction d'un objet,
- leur représentation et leur manipulation graphique in-

teractive,
- leur modélisation et leur stockage.

On ne s'intéresse ici qu'au dernier aspect.

Ce que l'on peut qualifier de première génération de systèmes de CAO a consisté en la juxtaposition d'un ensemble de programmes enchaînés de manière adéquate pour contrôler la validité du travail des concepteurs, ou leur fournir une aide spécifique dans la modélisation des objets, par exemple à l'aide d'outils graphiques interactifs.

L'interaction des programmes entre eux impliquait la génération de fichiers intermédiaires, puis l'"extraction" de ceux-ci des informations nécessaires au suivant. A chaque étape, le résultat d'un programme pouvait être examiné par le concepteur pour évaluer la validité de ses choix.

Vers 1970 sont apparus les premiers outils "réellement" interactifs de CAO, et l'on a assisté à une floraison de systèmes dans lesquels chaque étape de conception peut être directement pilotée par un utilisateur. L'inconvénient majeur de ces systèmes a été de conserver la fragmentation du processus de conception en étapes successives et itérées, et dépendantes les unes des autres. Il en est résulté une spécialisation des interfaces, et une dépendance complète des informations et de leur représentation vis-à-vis des outils sous-jacents.

On peut prendre comme exemple les systèmes de CAO de circuits intégrés, dans lesquels on a l'habitude de distinguer les étapes de (Figure 3.4) :

- spécifications fonctionnelles,
- spécifications logiques,
- spécifications électriques,
- spécifications d'implantation.

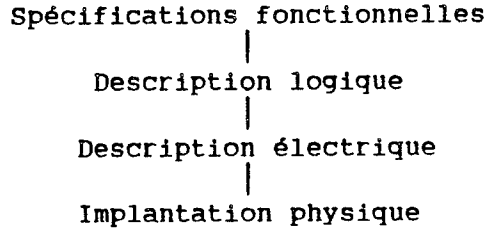


Figure 3.4 Etapes de conception d'un circuit VLSI.

Dans un système comme CASSIOPEE par exemple, chacune de ces étapes a une interface graphique particulière [LEC84]. Ceci implique, en plus des transformations requises pour le passage d'une étape à l'autre, une transformation particulière entre chaque niveau de représentation du circuit et son affichage sur

un terminal graphique. D'où également la nécessité d'un logiciel très performant pour assurer cette transformation et une grande redondance de l'information qui rend très délicat le maintien de la cohérence entre les diverses représentations.

Ces inconvénients résultent pour la plupart de l'héritage de méthodes de conception relativement anciennes.

Depuis la decennie 80, l'Intelligence Artificielle et les méthodes de calcul non-déterministe ont permis de jeter les bases de systèmes de CAO basés non plus sur des méthodes analytiques, comme c'était le cas jusqu'alors avec les cycles conception-vérification, mais sur des méthodes de synthèse basées sur les spécifications fonctionnelles des objets à concevoir [FIN85, HOW84, ZAU83].

Ainsi en est-il des compilateurs de silicium par exemple, et de la génération de circuits PLA à partir des équations booléennes spécifiant leur fonctionnalités.

C'est l'une des raisons qui, conjuguée à la diffusion de logiciels évolués de gestion de bases de données, a rendu possible la création de systèmes intégrés de CAO.

3.2.1 Systèmes intégrés de CAO.

Deux évolutions majeures des systèmes de CAO ont permis de les classer dans une deuxième génération :

- d'une part l'utilisation des techniques de l'Intelligence Artificielle, et en particulier des Systèmes Experts,

- d'autre part les progrès en matière de représentation et de gestion de données en général, en particulier les bases de données réparties.

Il est apparu évident que les applications nécessitant une interaction avec l'utilisateur comme la CAO avaient besoin de mécanismes d'aide à la décision.

Par ailleurs, l'utilité pratique des bases de données peut être améliorée si elles ne sont plus seulement des réceptacles passifs de données, mais sont dotés de mécanismes "intelligents" de raisonnement, tels que la déduction ou d'outils même rudimentaires d'exploitation de "connaissances" préenregistrées.

Mais l'existence des outils développés auparavant dans le cadre de la première génération de systèmes impose leur intégration dans tout nouveau système [KAT83]. Il est donc nécessaire

d'accéder à une multitude de programmes existant, de pouvoir les lancer puis de récupérer leurs résultats. D'où l'idée de construire des systèmes autorisant la coopération de ces programmes avec les nouveaux logiciels intégrés.

De cette conjonction a résulté une multitude de travaux pour doter les applications CAO d'outils utilisant à la fois des techniques d'Intelligence Artificielle et de bases de données réparties comme KADBASE ("Knowledge Aided Data Base Management System") [HOW84]. Leur objectif est de développer des systèmes distribués (logiquement et physiquement) pour la CAO.

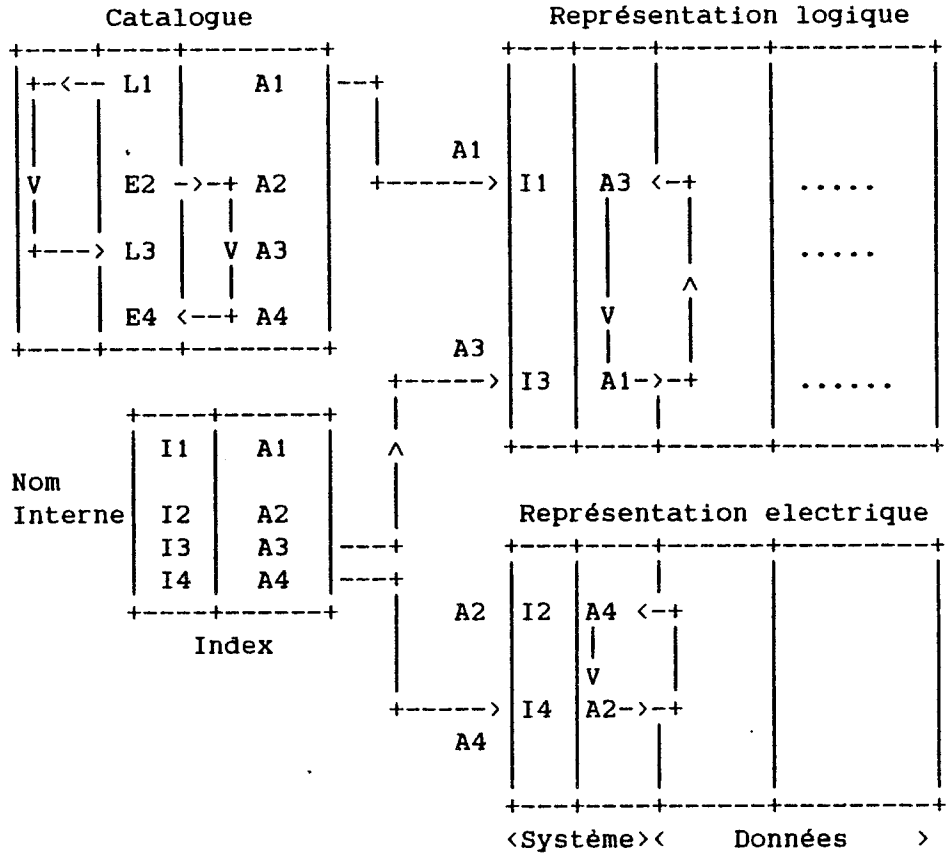
3.2.2 Représentations multiples.

Plusieurs caractéristiques distinguent les applications avancées des traitements plus traditionnels de gestion. L'information est d'une part partagée entre plusieurs utilisateurs qui en ont chacun une vision particulière et des traitements également spécifiques.

Un circuit intégré sera par exemple représenté et manipulé de manière très différente selon que l'on est en cours de conception logique ou électrique et selon la technologie dans laquelle on se place (C-MOS, N-MOS, etc).

Une approche unifiée de modélisation des concepts est donc nécessaire si l'on veut éviter le morcellement auquel donnait lieu les réalisations antérieures. Elle sera étudiée au Chapitre IV. Une uniformisation (donc une simplification) des méthodes de représentation et de gestion interne est également indispensable si l'on veut rentabiliser les efforts de conception automatisée.

On donne à titre d'exemple la représentation utilisée par CVT sur une structure relationnelle. La structure de base est la même que précedemment (paragraphe 3.1), c-à-d que les objets sont représentés sous forme d'ensemble de n-uplets qui peuvent être liés entre eux par des liens explicites de composition ou de référence. Une représentation logique L1 qui utilise un composant logique L3 et dont les représentations électriques sont repectivement E2 et E4 sera modélisée comme ci-dessous :



Les associations entre les diverses représentations peuvent également être modélisées explicitement à l'aide de liens supplémentaires.

La cohérence entre ces différentes représentations doit être maintenue et ne peut plus s'appuyer sur la notion habituelle de transaction.

En effet, il faut d'une part assurer une correspondance entre les éléments de chaque représentation, et ceci nécessite des règles d'abstraction qui n'existent pas dans le cas général [KAT83]. Dans le cas particulier de la technologie MD-MOS, les correspondances qui sont assurées dans CASSIOPEE entre le niveau logique et le niveau d'implantation sont décrites dans le tableau 3.5 [LEC84] :

composant entrée	composant entrée
composant et	nil
composant et/non	porte MD-MOS
équipotentielle en entrée	équipotentielle en entrée sur porte
équipotentielle n sorties	n équipotentielles en sortie
composant sortie	composant sortie

Figure 3.5. Correspondances logique/implanté MD-MOS.

Il faut également pouvoir représenter et manipuler les liens d'équivalence entre ces abstractions (Figure 3.6). Une étude de leur implantation à l'aide de la programmation en logique a été présentée dans [NGU83].

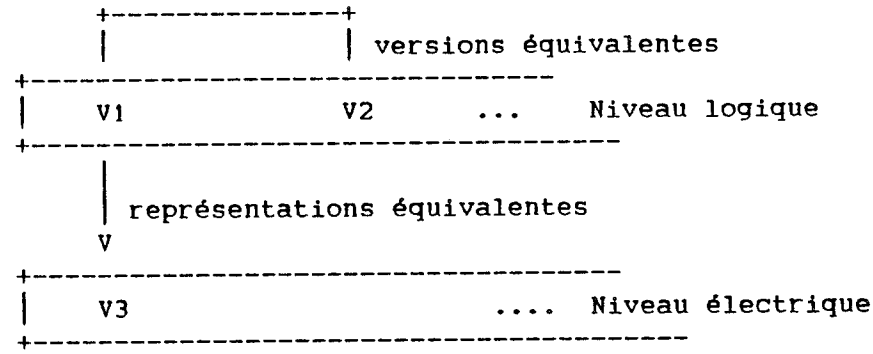


Figure 3.6. Liens d'équivalence entre versions et représentations.

Enfin les mécanismes habituels de maintien de cohérence dans les bases de données utilisent le séquencement ou la détection des interférences entre les opérations pour garantir leur validité ou rejeter les "transactions" en conflit.

Cette approche est basée sur la notion de sérialisation des transactions. Elle est inutilisable dans les applications avancées pour trois raisons :

- la cohérence n'est plus assimilable à un ordonnancement correct des opérations mais à la validité des informations élémentaires en situation,
- les opérations conflictuelles mettent en jeu non plus un seul niveau de représentation, mais plusieurs à la fois,
- à l'intérieur d'une représentation particulière, les conflits sont en général inexistant. Ils peuvent être résolus par des algorithmes de verrouillage classiques (Chapitre V, paragraphe 5.5).

o Validité des informations en situation.

La validité des informations en situation signifie leur cohérence vis-à-vis des liens qui les relient à d'autres informations. Ces liens peuvent être des liens de composition ou d'équivalence, mais aussi des liens fonctionnels calculés [RIE85].

> Dans le cas des objets CAO, une information n'a de sens que par le rôle ou la fonction qu'elle joue par rapport au reste de l'objet à concevoir. Individuellement, ce n'est qu'une donnée élémentaire qui répond à des spécifications particulières et dont le sens peut évoluer selon l'usage qu'on en fait. Ainsi une porte MD-MOS est un composant élémentaire autonome, dont le rôle et la cohérence sont spécifiés par les connexions qu'elle a avec d'autres composants.

Les informations manipulées dans les applications sont donc "auto-cohérentes". Leur validité ne peut pas être contrôlée par des algorithmes d'ordonnancement d'opérations élémentaires, mais seulement par des procédures ad-hoc de vérification de contraintes sémantiques. Ceci les différencient fondamentalement des données traditionnelles manipulées dans les bases de données de gestion.

o Niveaux de conflits multiples.

Le fait que les applications avancées mettent en jeu plusieurs représentations simultanées d'un même objet a également plusieurs conséquences pratiques.

- Ces représentations sont parallèles, c-à-d qu'elles peuvent être utilisées simultanément par des usagers différents.
- Chaque représentation est liée par des contraintes sémantiques particulières avec les autres.
- La cohérence entre les représentations doit être garantie périodiquement.
- Entre deux points de cohérence inter-représentations, ou points de cohérence forte, on ne peut garantir qu'une forme de cohérence partielle ou faible.

Cette dichotomie est une autre particularité des applications avancées. De nouvelles méthodes de vérification et de maintien de cohérence doivent être définies pour la prendre en compte. On en verra un exemple au Chapitre VI.

o Représentations sans conflits.

A l'intérieur d'une même représentation, les conflits d'accès sont faibles. C'est à dire que l'on peut admettre,

comme cela est fait en conception de circuits intégrés, qu'un seul utilisateur peut à un instant donné modifier un objet sous une représentation donnée. Les versions antérieures de cet objet sont par contre accessibles à tous en lecture. De plus, un objet devant être conforme aux spécifications d'un niveau (de représentation) donné, il faut lui associer un état intermédiaire de cohérence partielle [KAT83]. On distingue donc trois états de cohérence par représentation :

- en cours de modification (c-à-d non cohérent),
- en cours de validation (c-à-d partiellement cohérent),
- cohérent (c-à-d accessible).

Dans les applications avancées, ces différentes notions sont étroitement liées à la notion de version d'un objet.

3.2.3 Versions multiples.

La notion de version est essentielle en CAO et en Génie Logiciel. Elle correspond à la construction incrémentale des objets ou des programmes et aux révisions qu'ils subissent après évaluation.

C'est le pendant de la notion d'historique dans les bases de données.

D'une manière générale, il faut pouvoir conserver les versions successives d'un même objet car les modifications sont très longues à réaliser. Leurs implications sont également complexes (toute modification à un niveau logique par exemple a des répercutions sur tous les niveaux inférieurs : électrique, physique, etc). Il faut donc pouvoir revenir en arrière pour un coût minime et de manière instantanée. C'est pourquoi les systèmes intégrés fournissent des mécanismes de gestion de versions et de représentations multiples [EST84, KAT83, JUL86, KIM85].

A titre d'exemple, on donne ici la structure de la base de données du projet CVT (Cad Vlsi for Telecommunications). C'est un projet Européen de conception d'un système intégré en CAO de circuits VLSI. Il est mené conjointement par le CNET, le CSELT : Centro Studi e Laboratori Telecomunicazioni à Turin (Italie) et le FI/DBP : Forschungsinstitut der Deutschen Bundespost à Darmstat (RFA). Le Département Recherche en Conception Assistée du CNET Grenoble est maître d'œuvre de la partie SGBD du projet.

La base de données intègre dans un même ensemble les notions de

représentation, de version, de composant, d'équipotentielle, de port et de patte. Tous ces objets conceptuels sont associés à des objets de représentation graphique (Figure 3.7). Ils sont en effet manipulés par les concepteurs sur des éditeurs graphiques évolués quelque soit le niveau de représentation utilisé.

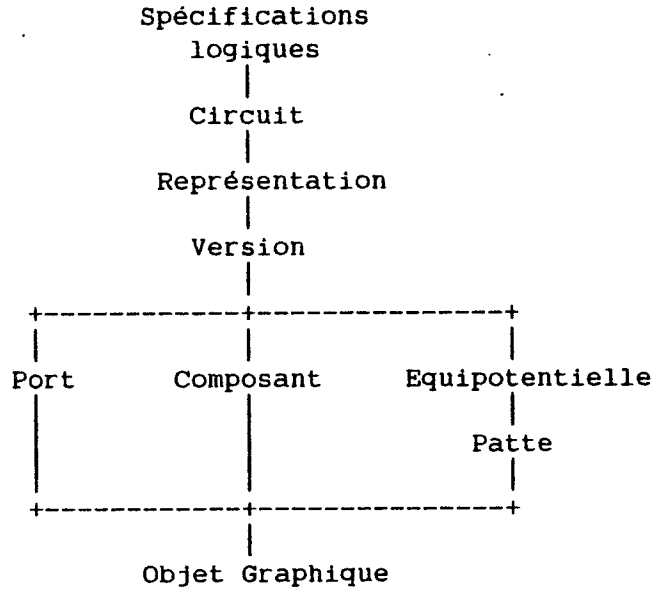


Figure 3.7 Structure de la base de données CVT.

Les informations sont stockées dans la base de données sous forme de relations. Les traits verticaux de la figure 3.7 sont tous des liens du type 1-n. Chaque relation est indexée sur au moins un identificateur interne à l'aide d'un B-arbre.

Toutes représentations confondues, le schéma comporte 104 relations et 522 attributs (index et liens compris). L'ensemble est opérationnel sur VAX (VMS) et sur des postes de travail APOLLO (Unix). A titre indicatif, les premiers essais on montré que le temps de création d'un circuit matriciel 3x3 avec toute sa connectique et ses éléments de représentation graphique s'effectuait en moins de 30 secondes CPU sur un VAX 8600 sous VMS. Ce temps est 5 à 6 fois plus court que lorsque le même programme est exécuté sur un VAX 11/750 (VMS).

3.2.4 Cohérence et complétude.

Une caractéristique des applications avancées est la l'évolution dynamique des structures de l'information. Ceci se reflète en CAO de deux manières :

- d'une part la conception d'un objet est un processus itératif qui est en général la répétition de trois étapes successives : la modélisation d'un objet, son instantiation partielle, puis son évaluation (Figure 3.8). Le résultat de celle-ci peut conduire à modifier les propriétés de l'objet et à en changer partiellement la structure,

- l'instantiation de l'objet est un processus incrémental, dans la mesure où sa conception est "assistée", et non pas "automatisée", ce qui équivaudrait à une synthèse entièrement automatique.

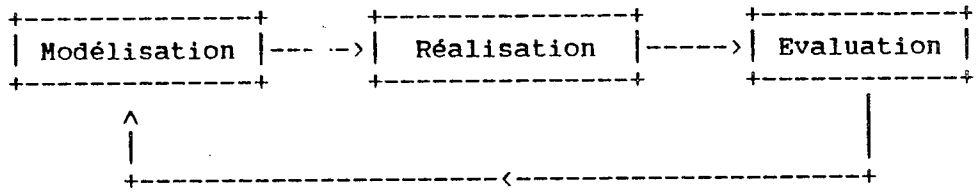


Figure 3.8. Processus de conception.

Lorsqu'au cours d'une étape de construction, l'objet est partiellement instantié, l'évaluation a pour objectif d'en vérifier la cohérence vis-à-vis des spécifications initiales.

De ces observations, il résulte que deux notions différentes sont à la base du processus de conception. Ce sont d'une part la complétude d'un objet en cours de construction, et d'autre part sa cohérence vis-à-vis des contraintes spécifiées lors de sa définition.

De façon formelle, la cohérence d'un objet vis-à-vis d'un ensemble de contraintes est assimilable à la notion de cohérence dans le domaine des bases de données. Si l'ensemble de ses caractéristiques constitue les données contenues dans la base, elles constituent un état de celle-ci. Sa cohérence est donc vérifiable par rapport aux contraintes définies dans le schéma. Si cette cohérence est vérifiée, l'état de la base de de données, donc l'objet qu'il représente, est un modèle au sens de la logique, de ces contraintes.

La théorie suppose néanmoins que pour que la base de données soit un modèle de l'ensemble des contraintes, il n'y ait pas de valeurs indéfinies présentes dans cet état.

Ceci représente un inconvénient majeur en CAO, puisqu'à priori, tout objet est "en devenir", c-à-d incomplet tant que sa conception n'est pas terminée. Par définition, on dira qu'un objet est "incomplet" si ses occurrences comportent des valeurs indéfinies. A contrario, un objet qui n'est pas incomplet sera dit "complet".

En CAO, cette notion est tout à fait différente de celle de cohérence. Si l'on peut estimer en première approximation qu'un objet incomplet est incohérent, puisque certaines contraintes ne pourront pas être évaluées à cause des valeurs indéfinies, ceci s'avère tout à fait insuffisant dans la pratique.

En effet, faire cette assimilation : incomplet = incohérent revient à ne jamais pouvoir utiliser les notions de théorie et de modèle d'une théorie. Par ailleurs, l'évaluation partielle de contraintes est une opération couramment effectuée dans les systèmes classiques de CAO par des programmes ad-hoc, tels que les simulateurs logiques, électriques et autres. Dans ce cas, la théorie n'est donc d'aucun secours pour la conception de systèmes avancés de gestion de données. Elle présente en fait l'inconvénient majeur de bloquer le processus de conception puisqu'aucun objet ne peut dans ce cas être validé à cause de son incomplétude.

Une autre approche a été proposée par Dominique Rieu [RIE85]. On y assimile tout objet incomplet à un objet "potentiellement cohérent". Ceci présente l'avantage de ne pas bloquer la conception d'un point de vue pratique. Dans ce cas toutefois, il est clair que la démarche est à l'opposé de la théorie, et en particulier de la logique appliquée aux bases de données.

Dans cette approche, à chaque objet sont associés deux états qui caractérisent l'un sa complétude, l'autre sa cohérence. Le tableau des combinaisons qui lie ces états est résumé ci-dessous (Figure 3.9). Les symboles "V" et "F" dénotent les valeurs "vrai" et "faux" respectivement.

Complétude	Cohérence	
F	V	(1)
F	F	(2)
V	F	(3)
V	V	(4)

Figure 3.9. Combinaisons Complétude/Cohérence.

Par défaut, tout objet incomplet est "potentiellement cohérent". Ceci est illustré dans la ligne (1) du tableau ci-dessus par le

couple d'états (F, V).

On va affiner ici cette notion de "cohérence potentielle" ou "cohérence par défaut" d'un objet incomplet pour la rapprocher de notions plus rigoureuses, et la remplacer par celle de cohérence "partielle" qui sera formalisée au chapitre VI. Elle est à la base d'une méthode originale de vérification de contraintes, dite "canonique". Elle est mise en œuvre à l'aide des notions de prototype d'objet et d'échantillon d'une base de données, qui y sont également définies.

> La démarche suivie est de déterminer les contraintes vérifiées par un objet en cours de construction. On caractérise chaque objet incomplet par la classe d'équivalence à laquelle il appartient. Cette classe d'équivalence est définie par le sous-ensemble de contraintes qu'il vérifie au cours d'une étape de sa conception. Durant cette construction incrémentale, l'objet évolue d'une classe à l'autre. Toute modification est alors testée, ou encore certifiée sur un représentant de la classe appelé "prototype", avant d'être appliquée à l'objet lui-même, c-à-d validée.

Cette approche permet d'éviter le carcan de la logique appliquée aux bases de données, en n'obligeant pas à considérer un état de la base de données comme un modèle d'une théorie pour contrôler la cohérence des objets. Ceci évite également d'avoir recours à la logique modale. Elle a l'avantage de rester proche de la démarche habituelle des concepteurs en CAO, puisqu'elle prend en compte l'évolution graduelle des objets et leur incomplétude. Elle est basée en plus sur des fondements rigoureux. Son étude formelle est le sujet du chapitre VI.

3.2.5 Répartition.

La figure 3.10 décrit l'approche implantée dans le cadre du projet CVT pour la répartition des informations et des moyens de calcul. Les postes de travail du type APOLLO possèdent une base de travail locale qui est en lectures multiples et en écriture pour un seul concepteur. Ils sont dédiés aux traitements graphiques interactifs et à la gestion des librairies individuelles. Le VAX gère les librairies et les gros outils logiciels comme les simulateurs. Les bases y sont en lectures partagées.

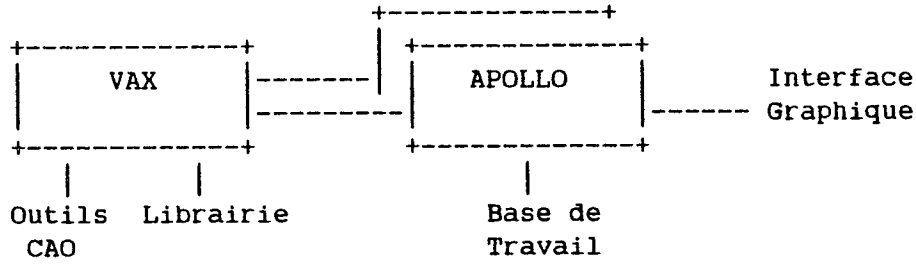


Figure 3.10 Première répartition dans CVT.

Ce type de répartition physique est en accord avec la hiérarchie de bases de données proposée par Dominique Rieu [RIE85] :

- base de versions,
- base de travail,
- base de projet,
- base de connaissances.

Au niveau le plus bas, se situe la gestion des objets proprement dits. La figure 3.11 décrit l'architecture du serveur du projet CVT [JUL86]. Le module SI (Storage Interface) contient les extensions de la mémoire relationnelle, en particulier tout ce qui concerne la gestion des liens entre données relationnelles et des identificateurs internes. Le module DBI (Data Base Interface) contient les interfaces spécifiques destinées aux outils de CAO de circuits VLSI. Le module SCHEDEF (Schema Definition) permet la création et modification dynamique de schémas de la base de données. C'est lui en particulier qui permet de définir les différentes représentations d'un même circuit.

Le module MIMER est la mémoire relationnelle du SGBD MICROBE. Il a été étendu par le CNET Grenoble pour pouvoir répartir les informations sur plusieurs bases parallèles. Les données sont stockées sous formes de relations normalisées. Elles sont accessibles via un nom "externe" ou un identificateur interne uniques. Elles peuvent être regroupées à la demande afin d'accélérer les recherches et d'implanter un mécanisme de gestion de versions. Enfin la notion de liens a été introduite afin d'expliciter les relations hiérarchiques entre les informations.

Les modules MIOPE et MIQUEL du SGBD MICROBE sont utilisés localement pour des interrogations ponctuelles et la mise au point du système.

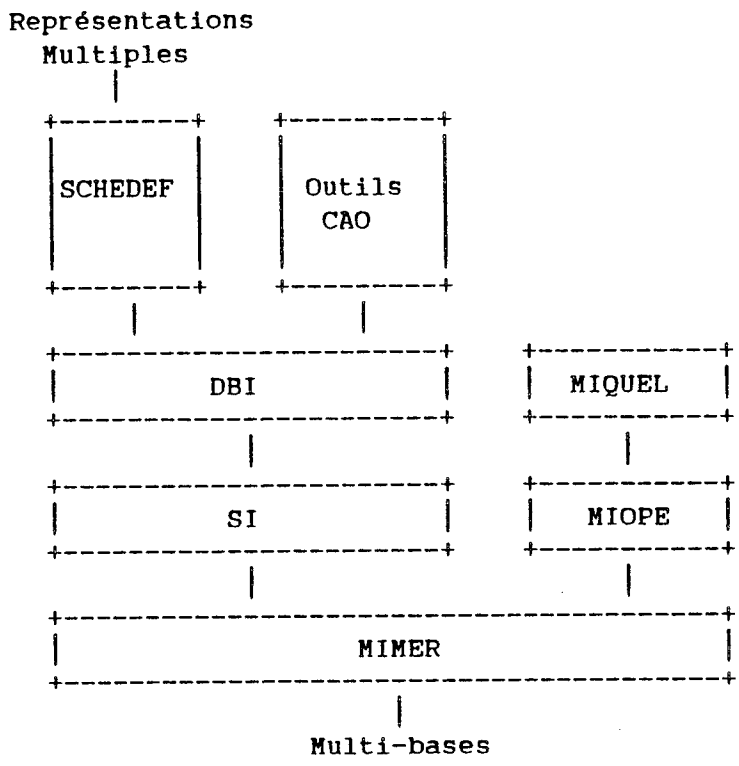


Figure 3.11 Architecture générale du serveur CVT.

3.3 Conclusion.

Deux problèmes essentiels semble devoir faire l'objet de solutions nouvelles. D'une part la modélisation des applications. D'autre part la répartition des informations.

La modélisation dans les applications avancées ne peut plus se cantonner à la description statique des structures de l'information. Déjà, la notion de représentations multiples impose des outils d'intégration évolués. Il faut pouvoir également spécifier le comportement dynamique des objets, et de ce fait les traitements qu'ils subissent.

Si les techniques de l'Intelligence Artificielle répondent bien à ces exigences, leur intégration aux bases de données reste à faire.

La modélisation qui en découle doit permettre à terme de représenter également l'environnement de l'application, c-à-d les interactions entre les différents acteurs (humains ou programmés) qui y interviennent.

La répartition de l'information est quant à elle une nécessité imposée au moins autant par les méthodes de travail modernes que

par le besoin de réutiliser des outils et de partager des données qui existent déjà.

Après avoir décrit ici les spécificités des applications avancées et donné quelques éléments de réalisation actuels, les deux chapitres qui suivent explorent ces deux aspects complémentaires.

CHAPITRE IV

MODELISATION

CHAPITRE IV

MODELISATION

On définit dans ce chapitre les caractéristiques des systèmes de gestion de bases de données avancées ou SGBDA. On ne fait pas usage dans ce qui suit du terme "bases de connaissances", car il recouvre à l'heure actuelle des concepts qui ne font pas l'unanimité. En particulier, il semble que ce vocable soit utilisé par ceux qui estiment que l'extension des systèmes actuels de représentation des connaissances, basés par exemple sur les "frames", à l'aide de techniques inspirées des SGBD pour le simple stockage de faits ou de règles, suffise aux besoins exprimés ici et là [BEN85]. Toutefois, on n'a pas encore vu de système intégré de base de connaissances qui offre des fontionnalités qui soient :

- d'une part adaptées à cette forme d'information, ne serait-ce que pour la classification des connaissances et un stockage adapté à celle-ci,
- et qui d'autre part le fasse de manière aussi poussée que les SGBD classiques, avec par exemple une indexation automatique des règles.

Des tentatives sont faites dans ce domaine sans que cela ait encore débouché sur des outils concrets [SCI84, WIE84].

Notre opinion est que la prise en compte de problèmes tels que le partage contrôlé d'informations ayant des représentations multiples et des versions multiples ne peut pas être résolu par cette approche. En particulier parce que les modèles et langages développés jusqu'à présent n'offrent pas de solution à cet égard.

Il est donc nécessaire de développer de nouvelles méthodes qui permettent d'implanter de manière saine des algorithmes de représentation, de définition, de manipulation et de partage de données "avancées", y compris celles que l'on a l'habitude d'appeler des connaissances, en partant des fonctionnalités des SGBD. On pourra alors définir des méthodes de traitement de l'information construites sur des bases clairement définies et qui mettront en œuvre de nouvelles techniques.

A notre avis, seule l'approche unifiée basée sur la programmation en logique permet à l'heure actuelle de développer et de mettre en œuvre des spécifications "prouvables", c-à-d basées sur des fondements théoriques clairement définis et qui font ap-

pel à des techniques de démonstration mathématique comme la déduction.

Toute autre approche manque de fondements théoriques et se ramène à des techniques plus ou moins évoluées de programmation ainsi que de présentation plus ou moins agréable de concepts à l'utilisateur. On peut citer par exemple les tentatives de programmation orientées objet bâties sur les mécanismes de "frames" utilisés pour la représentation des connaissances en Intelligence Artificielle. A contrario, un exemple de méthode rigoureuse de gestion de contraintes est fourni au Chapitre VI avec la vérification "canonique" de contraintes sémantiques et les notions de prototype d'objet et d'échantillon d' une base de données.

Après avoir présenté quelques-unes des solutions proposées actuellement et montré leurs insuffisances, on définit dans un premier temps une méthode de contrôle de la sémantique associée à des données généralisées.

On étend ensuite les concepts utilisés, pour faire appel aux notions d'objet, de propriété, de contrainte et de méthode. On détaille ensuite les contrôles sémantiques sur les objets. Ils font appel à la notion de liens, que l'on caractérisera.

On propose enfin une architecture ouverte pour la gestion de bases de données avancées, qui permette l'intégration d'applications existantes, ce qui est une des exigences essentielles des utilisateurs pour l' acceptation de nouveaux concepts et produits.

4.1 Solutions existantes.

Il y a un effort important dans le monde visant à l'intégration réciproque des SGBD et de l'Intelligence Artificielle.

Alors qu'on essaye ici d'implanter une architecture qui structure les informations en plusieurs bases de connaissances et bases de données qui interagissent, on voit également apparaître des propositions :

- d'extension de modèles de représentation des connaissances [BEN85],
- d'interface évoluée entre systèmes déductifs et SGBD [DEM85, ZAN85],
- d'extension des fonctionnalités des SGBD classiques [STO85].

Dans le premier cas, les choses sont abordées du côté de l'Intelligence Artificielle. Plus précisément, on y étend les

méthodes d'accès aux informations en incorporant à un système basé sur les "frames" et écrit en Lisp, les fonctions d'un SGBD relationnel [BEN85]. Cette approche, si elle intéresse les constructeurs de Systèmes Experts, n'étend pas les fonctionnalités du SGBD sous-jacent. Elle se contente d'en faire un bon usage.

La deuxième approche est plus constructive. Partant d'un SGBD relationnel dont les opérateurs sont étendus de manière ad-hoc, on donne à l'usager la possibilité de manipuler des objets complexes et la notion d'héritage entre ces objets [ZAN85]. La stucture arbitraire des objets implique la définition d'une interface qui est dans une logique d'ordre supérieur à 1. Elle a néanmoins l'inconvénient de recourir à des opérateurs non-triviaux pour être interprétable dans une algèbre relationnelle étendue [DEM85, ZAN85]. Des études similaires, quoique moins ambitieuses car limitées à la logique du premier ordre, sont décrites dans [NGU84b, RIE85].

La facilité avec laquelle on peut ici utiliser les mécanismes de la programmation en logique est cependant assez artificielle. En bonne logique, il n'est en effet pas nécessaire d'introduire de nouveaux opérateurs pour traiter des structures complexes puisque la conjonction de prédicats du premier ordre suffit pour cela. L'intérêt essentiel est alors de profiter de méthodes connues d'optimisation de requêtes relationnelles. De même, une étude prenant en compte la représentation d'objets complexes comportant des valeurs incomplètes est présentée dans [DEM85].

La troisième approche est la plus conservatrice, du point de vue des Bases de Données. Alors que dans les études précédentes on cherche à tirer parti directement de modèles évolués de représentation d'objets complexes en logique ou en "frames", on cherche ici à introduire de nouveaux opérateurs de déduction sans toucher aux concepts du modèle relationnel normalisé. Le principe est de définir un opérateur de fermeture transitive et un opérateur de déduction [STO85]. Les opérations de déduction y sont activées par des actions spontanées ... "différées" ("lazy triggers"...). Elles sont déclenchées par des procédures de réveil lors des opérations de mise à jour. Elles nécessitent pour cela la mémorisation de verrous particuliers sur les données qui sont touchées par les opérations de déduction.

Ces propositions souffrent d'une certain nombre de lacunes.

D'une part, la dichotomie entre schéma et contraintes est toujours présente, alors que les structures définies dans un schéma ne sont rien d'autre que des contraintes structurelles définies sur les données. Les dépendances entre les informations sont également des contraintes d'un type particulier. On va unifier cette approche en modélisant l'ensemble des informations à l'aide d'un seul formalisme.

D'autre part, aucune de ces propositions ne prend en considération les problèmes de répartition des informations et les problèmes de contrôle de cohérence. Les technologies respectives qui peuvent être utiles dans chaque cas ne sont pas non plus étudiées. C'est ce que l'on va tenter de faire dans les deux Chapitres suivant.

4.2 Connaissances et objets complexes.

On ne fait pas dans ce qui suit de distinction entre faits et connaissances, tout au moins au niveau externe. Ceci est la conséquence de plusieurs observations.

D'une part on utilise un seul et même outil pour spécifier et modéliser l'information : la programmation en logique des prédicats [NGU83, ZAN85]. On sait que l'on peut dans ce cadre utiliser une approche entièrement déclarative des informations.

D'autre part, comme il a été dit par ailleurs [STO85], la complexité d'utilisation d'un modèle de données croit plus qu'exponentiellement avec le nombre de concepts qu'il permet de manipuler. On ne veut donc pas introduire de nouveau formalisme qui viendrait se superposer à un modèle de représentation de connaissances particulier.

Ce souci de simplification et d'uniformisation explique sans doute le succès du modèle de données relationnel, dans lesquel l'usager n'opère que sur un seul concept, la relation. Ceci tendrait également à expliquer l'impact actuel des modèles orientés objet [SMA80].

L'objectif est ici de spécifier et de contrôler la sémantique associée à des applications avancées. On entend par ce terme la caractérisation :

- des objets qu'elles manipulent,
- de leurs propriétés statiques,
- de leur comportement dynamique.

On décompose cette démarche en trois volets :

- spécification de la sémantique attachée au schéma conceptuel d'une application, et donc aux concepts du modèle de données,
- spécification des opérations de manipulation et de vérification de contraintes statiques,
- vérification de contraintes dynamiques.

Dans un premier temps, on décrit la modélisation de données généralisées en logique du premier ordre. On s'appuie pour cela

sur le modèle de données développé dans le cadre du projet TIGRE.

On généralise ensuite cette démarche en adoptant certains principes de la programmation orientée objet. On montre qu'une implantation par la programmation en logique est également possible.

4.3 Modélisation de données généralisées.

L'idée d'incorporer la notion d'objets complexes dans un modèle de données généralisées n'est pas neuve. Les insuffisances des modèles "classiques" comme le modèle relationnel sont connus depuis longtemps. Elles concernent aussi bien la pauvreté des structures de l'information que la représentation de la sémantique qui leur est attachée.

Modéliser et manipuler des données généralisées implique la conception de modèles plus riches qui permettent par exemple de décrire des documents, des graphiques ou encore des images. Les manipuler suppose qu'en plus de fonctions ad-hoc de traitement, on leur ait associé une sémantique précise.

Il se trouve que la description d'objets complexes est formalisable en logique du premier ordre [DEM85, ZAN85]. De même, la spécification de leur sémantique est facilement exprimée sous forme de prédicats du premier ordre [CET84].

On s'attache dans ce paragraphe à montrer comment ceci est appliqué au modèle de données généralisées du projet TIGRE [TIG83].

On part ici de l'hypothèse que TIGRE existe et qu'il permet de modéliser des applications avancées grâce à un modèle de données fortement basé sur la notion de type.

Il intègre par exemple les notions d'entité, d'association, de classe, de sous-classe, d'aggrégation et de généralisation [TIG83].

Les opérations licites sur ces concepts sont restreintes par la sémantique du modèle. En particulier, les attributs d'une entité ne peuvent pas être des entités. Une association peut lier deux entités entre elles. Un aggrégat est une structure non ordonnée d'entités. Enfin deux entités liées par une association peuvent être aggrégées en un nouveau type.

Dans ce contexte, le serveur TIGRE offre les outils de stockage et de manipulation d'objets volumineux et complexes. Ce sont par exemple des documents ou des circuits électroniques. Parallèlement, la programmation en logique permet d'implanter des outils de définition et de contrôle de la sémantique des appli-

cations.

Par rapport à d'autres approches, on ne vise donc pas à étendre les fonctionnalités d'un système déductif. Par exemple, on ne propose pas de méthode qui optimise les accès à des banques de clauses ou qui permette l'évaluation de clauses récursives [ULL85].

On veut au contraire enrichir les fonctionnalités d'un SGBD de 3e génération avec la technologie des systèmes déductifs. Dans la mesure où les informations et les traitements sont alors répartis, on montrera en particulier que cette approche est compatible avec les techniques développées dans le domaine des bases de données réparties et des systèmes distribués (Chapitre V).

Cette approche présente de plus l'avantage d'être conforme à un environnement dans lequel les connaissances ou informations de base sont stockées sur des serveurs, et partagées par des applications ou des utilisateurs agissant sur des postes de travail individuels [KAT83b].

La formalisation des concepts du modèle de données généralisées du projet TIGRE est donnée ci-dessous. Par analogie avec la notion de type abstrait, on utilise la notion de "constructeur" afin d'encapsuler la définition des informations et les opérations de manipulation.

La syntaxe du langage Prolog utilisée ici est celle de la version C-Prolog d'Edimbourg qui est disponible sur le VAX 11/780 du Laboratoire de Génie Informatique de l'IMAG. C'est celle qui a été utilisée pour la réalisation de ce qui suit et pour le contrôle de cohérence par certification (Chapitre VI). On rappelle pour mémoire qu'une clause :

```
p(X,Y) :- q(X), r(X,Y), s(Y,_)
```

est interprétée comme une implication :

"pour toutes valeurs des variables X et Y qui rendent les prédicats q, r et s vrais, alors p est évalué à vrai".

Les noms de prédicats et les constantes sont notés en minuscules. Les variables en majuscules. Une variable anonyme, notée "_", est sans objet dans l'évaluation du prédicat dans lequel elle apparaît. La virgule séparant deux prédicats est interprétée comme une conjonction logique. La disjonction est notée ";" ou par plusieurs règles ayant même partie gauche :

```
p(X,Y) :- r(X,Y), q(Y)
p(X,Y) :- t(X,Y), r(X,_)
          ou
p(X,Y) :- (t(X,Y), r(X,_)) ; (r(X,Y), q(Y)).
```

Enfin, une liste d'éléments est notée [X|Y], où X est le terme tête de la liste et Y la sous-liste queue de la liste.

Schématiquement, le premier volet : "Spécification de la sémantique attachée au schéma conceptuel d'une application" (paragraphe 4.1) est réalisé en quatre étapes :

1 - génération automatique d'un noyau d'opérateurs de base sur les types de données généralisées,
2 - traduction automatique des types de données de base de l'application sous formes de clauses de Horn,
3 - traduction des entités et associations de l'application,
4 - génération automatique de procédures de contrôle de contraintes sémantiques sur les données.

Ces procédures sont des ensembles de clauses de Horn qui seront activées automatiquement ou à la demande. Elles correspondent aux opérations élémentaires de création, suppression et mise à jour sur chaque type d'information manipulé par l' application.

Dans un premier temps, on GENERE donc un ensemble de clauses qui définissent les contraintes de type associées aux informations (étapes 1, 2 et 3).

Ensuite on définit comment UTILISER ces contraintes en conjonction avec les programmes d'application (étape 4).

Si l'on désire uniquement contrôler les contraintes de type sur un schéma de données, on peut s'arrêter à l'étape 3. On peut ainsi contrôler la correction de ce schéma destiné à une application particulière. On dispose de cette façon d'un outil de vérification et d'aide à la conception de schémas qui peut être invoqué à tout moment et qui délivre une valeur booléenne reflétant la validité d'un schéma particulier. On peut de ce fait contrôler l'évolution dynamique des structures de données.

La première étape consiste à définir le noyau d'opérateurs de base pour la vérification des types élémentaires du modèle de données généralisées. Ce noyau constitue un méta-modèle en ce sens qu'il spécifie les éléments fondamentaux du modèle TIGRE. Il comprend :

- les types de base (entier, chaîne de caractères, réels),
- les types dérivés et construits (tableaux, enregistrements, intervalles),
- les entités et associations,
- la spécialisation d'entité et d'association,
- l'union d'entités,

- l'aggrégation d'entités et d'associations.

o Les concepts de base sont définis par les clauses suivantes :

```
type (X) :-          type_de_base(X) ; type_dérivé(X) ;
                     type_construit(X) ; classe(X).
type_de_base (X):-   entier(X) ; chaîne(X) ; booléen(X) ;
                     texte(X) ; intervalle(X).
type_dérivé (X) :-   type_de_base(X).
type_construit(X):- tableau(X) ; enregistrement(X) ; document(X).
classe(X) :-         entité(X) ; association(X) ;
                     généralisation(X) ; aggrégat(X).
généralisation(X):- spécialisation(X) ; union(X).
aggrégat([]).
aggrégat(X) :-       X is [Y|L], not (aggrégat(Y)), type(Y),
                     aggrégat(L).
```

o La spécialisation S d'une entité E selon un prédicat de définition évaluable P est définie par :

```
spécialisation (S,E,P) :- entité(S), entité(E), call(P).
```

où S et E doivent avoir été déclarés comme entités, et call(P) un prédicat prédéfini qui évalue la variable P considérée comme un prédicat.

o L'union U de deux types entité E1 et E2, restreints sur les attributs P1 et P2 est :

```
union (U,E1,E2,P1,P2) :- entité(E1), entité(E2),
                         type(P1), type(P2),
                         compatible(P1,P2).
```

où compatible est un prédicat évaluable.

La définition des types construits et dérivés fait appel aux types existant. De même que le mécanisme de types abstraits permet de définir de nouveaux types à partir de ceux qui existent, on génère des clauses qui font appel à celles qui sont déjà générées.

Ce mécanisme étant récursif, on peut définir des structures arborescentes complexes.

o Un document qui est constitué d'une introduction, d'une

liste de chapitre et d'une conclusion, où chaque chapitre comporte un titre, et un contenu du type prédéfini texte est

```
document (X) :- type(X), structure(X).
structure([Introduction,C,Conclusion]):-
                texte(Introduction), Chapitres(C),
                texte(Conclusion).
Chapitres([]).
Chapitres(C):-  C is [X|Y], Chapitre(X), Chapitres(Y).
Chapitre(X) :-  X is [Titre|Contenu], chaîne(Titre),
                texte(Contenu).
```

Cette première étape peut être définie indépendamment de toute application puisqu'elle consiste en la spécification des éléments fondamentaux du modèle généralisé.

Notons qu'à partir d'un schéma défini conformément à la syntaxe du modèle TIGRE, il est possible de générer automatiquement l'ensemble des clauses de Horn qui le définit en logique du premier ordre. C'est l'objet d'un système de réécriture lui-même écrit en Prolog.

La deuxième étape, c-à-d la traduction des types de données spécifiés par un utilisateur fait appel à ce noyau de base.

o Une entité "employé" définie dans TIGRE par :

```
Define entity employé
   Key NUM       : integer end
       NOM       : string (20)
       SAL       : montant
       CAT       : (ingénieur, secrétaire)
       ADR       : type_adresse
       CONTR     : type_contrat.
```

est modélisée sous la forme :

```
entité(employé).
constructeur (employé, NUM, NOM, SAL, CAT, ADR, CONTR) :-
    type          (employé),
    key           (employé, NUM),
    integer       (NUM),
    string        (NOM),
    montant       (SAL),
    catégorie     (CAT),
    type_adresse  (ADR),
    type_contrat  (CONTR).
```

où les types integer, string, ..., type_adresse et type_contrat sont prédéfinis. Par exemple :

```
type_adresse (X) :- X is [No|L], integer(No), lieu(L).
lieu(L)          :- L is [Voie|Ville], string(Voie),
                    string(Ville).
```

et :

```
     montant(X)  :- integer(X), X >= smic, X <= plafond.
```

o Ainsi, la qualification d'une secrétaire définie comme une valeur scalaire dans (sténo, dactylo) est :

```
scalaire(qualification).
qualification (X) :- type(X), (X='sténo' ; X='dactylo').
```

o Une spécialisation d'entité est donnée par :

```
Define entity secrétaire where categorie = 'sténo'
       bilingue : booléen
```

et :

```
spécialisation (secrétaire, employé, 'sténo') :-
    constructeur (secrétaire (NUM, NOM, ..., BILINGUE)),
    arg(2,spécialisation(secrétaire,_,_),NUM),
    constructeur(employé(NUM, NOM, ...),
    booléen (BILINGUE).
```

où arg(k,p(x1,x2,...,xn),E) affecte à la variable E le k-ième argument de p, c-à-d xk, si k<=n.

o De même, une association est modélisé par un ensemble de clauses qui spécifient les types associés, leurs rôles respectifs et leur cardinalité.

```
Define lieu_de_travail between
       employé : poste (1,*)
       bureau  : site (1,1)
       DEPUIS  : date
```

et

```
association (lieu_de_travail,employé,site).
rôle        (lieu_de_travail,employé,poste,1,*).
rôle        (lieu_de_travail,employé,bureau,1,1).
constructeur(lieu_de_travail(EMP,SITE,DATE)) :-
             association(lieu_de_travail),
             employé(EMP),
             bureau(SITE),
```

date(DATE).

Les définitions de généralisations et d'aggrégats s'effectuent de manière similaire.

Il importe de garder présent à l'esprit que ces clauses sont générées automatiquement à partir d'un schéma fourni pour l'application. De même, leur invocation peut être faite automatiquement lors des modifications de ce schéma afin d'en contrôler l'évolution.

La notion de constructeur est également utilisée lors des opérations d'insertion, suppression et modification pour contrôler leur validité avant d'être rendues effectives. Ainsi une création d'entité peut être contrôlée par la séquence :

créer (X) :- L =.. X, constructeur(L), asserta(L).

où L =.. [p|q] affecte à la variable L la valeur p(q).

Cette décomposition des opérations en deux temps sera utilisée pour le contrôle de cohérence dans la méthode de certification développée au Chapitre VI.

On sait par ailleurs que le langage Prolog permet d'implanter les opérateurs de l'algèbre relationnelle s'il est doté de l'opérateur ensembliste "set_of". Si le langage développé dans le cadre d'un modèle généralisé est complet au sens de Codd, c-à-d qu'il a un pouvoir d'expression équivalent à l'algèbre relationnelle [DEL82], on garantit par cette démarche que toute opération réalisable en relationnel l'est également en Prolog.

4.4 Modélisation d'objets.

Pour ne pas rester tributaire d'un modèle de données particulier, on a élargi la démarche précédente en s'inspirant des concepts développés dans le cadre de la programmation orientée objet.

On s'appuie sur la logique des prédicats du premier ordre pour spécifier les concepts de cette modélisation.

On a adopté une modélisation à base d'objets pour réaliser l'interface utilisateur avec le système de validation (Chapitre VI). Il n'y a plus que des objets, dont la définition constitue le schéma de la base de données. Ces objets ont des propriétés, et sont dotés également de contraintes et d'opérateurs attachés, ou méthodes, selon la terminologie habituelle.

Les propriétés sont des constituants ou attributs au sens du modèle relationnel, c-à-d des caractéristiques simples ou complexes. Une caractéristique simple est par exemple le numéro ou la marque d'une voiture. Une caractéristique complexe est un objet faisant l'objet d'une définition propre.

Les contraintes expriment le comportement de l'objet vis à vis de l' extérieur, c-à-d des autres objets et de l'utilisateur.

Les contraintes existentielles décrivent les dépendances d'un objet vis-à-vis des autres objets. Elles sont connues de l'utilisateur qui définit l'objet. Ce sont par exemple les dépendances fonctionnelles, multivaluées et d'inclusion du modèle relationnel.

Les contraintes opérationnelles décrivent le comportement dynamique de l'objet soumis aux manipulations de l'usager. Elles servent essentiellement à exprimer les conditions requises sur un objet avant de pouvoir lui appliquer une opération, et les effets de bord consécutifs à l'application de cette opération. Ces dernières sont utilisées pour la propagation des mises à jour. Elles sont semblables aux contraintes de mise à jour ("update dependencies") des bases de données relationnelles, dont on sait qu'elles peuvent être déclarées et évaluées en logique des prédicats du premier ordre.

L'exemple suivant donne la définition d'un objet "cellule" en terme d'interface, ou ports d'entrée-sortie, et de composants (Figure 4.1). Ces derniers sont des instances de cellules.

```
CELLULE
        propriétés :
                        PORT      : type, numéro
                        NET       : numéro, set_of PIN
                        COMPOSANT : set_of CELLULE
                        PIN       : set_of PORT
                        UTILISE   : set_of CELLULE
        contraintes :
                        COMPOSANT : instance_of CELLULE
                        VSS       : instance_of PIN_OUT
                        VDD       : instance_of PIN_IN
        opérations :
                        delete CIRCUIT => delete UTILISE
        méthodes   :
                        CONNECT Y Z: PORT_IN <- PORT_OUT
                                                of CELLULE Y
                                     PORT_OUT<- SIGNAL Z.
```

Figure 4.1. Définition d'un objet CELLULE.

Les mots-clés "propriétés", "contraintes" et "opérations" ouvrent les définitions des propriétés de l'objet, ainsi que de ses con-

traintes existentielles et opératoires. La caractérisation d'une propriété par un "set_of" spécifie que sa valeur est un ensemble d'instances non ordonnées du type référencé. La clause "instance_of" spécifie le fait que la propriété est une réalisation du type référencé. La contrainte opérationnelle spécifiée ici stipule que la suppression d'une instance de l'objet CELLULE entraine comme effet de bord la suppression automatique des instances de CELLULE qu'elle utilise.

Par définition, un objet qui ne fait référence à aucun autre objet que lui-même dans sa définition sera dit indépendant ou explicite.
Par contre, un objet qui comporte une référence à une ou plusieurs propriétés entrant dans la définition d'un autre objet sera dit implicite.
Enfin, un objet qui fait simultanément référence dans sa définition à des propriétés d'objets implicites et explicites sera dit hybride (Figure 4.2).

On constate sur cette définition que :

- un objet peut être défini de façon récursive,
- une propriété peut faire implicitement référence à une autre propriété, comme VDD (la tension d'alimentation) et VSS (la masse) qui sont définies dans l'exemple précédent comme réalisations de la propriété PIN, elle-même définie comme ensemble (set_of) de PORT (Figure 4.1). On parlera dans ce cas de propriété implicite.

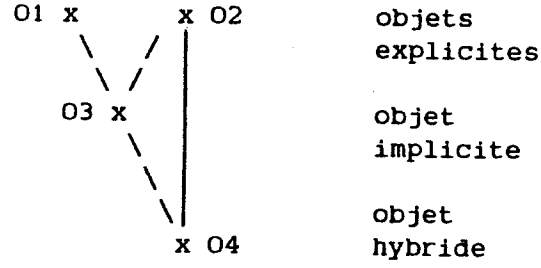


Figure 4.2 Relations entre types d'objets.

Les objets O1 et O2 sont explicites. L'objet implicite O3 est défini à l'aide des objets explicites O1 et O2. L'objet hybride O4 est défini à l'aide de l'objet explicite O2 et de l'objet implicite O3.

Les exemples qui suivent illustrent des définitions d'objets explicites, implicites et hybrides. Pour plus de clarté, les méthodes ne sont pas mentionnées.

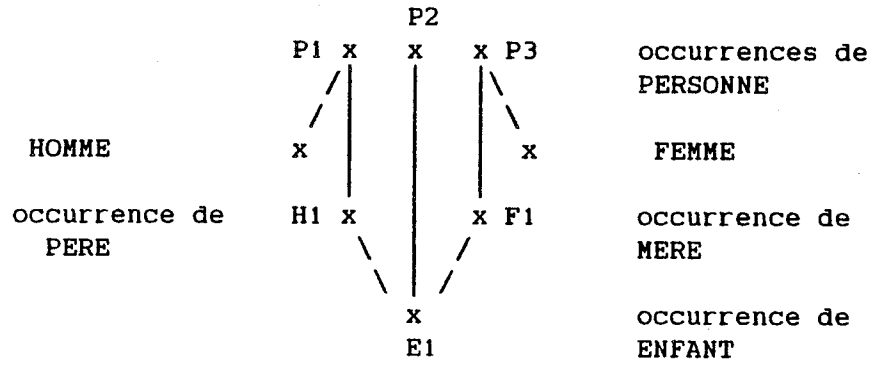
```
PERSONNE
/* objet explicite */
propriétés : identificateur (entier)
             nom            (chaîne)

             sexe           (M, F)
             age            (entier).

HOMME
/* objet implicite */
propriétés :
contraintes :
                 instance_de PERSONNE
                 sexe = M.

FEMME
/* objet implicite */
propriétés :
contraintes :
                 instance_de PERSONNE
                 sexe = F.

ENFANT
/* objet hybride */
propriétés :
                 père : instance_de HOMME
                 mère : instance_de FEMME.
contraintes :
                 instance_de PERSONNE
```

Ces définitions sont mémorisées à l'aide des formules suivantes :

```
(x,i,n,s,a) [ personne(x)  <- ident(x,i)  ∧ nom(x,n)  ∧ sexe(x,s)
                              ∧ age(x,a)  ∧ entier(i) ∧ chaîne(n)
                              ∧ (s=F ∨ s=M) ∧ entier(a) ]
      (x) [ homme(x)       <- personne(x) ∧ sexe(x,M) ]
      (x) [ femme(x)       <- personne(x) ∧ sexe(x,F) ]
    (x,p,m) [ enfant(x)    <- personne(x) ∧ homme(p) ∧ pere(x,p)
                                          ∧ femme(m) ∧ mere(x,m) ]
```

où "<-" est l'implication logique et (x) la quantification universelle d'une variable x.

Les occurrences de ces objets sont liées de la façon suivante :



Les formules précédentes qui définissent les objets personne, homme, femme et enfant constituent donc un schéma.

Dans le cas général, ces formules peuvent faire intervenir la notion de récursivité. La définition récursive d'objets pose deux types de problèmes. D'une part leur traitement par des moyens informatiques habituels nécessite un contrôle explicite de leur mise à jour afin de ne pas rentrer dans des boucles infinies d'évaluations. Ceci peut être résolu de plusieurs façons :

- soit en introduisant une distinction explicite entre l'instance de l'objet manipulé et celles de ses ancêtres, par exemple en créant un lien de nature différente,
- soit en créant entre l'objet et ses ancêtres un nouveau type d'objet.

La première approche est utilisée dans le système BD-CAO par Dominique Rieu [RIE85]. La deuxième a été spécifiée dans la cadre du projet ACACIA [CET84].

4.5 Formalisation.

Dans tout ce qui suit, on considère qu'un schéma est formé de :

- règles de spécification des objets, qui définissent leur type ou encore leur structure. La première règle du schéma suivant par exemple est une règle de spécification de l'objet "personne"

```
(x,i,n,s,a) [ personne(x)  <- ident(x,i)  ∧ nom(x,n)  ∧ sexe(x,s)
                              ∧ age(x,a)  ∧ entier(i) ∧ chaîne(n)
                              ∧ (s=F V s=M) ∧ entier(a) ]
      (x) [ homme(x)       <- personne(x) ∧ sexe(x,M) ]
      (x) [ femme(x)       <- personne(x) ∧ sexe(x,F) ]
  (x,p,m) [ enfant(x)      <- personne(x) ∧ homme(p) ∧ pere(x,p)
                                          ∧ femme(m) ∧ mere(x,m) ]
```

- règles de déduction de propriétés. Par exemple un père est nécessairement un homme :

(x,p) homme(p) <- pere(x,p)

- contraintes d'intégrité concernant les valeurs des propriétés des objets. Par exemple le poids des hommes de plus de 50 ans est supérieur à 70 kilos :

(x,d,a) supérieur(d,70) <- homme(x) ∧ age(x,a) ∧ supérieur(a,50) ∧ poids(x,d)

L'ensemble des règles de déduction et l'ensemble des contraintes d'intégrité peuvent être vides. Les quantificateurs existentiels sont supposés être éliminés par skolémisation. Les fonctions de skolem correspondantes sont supposées être stockées explicitement dans la base de données [CHA86].

Toutes les règles et contraintes sont restreintes ici à des clauses régulières non récursives [BAN85]. En d'autres termes, les seules règles et contraintes admises sont des conjonctions de disjonctions de formules atomiques comportant un et un seul littéral positif et où toutes les variables sont universellement quantifiées :
(x1) (x2) ... (xn) ∧ (Fi1(xi1,...xin) V ~Fi2(xik,...xip) V ~Fim(xij,...,xit)).

Dans ce contexte, un objet est alors défini par un littéral positif comportant une seule variable. Par exemple personne(x) spécifie une personne quelconque x.

On veut par la suite pouvoir traiter, c-à-d interroger et modifier, des objets incomplets. Il est clair que personne(x0) permet de définir l'élément x0 de l'ensemble personne, mais ne permet pas de connaître à lui seul ses propriétés (nom, age, etc) ni de les manipuler.

On s'intéresse au cas où seulement une partie de ces propriétés sont valuées. Plusieurs phénomènes peuvent alors se produire :

- certaines règles de spécifications peuvent être satisfaites, même en présence d'informations incomplètes sur les objets. Par exemple, on peut affirmer que personne(x0) est vrai sans rien connaître d'autre sur x0,

- des incohérences peuvent être détectées parmi les règles de déduction de propriétés, même en présence d'informations incomplètes. Par exemple affirmer que pere(x0,p) et sexe(p,'F') sont vrais simultanément est incohérent avec les définitions de pere et de homme,

- des contraintes d'intégrité peuvent être insatisfaisables. Par exemple supérieur(d,70) est indécidable si l'on

ne connaît pas le poids d'un homme de 50 ans.

Par opposition à l'hypothèse classique du monde fermé, dans laquelle toute information non démontrable est considérée comme fausse, on se place donc ici dans une hypothèse de monde ouvert.

On s'attachera par la suite à montrer que des méthodes de contrôle de cohérence originales peuvent être définie dans ce cadre (Chapitre 6).

4.6 Hiérarchie de contraintes.

Les contraintes sémantiques permettent de définir et maintenir la cohérence sémantique d'un ensemble d'informations. En général, les études menées dans le domaine des bases de données se sont attachées à la cohérence sémantique d'un ensemble de données contenues dans une base vis-à-vis

- du schéma de la base de données,
- des contraintes particulières à l'application.

Dans le premier cas, il doit y avoir conformité des valeurs des données par rapport aux types et aux structures déclarées dans le schéma. Dans le deuxième cas, des procédures d'évaluation particulières sont écrites pour contrôler et éventuellement forcer certaines valeurs dans la base de données. Cette technique peut être réalisée à l'aide de "démons" ou actions spontanées déclenchées par les opérations de mise à jour.

On s'intéresse ici à un niveau supplémentaire de cohérence sémantique. Il concerne la modification du schéma de la base de données. Il concerne donc à la fois l'évolution dynamique des types et structures des données, mais également celle des contraintes qui leur sont associées.

Il y a également une autre classe de contraintes qu'il faut prendre en compte. Ce sont les contraintes d'environnement, qui influent directement sur la mise en place opérationnelle d'une application.

Ce sont les contraintes :

- topologiques,
- d'interface ou encore de dialogue avec l'utilisateur,
- de communication entre individus ou acteurs du système,
- de confidentialité.

On utilise ici le terme "acteur" à dessein pour désigner tout individu ou processus actif opérant sur ou au sein du système. La connotation qu'il présente avec la notion d'"objet" de la programmation "orientée objet" est claire.

Ces contraintes forment une hiérarchie dont le respect assure la bonne fin des traitements dans la mesure où tous les processus et acteurs les respectent (Figure 4.3).

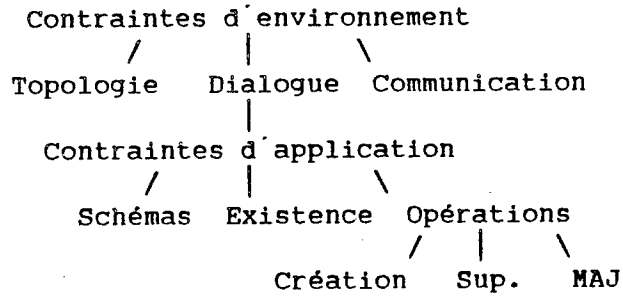


Figure 4.3 Hiérarchie de contraintes.

4.6.1 Contraintes d'environnement.

Les contraintes d'environnement sont liées à ce qui est extérieur à l'application. Cela concerne aussi bien la nature du réseau de communication qui relie les calculateurs que les caractéristiques des postes de travail.

Dans la mesure où elles définissent le système d'information sur lequel on s'appuie, elles forment une "méta-connaissance" qui le gouverne tout entier.

- Contraintes topologiques.

  Les contraintes topologiques régissent les rapports entre les différents acteurs à partir de leur implantation physique. A ce titre, ce sont les contraintes de plus bas niveau. Elle régissent en particulier les liens hiérarchiques entre sites. On verra au Chapitre V que l'on peut ignorer complètement cet aspect et développer des algorithmes de manipulation des objets qui ont tous les mêmes droits.

- Contraintes de dialogue.

Les contraintes de dialogue déterminent les fonctionnalités maxima de l'usager, indépendamment de l'application considérée. Elles particularisent les contraintes d'environnement. Elles sont fonction par exemple du type d'interaction existant avec l'usager. Elles seront différentes selon que l'on envisage un dialogue purement graphique pour du dessin ou la mise en œuvre d'un simulateur de circuit intégré.

o Contraintes de communication.

Les contraintes de communication sont en principe transparentes à l'usager et à l'application. Ce sont elles qui régissent les échanges physiques d'information. Elles ne sont pas considérées ici.

4.6.2 Contraintes d'application.

On considère dans ce qui suit deux classes de contraintes. D'une part les contraintes existentielles. D'autre part les contraintes opérationnelles. Au niveau formel de conception du schéma d'une base de données, ces deux concepts sont suffisants pour modéliser la dynamique des objets. Les contraintes existentielles permettent en effet de modéliser les dépendances fonctionnelles, multivaluées et d' inclusion. Les contraintes opérationnelles permettent quant à elles de modéliser les conditions requises pour autoriser une mise à jour et spécifier ses effets de bord. Elles spécifient le comportement dynamique des objets de manière détaillée.

Au niveau pratique de l'implantation d'une application, deux notions sont également nécessaires. Ce sont les contraintes calculées et les liens.

o Contraintes calculées.

Les contraintes calculées permettent d'exprimer des dépendances non explicites dans la bases de données. Par exemple que l'impédance totale de deux circuits en série est égale à la somme de leurs impédances.

o Liens.

Les liens permettent quant à eux de modéliser et de matérialiser des contraintes de manière explicite, par exemple la relation qui existe entre les points origine et extrémité d un segment et sa longueur [RIE85]. En général,

tout lien peut être modélisé par une contrainte calculée. Par exemple, la longueur d'un segment, propriété de l'objet segment, est égale à la racine carrée de la somme des carrés des différences de leurs coordonnées. L'inverse est par contre faux dans le cas général.

On avait émis l'hypothèse que les liens de composition et d'équivalence par exemple était suffisants pour modéliser le comportement dynamique des objets [NGU85]. Ceci s'est avéré insuffisant, en particulier dans le cadre des applications CAO [RIE85]. Ils doivent être complétés par des liens "fonctionnels", qui permettent de prendre en compte les objets calculés.

4.7 Gestion de contraintes.

Les modèles de données usuels ne procurent en général que des possibilités limitées d'expression de contraintes d'intégrité. Elles font partie du schéma de la base de données quand il s'agit de contraintes de strucutre ou de type. Elles sont exprimées de manière indépendantes lorsqu'il s'agit de contraintes dynamiques, c-à-d qui sont fonction d'un changement d'état de la base de données.

L'intérêt d'une définition déclarative des contraintes provient d'une part de l'indépendance qu'elle permet vis-à-vis des méthodes d'évaluation mais aussi de leur' indépendance vis-à-vis des modèles de données.

Diverses solutions ont été proposées dans ce sens [MOR83, NIC79, SIM84]. L'objectif est de caractériser des formulations versatiles et des méthodes d'évaluation efficaces qui permettent de modifier l'ensemble des contraintes dynamiquement.

On peut distinguer les formulations basées sur des langages particuliers du type démons ou "trigger", sur la logique [NIC79, SIM84] et des formulations spécifiques du type "equations" paramétrables [MOR83].

La caractérisation des contraintes généralement admises se fait entre :

- contraintes interprétées qui sont traitées à l'exécution des opérations de mise à jour,
- contraintes compilées qui sont transformées en une forme plus adaptée à l'exécution et en général simplifiées (soit par réduction des tests à effectuer, soit par réduction des informations à balayer),
- contraintes parallèles évaluables simultanément par activation d'un évènement commun,
- contraintes enchaînées qui s'activent en cascade,

- contraintes à évaluation immédiate qui supposent une propagation en chaînage avant des effets d'une mise à jour,
- évaluation différée qui s'appuient sur un chaînage des contraintes pour réévaluer celles qui sont dépendantes.

Par la suite, on utilise pour la certification des opérations (Chapitre VI) :

- une expression des contraintes en logique du premier ordre,
- l'interprétation de ces contraintes, ,br - une évaluation à priori dont le résultat peut éventuellement être ignoré.

4.8 Application à la conception de circuits intégrés.

Cet exemple est simplifié car le but de ce paragraphe n'est pas de spécifier un outil opérationnel, mais plutôt de montrer sur un cas concret comment nos propositions sont adaptées à la gestion d' informations complexes dans une application avancée.

On montre en particulier que les contraintes propres à chaque niveau de représentation d'un circuit ainsi que les contraintes inter-niveaux peuvent être spécifiée dans le même formalisme que celui qui est utilisé pour définir les composants de base et les aggrégats successifs générés au cours de la conception.

Dans la mesure où il est possible d'exprimer ici la sémantique des traitements à travers toutes les étapes de la conception, il n'est plus nécessaire d'avoir recours à des extracteurs propres à chaque étape. De ce fait, seule la nécessité de réutiliser des logiciels existant conduira à maintenir leur existence au-delà des possibilités offertes par la technologie actuelle.

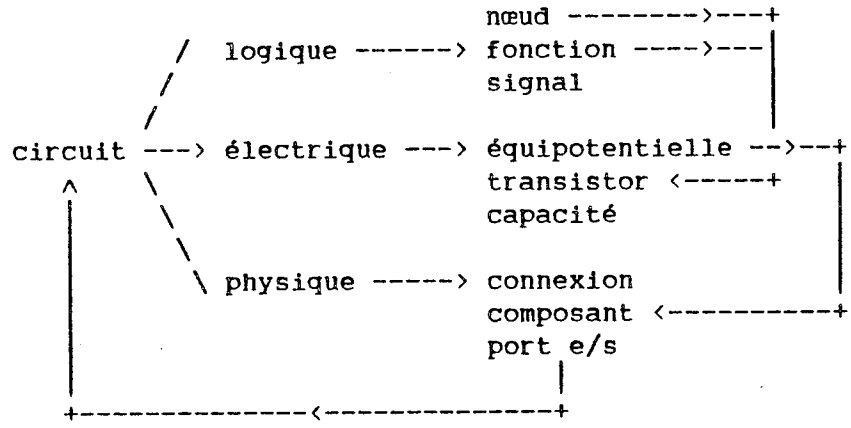
La modélisation présentée ici favorise la conception descendante d'un circuit à partir de ses spécifications fonctionnelles. On ne précise pas les éléments qui permettraient d'affiner la conception. On se contente de définir un générateur qui explore brutalement toutes les solutions possibles répondant à des spécifications données. Fondamentalement, cette modélisation est similaire à celle de RESIDUE [FIN85].

Un circuit est défini comme une aggrégat de trois représentations différentes :

- une représentation logique,
- une représentation électrique,
- une représentation physique.

On a confondu ici les niveaux fonctionnels et logiques qui sont habituellement utilisés en CAO.

Du point de vue logique, un circuit est constitué de nœuds qui réalisent chacun une ou plusieurs fonctions qui sont reliées entre elles par des signaux. Les nœuds sont réalisés par des assemblages de composants élémentaires qui réalisent des opérateurs de base (ET, OU, NONET, ...).



Du point de vue électrique, les opérateurs sont constitués de transistors et de capacités qui ont des caractéristiques propres (taille électrique, ...). Ces propriétés sont utilisées pour contrôler qu'un certain nombre de règles d'implantation sont respectées et qu'aucun élément n'est superflu. On utilise pour cela des règles de conception et de simplification qui sont également définies ci-dessous.

Du point de vue physique, un circuit est constitué d'un certain nombre de composants reliés par des équipotentielles à travers des ports d'entrée-sortie.

Par souci de simplicité, on définit un circuit depuis ses aggrégats les plus complexes jusqu'aux composants élémentaires.

Conformément à la démarche exposée précedemment (paragraphe 4.2), chaque entité est définie à l'aide de son type et du constructeur associé. On notera que l'appel à une clause telle que "circuit(C)" provoque la génération des instances d'objets (composants et composés) par évaluation des prédicats qui les définissent.

```
circuit(C) :- constructeur(C), gensym(C), asserta(circuit(C)).
```

où gensym est un prédicat évaluable qui génère à chaque appel i la valeur Ci.

```
aggregat(circuit).
constructeur(circuit(C)) :- schéma_logique(C,L),
                            schéma_électrique(C,E),
                            schéma_physique(C,P).
```

Un schéma logique est défini par une liste de nœuds et de signaux.

```
aggregat(schéma_logique(C,L)) :-     constructeur(schéma_logique(C,L)),
                                     gensym(L),
                                     asserta(schéma_logique(C,L)).
constructeur(schéma_logique(C,L)) :- lsignaux(C,L,S), lnœuds(C,L,N).
```

Un signal S est associé à un schéma logique L.

```
aggregate(lsignaux).
constructeur(lsignaux(C,L,[X|Y])) :- signal(L,X), (Y=[] ;
                                                    lsignaux(C,L,[Y])).
```

Une liste de nœuds est constituée de nœuds, chacun étant associé aux fonctions qu'il réalise.

```
aggregate(lnœuds).
constructeur(lnœuds(C,L,[X|Y])) :- nœud(X,_) , (Y=[] ;
                                                 lnœuds(C,L,[Y])).

nœud(N,[X|Y]) :- fonction(N,F), (Y=[] ; nœud(N,[Y])).
```

Une fonction F est associée a un nœud.

```
constructeur(fonction(N,F)) :- nœud(N,_), carac_fonc(F,T,C,NE,LE,S).
```

Une fonction est caractérisée par son type T, son code C, son nombre d'entrées NE, sa liste d'entrées LE et sa sortie S.

```
carac_fonc(F,T,C,NE,LE,S) :- type(F,T), code(F,C), nb_entrées(F,NE),
                             liste_entrées(F,L), sortie(F,S).
```

Les schémas électriques et physiques sont définis de manière similaire.

Un certain nombre de règles de conception sont également modélisables selon ce même formalisme. On en donne ci-dessous quelques-unes. On suppose qu'une occurrence de transistor est modélisée à l'aide du fait : trans(n,s,g,d,t,wl), où n est l'identification du transistor, s, g et d sa source, sa grille et son drain respectivement, t son type (enrichi ou depleté) et wl sa taille électrique.

Deux transistors enrichis en série de taille électrique T1 et T2 sont remplacés par un seul transistor de taille T = T1*T2/T1+T2.

```
R1(N1,N2) :- trans(N1,S1,_,_,enrichi,T1), trans(N2,S2,_,_,enrichi,T2),
             N1 ~= N2, retract(trans(N1,_,_,_,_,_)),
             retract(trans(N2,_,_,_,_,_)),
             T3 is T1*T2, T4 is T1+T2, T is T3/T4,
             asserta(trans(N,S,D,G,enrichi,T)).
```

Deux transistors en parallèle de taille T1 et T2 sont remplacés par un transistor de taille T=min(T1,T2).

```
R2(N1,N2) :- trans(N1,S1,_,_,enrichi,T1), trans(N2,S2,_,_,enrichi,T2),
             N1 ~= N2, T is min(T1,T2),
             retract(trans(N1,_,_,_,_,_)), retract(trans(N2,_,_,_,_,_)),
             asserta(trans(N,S,D,G,enrichi,T)).
```

La taille électrique T=W/L d'un transistor dont la grille est reliée à l'alimentation VDD uniquement par des transistors est réduite de moitié.

```
R3(N) :- trans(N,S,G,D,enrichi,T),
         T2 is T/2, retract(trans(N,_,_,_,_,_)),
         asserta(trans,(N,S,G,D,enrichi,T2)).
```

Une équipotentielle reliée à l'alimentation VDD par un transistor depleté de taille électrique T1 et à la grille d'un transistor enrichi doit être relié à la masse VSS par un réseau de transistors de taille T2 telle que 3*T1<=T2<=4*T1.

```
R4(N)         :- entree-sortie(Nvdd), trans(N,S,G,D,depleté,T1),
                 grille(S), vss-enr(S,T1).
grille(G)     :- trans(N,S,G,D,enrichi,T).
vss-enr(S,T)  :- vss(S), taille(S,T).
taille(S,T)   :- trans(N,S,G,D,enrichi,T2), 3*T<=T2, T2<=4*T.
```

Il est évident par contre que les programmes de génération automatisée d'une implantation physique à partir de spécifications électriques, ou encore les programmes de test des circuits par simulation ne sont pas réalisables avec cette méthode.

Il importe donc que les outils de modélisation puissent faire appel à des outils spécifiques à la conception. En particulier, les outils de conception de circuits sont fonction de la technologie employée. Ils sont donc extrêmement particuliers et difficilement intégrables dans une approche unifiée. L'architecture d'un système de Base de Données Avancées destinée à des applications comme la CAO doit donc tenir compte de ce fait. Ceci est détaillé dans les paragraphes qui suivent.

4.9 Architecture générale.

La figure 4.8 décrit l'architecture fonctionnelle des différents éléments qui interviennent dans la gestion de la sémantique.

Au cœur de l'ensemble se trouve la base de méta-connaissances qui pilote le système (BMC). Elle est implantée en logique du premier ordre, ce qui veut dire que tous les acteurs du système sont spécifiés en logique des prédicats. Leur comportement et leurs interactions également. On y touve spécifiés les enchaînements de contrôles qui doivent être opérés dans toute interaction entre deux acteurs.

Les bases de connaissances (BC) auxquelles elle accède spécifient les contraintes à respecter dans chaque environnement particulier (dialogue interactif, communication entre plusieurs acteurs, modélisation, accès et manipulations d'objets).

La base de connaissance "modélisation & stockage" est utilisée pour accéder aux informations des bases de données. Son rôle est de fournir les systèmes de réécriture qui assurent la correspondance entre les modèles de données utilisés et la logique du premier ordre.

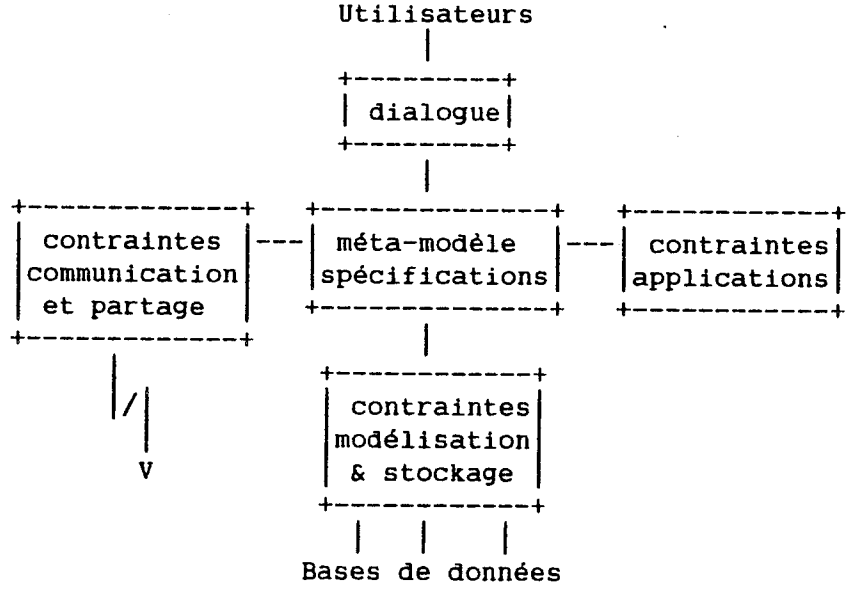


Figure 4.8 Architecture fonctionnelle.

De manière plus précise, et en faisant abstraction des modules de communication et de partage, deux réalisations ont été entreprises sur ces principes.

La première concerne le contrôle de contraintes sémantiques sur le SGBD TIGRE. Elle est décrite au Chapitre VI. La deuxième con-

cerne le système BD-CAO [RIE85].

Bien que les détails pratiques de cette deuxième réalisation soient parfois différents, le principe reste identique. Il s'agit de faire coopérer un ou plusieurs SGBD avec un système logique chargé d'un certain nombre de contrôles sémantiques, concernant par exemple la modification de schémas d'application ou les données elles-mêmes.

Ces contrôles sont effectués par un processeur logique de validation soumis à un système de déduction (Figure 4.9). Les règles de vérification sont stockées dans une ou plusieurs bases de règles, et codées sous forme de clauses de Horn. Le processeur peut faire appel aux SGBD pour extraire des informations. Ceci est nécessaire aux contrôles des opérations de mise à jour et aux calculs de leurs effets de bord. Les outils spécifiques à l'application restent accessibles de manière indépendante. Il est toutefois souhaitable qu'ils fassent l'objet de protocoles d'accès qui soient également gérés par le système logique.

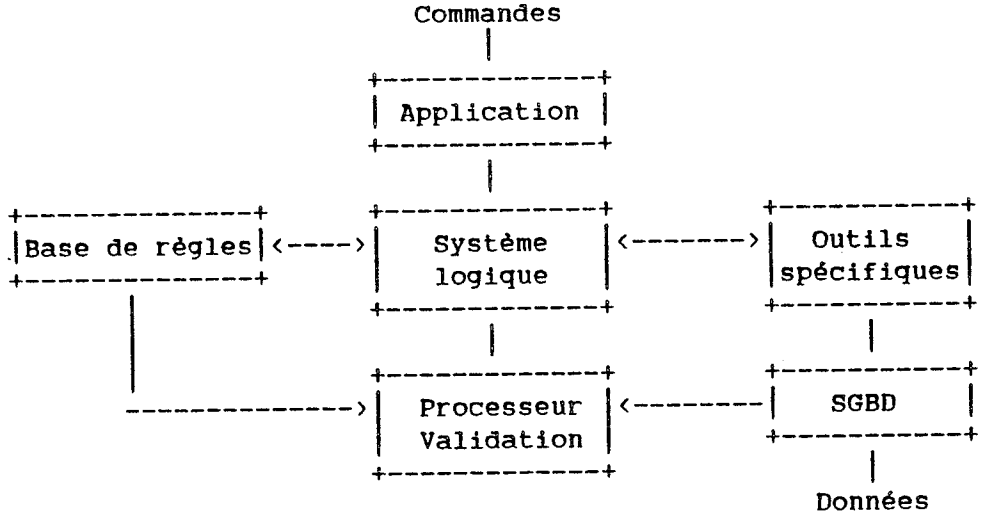


Figure 4.9 Réalisation.

Les interactions entre la base de méta-connaissances et les bases de connaissances sont spécifiées à l'aide de protocoles d'échanges.

Il existe par exemple un protocole d'échange pour chaque type de contrainte (existentielle ou opérationnelle) et pour chaque type de propriété d'un objet (extension, intension).

Lorsque le schéma d'une application est modifié, le dialogue entrepris par l'usager est coordonné par les (méta)-règles qui activent les protocoles associés aux opérations demandées en fonction des éléments touchés (Figure 4.10).

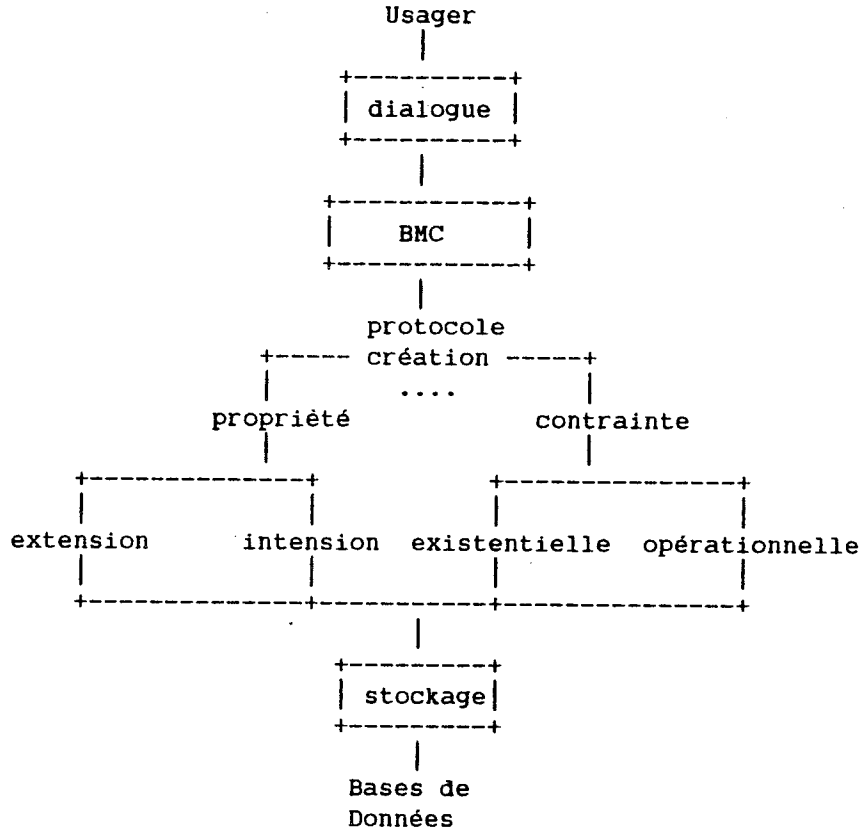


Figure 4.10 Protocoles d'échanges BMC-BC.

4.10 Conclusion.

Si la modélisation de structures de données simples et de contraintes d'intégrité élémentaires ne pose aucun problème particulier, celle de contraintes sémantiques évoluées demeure un point d'achoppement dans la plupart des modèles classiques.

Les caractéristiques des applications avancées impliquent de surcroît des besoins nouveaux comme des procédures de rétablissement automatique de la cohérence ou la propagation d'effets de bord.

Traditionnellement, ceci était réalisé à l'aide de programmes ad-hoc. La lourdeur de ce procédé nécessite des méthodes plus élaborées.

Par ailleurs, la séparation entre schéma et contraintes devient caduque avec les mécanismes de spécifications déclaratives disponibles aujourd'hui.

La modélisation en logique permet dans ce cadre une uniformité de spécification et de représentation inégalées. Par ailleurs, ses relations avec les langages de traitement symbolique permettent d'entrevoir une implantation très rapide de systèmes que l'on peut optimiser par la suite en fonction des besoins. On verra au Chapitre VI un exemple d'évaluation de contraintes original basé sur ces principes.

CHAPITRE V

REPARTITION

5.1 Couplage faible.

5.2 Hypothèses et objectifs.

5.3 Localisation dynamique.

5.4 Décomposition adaptative.

5.5 Décomposition mixte.

5.6 Limites.

5.7 Comparaison avec d'autres approches.

5.8 Conclusion.

CHAPITRE V

REPARTITION

L'intégration d'outils différents en un ensemble homogène tel qu'un système intégré de CAO pose plusieurs types de problèmes.

Au niveau conceptuel, l'existence de moyens de traitement d'origines différentes implique une coopération à travers des interfaces qui permettent :

- de traduire les concepts manipulés par un outil dans un modèle compréhensible par les autres,
- d'échanger de l'information entre eux,
- de déclencher l'exécution de l'un quelconque de ces outils et d'en récupérer les résultats.

Au niveau opérationnel, cela veut dire organiser cette coopération de manière efficace et ordonnée.

Une première solution est fournie par l'utilisation d'un SGBD classique pour centraliser et contrôler les accès aux données dispersées dans de multiples fichiers utilisés par les outils CAO. C'est la première étape du projet ACACIA [CET84].

Une approche techniquement plus élaborée consiste à étendre les SGBD classiques et à les utiliser comme dépositaires de toutes les informations stables et valides. Ils contiennent alors toute l'information de base à partir de laquelle chaque outil peut extraire des données à modifier ou les données de référence (composants prédéfinis par exemple). C'est cette approche qui a été adoptée dans le projet CVT, avec le développement sur le SGBD relationnel MICROBE d'interfaces spécifiques pour les outils CAO [JUL86].

Une troisième approche consiste à décentraliser la gestion de l'information. Par opposition à la précédente, qui est centralisatrice dans la mesure où le SGBD est le point de référence et de stockage permanent de toute information cohérente, ceci nécessite le développement de nouvelles techniques de coopération entre des moyens répartis.

5.1 Couplage faible.

La répartition peut être vue tant d'un point de vue logique que d'un point de vue physique. En supposant les problèmes de communication physique résolus, il reste un domaine où la notion d'application répartie est encore à définir et à mettre en œuvre. C'est celui des protocoles d'accès et de mise à jour au niveau des objets manipulés par les utilisateurs. C'est ce qui correspond au niveau 7 de la norme OSI d'interconnexion des systèmes ouverts. Celui des protocoles d'application. On sait depuis longtemps transférer des messages et des fichiers. On sait depuis moins longtemps transférer des relations (au sens du modèle de données relationnel). On ne sait pas encore transférer un circuit intégré dans la jième version de la ième alternative de sa représentation électrique.

La technologie des bases de données réparties, et en particulier des bases de données relationnelles réparties, a fait l'objet de nombreux travaux au cours desquels le Laboratoire de Génie Informatique et l'IMAG ont tenu un rôle de premier plan [NGU..b].

Ceci concerne aussi bien :

- la gestion de copies multiples de données,
- l'accès à des données réparties par la décomposition de requêtes,
- la mise à jour d'informations réparties.

On sait que selon la nature et la topologie du réseau de communication qui relie les calculateurs, on peut développer des algorithmes de gestion de données adaptés à chaque type de répartition : réseau en point à point, réseau à commutation de paquets ou réseau local à diffusion. L'incidence du type de réseau dont on dispose est très importante sur le plan pratique, car les vitesses de transmission peuvent varier dans un rapport de 1 à 100 selon les technologies (par exemple entre réseau à commutation de paquets et réseau local à diffusion). Certains algorithmes d'accès aux informations sont donc reservés à certains types de réseaux, comme les algorithmes de décomposition statique de requêtes pour les réseaux à commutation de paquets.

On se place ici délibérément dans un environnement du type réseau local à diffusion. L'architecture générale est basée sur l'interconnexion à l'aide d'un réseau à large bande passante (1 à 10 Mbits/sec) de postes de travail individuels et de serveurs spécialisés (serveur de données, outils de simulation de circuits, imprimante graphique, etc).

On suppose également que les postes de travail individuels sont suffisamment puissants pour supporter des logiciels graphiques interactifs et une base de données personnelle, c-à-d qu'ils sont

capables de gérer des éditeurs graphiques évolués ainsi que quelques dizaines de millions d'octets d'informations en mémoire secondaire (Figure 5.1).

L'expérience montre que pour des applications en CAO de circuits VLSI, un poste individuel monoprocesseur à base de 68000 est insuffisant, si ce n'est avec l'adjonction de dispositifs matériels spécialisés. Ceux-ci peuvent être des outils de filtrage de données qui travaillent à la vitesse de transfert des canaux d'entrée/sortie comme VERSO [SCH85], ou encore des processeurs de gestion d'écran graphique.

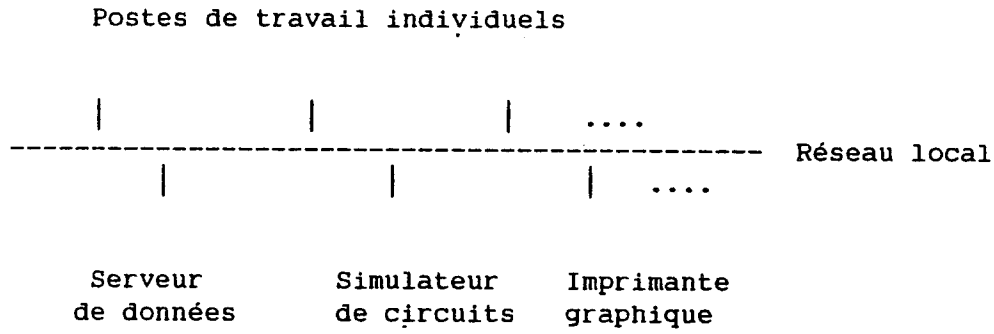


Figure 5.1 Environnement général.

Notre objectif dans ce chapitre est d'utiliser la technologie des bases de données réparties pour la conception d'applications avancées. On étudie en particulier l'impact des algorithmes d'accès et de mise à jour d'informations réparties sur les méthodes de gestion des données dans les applications avancées.

Habituellement, les systèmes intégrés de CAO font une distinction très nette entre serveurs et postes de travail. Ils sont en général faiblement couplés, c-à-d que les échanges d'informations sont ponctuels et régis par des rapports hiérarchiques de type maître-esclave. Un transfert de données est effectué pour répondre à une nécessité opérationnelle immédiate (édition graphique interactive d'un circuit sur un poste individuel, ou simulation électrique sur un serveur par exemple). Il doit être demandé explicitement par l'utilisateur au site possédant l'information ou le logiciel adéquat [CET84].

> On veut s'affranchir ici de cette contrainte et introduire la notion de couplage faible non hiérarchisé. C'est à dire que tous les utilisateurs ou concepteurs ont à priori les mêmes prérogatives. De plus, tous les serveurs du système réparti sont considérés comme des outils individuels accessibles par chacun à tout moment, comme s'ils étaient des composants banalisés disponibles sur chaque poste de travail. La répartition est invisible pour l'utilisateur.

Cette approche a deux conséquences immédiates : il faut d'une part définir des protocoles d'accès et d'échanges d' informations qui soient dynamiques. En effet, il n'est plus possible alors de se baser sur une priorité liée à la hiérarchie maître-esclave. D'autre part, les conflits d'accès qui résultent de cette mise à plat des droits d'accès crée des conflits qui n'existaient pas auparavant.

On décrit une méthode unifiée d'accès et de mise à jour d' informations réparties basée sur la programmation dynamique. Cette méthode est utilisée en recherche opérationnelle pour la résolution de problèmes décomposables en étapes discrètes, par opposition à la programmation linéaire, où les notions de temps et d'étapes n'interviennent pas [SAK83]. On verra qu'elle est particulièrement bien adaptée à l'évaluation de questions et à la mise à jour de données réparties.

Cette étude, qui a débuté à l'IMAG dans le cadre des projets en Bases de Données Réparties, a été reprise par la suite dans d'autres projets étrangers. Elle se limitait toutefois au problème de l'exécution et de la décomposition de requêtes relationnelles dans une base de données relationnelle répartie [NGU82b].

On étend ici cette démarche en adaptant les protocoles aux applications avancées. Cela veut dire qu'on ne considère plus le niveau d'exécution des opérateurs relationnels, mais celui de l'accès et de la modification de données de haut niveau, telles qu' elles sont perçues et manipulées par les utilisateurs.

5.2 Hypothèses et objectifs.

Partager des informations dans un environnement réparti nécessite la coordination de trois étapes :

- la localisation des données élémentaires,
- l'exécution des traitements s'y rapportant,
- le regroupement des résultats intermédiaires.

Ces trois phases peuvent être réitérées comme c'est le cas pour les bases de données relationnelles réparties. Les résultats intermédiaires sont alors des relations qui peuvent être à leur tour considérées comme des données élémentaires.

On va utiliser ici une méthode d'exécution des opérations d'accès aux informations réparties qui a été mise au point dans ce contexte. Elle est basée sur l'hypothèse du couplage faible non hiérarchisé. Tous les processeurs, qu'ils soient dédiés aux util-

isateurs ou aux serveurs ont les mêmes droits et privilèges. Dans cet environnement, l'information à jour peut évoluer parallèlement sur plusieurs sites et il importe de prendre en compte un mécanisme de gestion de versions réparties. Si de plus les objets manipulés possèdent plusieurs représentations simultanées, la reconstitution d'une partie ou de la totalité d'un objet doit faire appel à des mécanismes de localisation dynamique : l'information à jour peut migrer d' un site à un autre de façon aléatoire, et il est inutile que chaque utilisateur en soit constamment informé.

Il faut donc rechercher l'information là où elle se trouve. Sa localisation, sauf pour les éléments stables et définitivement figés, n' est pas disponible sur un catalogue central.

On s'inspire ici d'une méthode de localisation dynamique qui a été proposée initialement pour les bases de données réparties et qui a été reprise depuis dans d'autres projets. Elle est basée sur la décomposition dynamique de requêtes, exprimées sous forme d'arborescences d'opérateurs de l' algèbre relationnelle.

Par opposition aux méthodes de décomposition statique, qui se basent en général sur des données statistiques relatives aux relations de base, et sur les cardinalités prévisibles des résultats intermédiaires, on s' efforce ici de ne pas utiliser ces informations. En effet, dans des applications du type CAO, il est illusoire de vouloir déterminer à priori ce type de paramètres, car :

- le volume de données dépend essentiellement de la représentation (électrique, logique, etc) et de l'outil utilisés (simulateur, etc), qui sont fixes pour un concepteur donné,
- par contre ce volume n'est en aucun cas aléatoire : un circuit qui comporte des centaines de milliers de transistors ne sera pas visualisé à l'aide de quelques dizaines de lignes sur un écran dans certains cas, et de quelques milliers dans d' autres cas.

Ce qui importe ici, ce n'est pas le volume de données que l'on sait être important et stable, c'est la localisation, qui devient aléatoire, et le coût d'accès élémentaire à l'information.

La méthode d'accès aux informations s'inspire de la programmation dynamique. C'est une méthode d'optimisation utilisée en recherche opérationnelle pour les problèmes d'ordonnancement dans lesquels on peut décomposer l'évolution des opérations en un nombre fini d'étapes séquentielles [SAK83]. Ceci la différencie des méthodes basées sur la programmation linéaire, dans lesquelles la notion

de temps ou d'étapes n'intervient pas.

De par sa nature, il aurait mieux valut selon certains auteurs appeler cette méthode "optimisation récursive". Elle est particulièrement bien adaptée à l'évaluation d'arborescences d'opérateurs de l'algèbre relationnelle qui sont récursives par définition. Cette approche a été reprise par d'autres projets pour l'ordonnancement des opérations relationnelles adressées à un SGBD, par exemple l'évaluation de séquences d'opérations de jointure [WON85].

Elle est applicable à tout problème justiciable du "principe d'optimalité". Ceci signifie que toute décision prise à l'étape i doit être optimale au vu de la fonction à optimiser et ne dépend que de l'état atteint à l'étape i-1. De ce fait, toute optimisation globale nécessite que les sous-séquences de décisions soient optimales à partir de leur état initial.

Cette propriété est fondamentale. Elle signifie qu'un problème peut être résolu de proche en proche sans examiner toutes les combinaisons ou décisions possibles à l'étape i, mais uniquement celles qui s'appliquent à un état i-1 optimal. A la différence de la programmation linéaire, le nombre de contraintes que l'on peut prendre en compte ici est très faible, mais suffisant pour le problème qui nous intéresse.

> On considère dans ce qui suit que le réseau de communication entre serveurs et postes de travail est une ressource critique dont il convient d'optimiser l'utilisation. On rappelle qu'avec l'hypothèse du couplage faible non hiérarchisé, tous les processeurs ont le même droit de préemption sur le réseau.

Dans le cas d'un réseau à diffusion du type CSMA comme Ethernet, les protocoles de diffusion et de résolution des collisions sont résolus par écoute passive de tous les processeurs connectés. Si un processeur émet un message et que l'écoute ne restitue pas un message identique, c'est qu'il y a conflit à l'émission avec un autre processeur. Dans ce cas, les communicateurs reémettent le message avec un délai d'attente aléatoire qui croît avec le nombre de collisions. En réception, tous les communicateurs recoivent tous les messages et ne transmettent au processeur local que ceux qui lui sont destinés, c-à-d qui comportent en tête son adresse sur le réseau.

Les protocoles de communication sont en général micro-programmés sur les communicateurs.

> L'objectif est ici de minimiser les transferts sur le réseau afin de réduire globalement la demande de cette ressource.

Comme il n'est pas possible de fixer a priori une fonction globale de coût d'exécution des requêtes, on utilise une heuristique qui prend en compte les derniers transferts effectués sur le réseau par chaque processeur.

Cette approche a l'avantage de ne pas reposer sur la notion habituelle de transaction, qui en CAO est un concept virtuellement inutilisable vu la durée des opérations de conception.

De plus, elle ne fait pas intervenir le coût des opérations de traitement local. Ceci peut paraître surprenant, mais vient du fait qu'un grand nombre d'opérations comme la simulation électrique d'un circuit sont très longues et coûteuses (plusieurs heures CPU pour la simulation d'un circuit de quelques dizaines de milliers de transistors sur un VAX 750 sous VMS). Prendre en compte ces facteurs reviendrait à négliger complètement les coûts de transfert sur le réseau de communication, et par là-même certains temps de réponse. Ceci est tout à fait rédhibitoire pour les manipulations graphiques interactives par exemple.

5.3 Localisation dynamique.

L'accès à des informations réparties est possible à l'aide d'algorithmes de localisation qui peuvent être caractérisés en deux classes complémentaires :

- les algorithmes de localisation statique,
- les algorithmes de localisation dynamique.

Dans le premier cas, l'évaluation d'une question est faite à partir d'un plan d'exécution déterminé avant tout accès proprement dit.

Dans le deuxième cas, ce plan est déterminé dynamiquement au fur et à mesure du calcul des résultats intermédiaires.

Ces deux approches sont complémentaires. La première est basée sur des prévisions concernant en général le volume de ces résultats. La deuxième se base uniquement sur les valeurs effectives obtenues en cours d'évaluation et sur leur ordre d'obtention. Elle est particulièrement utile lorsque :

- les prévisions sont erronnées ou incalculables,
- la localisation des informations n'est pas stable, ce qui est une de nos hypothèses de base.

L'objectif est ici d'optimiser l'allocation du réseau de communication considéré comme une ressource critique. On cherche pour

cela à minimiser les transferts d'informations.

On utilise la technique de diffusion du réseau local pour les transferts physiques et pour propager les décisions prises à chaque étape de l'évaluation. Afin d'éviter les conflits de transfert lors de l'évaluation de sous-questions parallèles, tous les sites du réseau sont ordonnés en anneau virtuel.

Le principe utilisé ici consiste à décomposer progressivement les questions en sous-ensembles évaluables en parallèle. Les sites sont déterminés en fonction du volume des résultats partiels par rapport à un seuil de transfert qui agit comme garde-fou. Si le résultat intermédiaire obtenu sur un site est en volume inférieur à la valeur courante du seuil, il est transféré sur le site du nœud frère dans le graphe, si celui-ci est connu. Sinon, on attend cette information avant de décider le transfert. Si les deux résultats intermédiaires sont connus, on transfère le plus petit vers l'autre. L'ordre et le site d'obtention des résultats sont donc également pris en compte. L'intérêt de cette approche est de permettre une décomposition adaptative moyennant une fonction appropriée pour la mise à jour du seuil de transfert (paragraphe 5.4). Elle a par contre l'inconvénient de n'être efficace que lorsque les opérations binaires diminuent la cardinalité d'un résultat intermédiaire par rapport aux opérandes. Elle est donc inapplicable dans le cas d'un produit cartésien de relations.

Exemple :

Supposons la base de données construite sur le schéma suivant :

```
R1 (A,B,C) site S1 volume 1500
R2 (A,F,G) site S2 volume 1000
R3 (F,I,J) site S3 volume 1500
R4 (I,K,L) site S3 volume 1000
```

et la question Q à évaluer :

```
Q : union (intersect (R1, join (R2,R3)),
           (intersect (R4, join (R2,R3))).
```

On suppose que union, intersect et join sont les opérateurs usuels de l'algèbre relationnelle. Tous les sites sont censés pouvoir les évaluer. Cette question peut être représentée à l'aide du graphe suivant (Figure 5.2) :

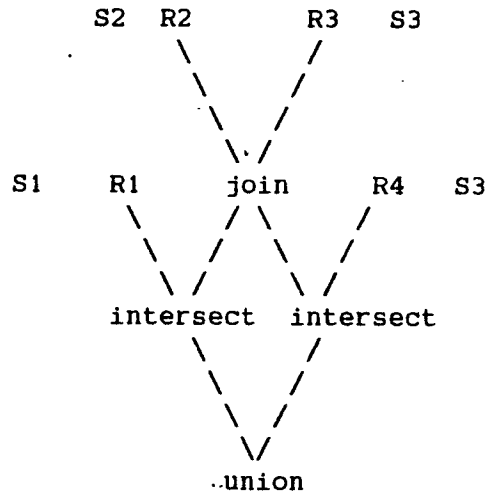


Figure 5.2 Graphe de la question Q.

Soit Tn+1 = (Tn + Rn)/ 2 la relation de recurrence qui associe la valeur Tn+1 du seuil à l'étape n+1 à la valeur du dernier transfert Rn et à la valeur Tn du seuil à l'étape n. On rappelle que les sites sont ordonnés et que la décomposition est donc évaluée par étapes séquentielles et énumérables. La valeur initiale T0 du seuil est fixée arbitrairement à la moyenne des volumes des relations de base.

La première étape consiste à déterminer les sous-arbres monosites. Sur cet exemple ce sont tous des relations de base. La première évaluation à faire est donc celle du sous-arbre join (R2,R3). Par comparaison avec T0, R2 est transféré sur le site de R3 c-a-d S3. La sous-question join (R2,R3) est donc évaluable sur S3. Comme R4 est également située sur S3, l'opération intersect (R4, join (R2,R3)) est donc également évaluable sur S3 (Figure 5.3). Le transfert de R2 vers R3 est noté : R2 ===> R3.

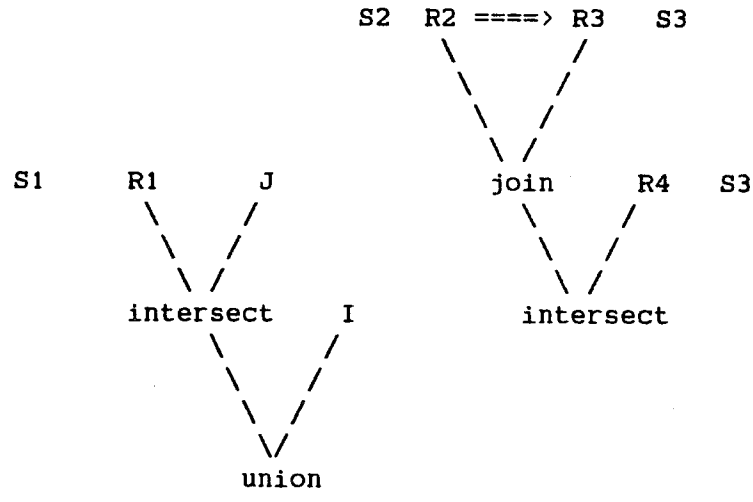


Figure 5.3 Première étape.

A cet instant, les décisions de transfert de R2 vers R3 et d'exécution du sous-graphe intersect sur S3 sont diffusées par le site S2 qui les a prises. Chaque site effectue alors les opérations correspondant à sa partie, c-a-d transfert et mise à jour du seuil pour S2, attente de R2 pour S3. La fonction de mise à jour du seuil définie précédemment donne T1 = (T0 + |R2|) /2, soit T1 = 1125, si |R| dénote le volume de R en octets par exemple, c-à-d sa cardinalité multipliée par la longueur des n-uplets.

Lorsque ce transfert est effectué, l'opération join (R2,R3) est lancée sur S3. Appelons cette opération J.

Lorsqu'elle est terminée, et en supposant que |join (R2,R3)| = 100, ces deux informations sont diffusées par S3. Comme |R1| > |join (R2,R3)|, c' est le résultat intermédiaire qui est transféré vers R1. La valeur du seuil est mise à jour par S3 à T2 = (T1 + |join(R2,R3)|) /2, c-a-d 612. Parallèlement, S3 lance l'exécution de intersect de R4 avec le résultat intermédiaire (Figure 5.4). Soit I cette opération.

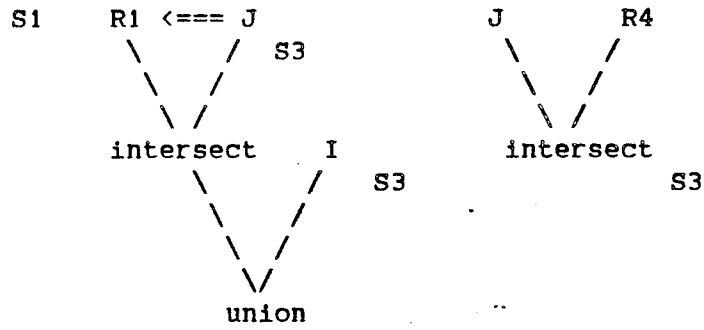


Figure 5.4 Deuxième étape.

Le site S1 attend J pour démarrer l'exécution de intersect (R1,J). Lorsque S3 a terminé intersect (J,R4), la comparaison avec la valeur courante du seuil T2 permet de déterminer le prochain transfert a effectuer. Supposons |intersect(J,R4)| = 75. On a T2 > 75, donc S3 transfert le résultat partiel sur le site S1 (Figure 5.5).

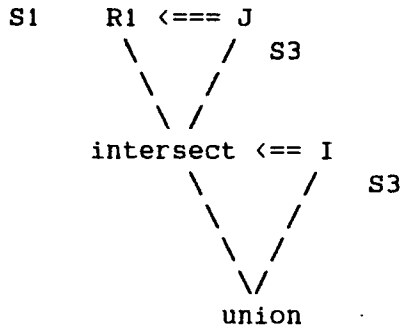


Figure 5.5 Troisième étape.

Enfin, lorsque intersect (J,R1) est terminé, S1 lance l'exécution de union (I, intersect (J,R1)) (Figure 5.6).

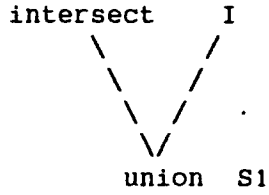


Figure 5.6 Quatrième étape.

Le résultat final est enfin envoyé à l'utilisateur.

Le principe de l'algorithme de décomposition des questions est schématisé ci-dessous. On suppose que "diffuser(message)" signifie propager le "message" à tous les sites et "mémoriser(message) signifie conserver "message" sur le site qui le reçoit. De plus, un opérateur OP a au plus deux opérandes notés FILS et RX. Ce dernier peut être un résultat intermédiaire calculé sur un site distant. Loc(P) est l'évaluation du site d'exécution de l'opérateur P. |R| est le volume de la relation R :

répéter A :

1 - si réception d'une requête alors
    1.1 caractériser les sous-arbres locaux,
    1.2 envoyer les sous-arbres à leur site d exécution,
    1.3 répeter A.

2 - si réception de sous-arbre alors
    2.1 exécuter le sous-arbre localement,
    2.2 répeter A.

3 - si réception de résultat intermédiaire RX d'un autre site alors
    3.1 si FILS terminé alors continuer l'exécution locale,
    3.2 répeter A.

4 - si résultat intermédiaire local RX calculé alors

```
    4.1 si |RX| <= Seuil alors mémoriser(|RX|<=Seuil),
    4.2 sinon mémoriser(|RX|>Seuil),
    4.3 répeter A.

5 - si réception du jeton alors
    5.1 éliminer les messages envoyés précedemment,
    5.2 exécuter le paragraphe 6 (traitement des messages reçus),
    5.3 envoyer les messages en attente,
    5.4 transmettre le jeton au suivant sur l'anneau,
    5.5 répeter A.

6 - tant qu'il existe des messages à traiter localement faire

    6.1 si message est : |RX| <= Seuil alors
        6.1.1 si FILS non terminé alors
              - mémoriser(Loc(OP) <- Loc(RX)),
              - mémoriser(transférer FILS sur Loc(RX)),
        6.1.2 sinon
              - si |FILS| < |RX| alors
                . mémoriser(transférer FILS sur Loc(RX)),
                . mémoriser(Loc(OP) <- Loc(RX)),
              - sinon
                . mémoriser(Loc(RX) <- Loc(FILS)),
                . mémoriser(Loc(OP) <- Loc(FILS)),
              - mettre à jour le Seuil,
              - mémoriser(Seuil),
    6.2 si message est : |RX| > Seuil alors
        si FILS est terminé alors exécuter le paragraphe 6.1.2.
    6.3 si message est : transférer FILS sur site S alors
        si FILS terminé alors
        - transférer FILS sur site S,
        - mettre à jour le Seuil,
        - mémoriser(Seuil),
    6.4 si message est : Loc(OP) <- site S alors
        mettre à jour la requête locale.

    fin tantque 6.

fin A.
```

Cette décomposition met en œuvre des transferts basés sur la comparaison des volumes des résultats intermédiaires avec la valeur courante du seuil, et dans l'ordre d'apparition.

Comme il a été dit plus haut, le volume est un critère insuffisant à lui seul, car ses variations ne sont pas aléatoire dans un contexte donné, comme celui de la CAO. Il importe donc de pondérer les variations du seuil à l'aide d'heuristiques qui prennent en compte la "connaissance" que l'on a de l'application.

Dans l'exemple précédent, la fonction d'évolution du seuil est fixée une fois pour toute, et tient compte de manière directe du dernier transfert Rn effectué sur le réseau, et de manière indirecte des transfert précédents à travers la formule :

$$Tn+1 = (Rn + Tn)/2.$$

On va améliorer cette fonction à l'aide de la programmation dynamique, ce qui permet de développer les bases d'une décomposition adaptative.

5.4 Décomposition adaptative.

L'objectif de la programmation dynamique est d'obtenir une solution optimale à des problèmes qui peuvent être décomposés en un nombre fini d'étapes k. D'une manière générale, on évite par cette technique l' exploration de toutes les combinaisons possibles de solutions. Elle permet à une étape i donnée, de n'étudier que les décisions qui auront pour effet de maintenir la fonction de coût à un optimum. Ce qui est important, c'est que ces décisions ne dépendent pas des décisions prises lors des étapes précédentes.

> Le problème à résoudre doit être entièrement caractérisé à chaque étape i par une variable d'état Si. A chaque étape, on dispose d'un ensemble de décisions possibles T(i), parmi lesquelles il faut choisir la valeur d'une variable de décision Ti qui minimise une certaine fonction de coût C. Celle-ci peut être définie comme la somme des coûts cumulés Ck de chaque étape élémentaire.

Dans le problème qui nous intéresse, le coût de transport d'une unité d'information est supposé constant et égal à c.

On veut ici évaluer de manière optimale une requête adressée à un SGBD réparti en minimisant le seul coût de transfert de l'information sur le réseau. En considérant que les requêtes sont exprimées sous formes d'arborescences binaires, le nombre de solutions possibles est s-1, si l'on a s sites différents pour les relations de base. C'est le nombre de nœuds intermédiaires d'une arbrescence binaire à s feuilles.

Une première utilisation de la programmation dynamique consiste ici à résoudre le problème d'allocation de ressources suivant :

> "Etant donné n relations (de base ou intermédiaires) réparties sur s sites différents, et étant donné un coût unitaire de transport de l'information ç allouer ces rela-

tions sur les s sites de façon à minimiser le coût total de transport".

Dans les méthodes d'exploration exhaustive le coût de détermination de la politique optimale est proportionnel à (s-1) C (s+n-1, s-1) [SAK83], où C(n,p) est le nombre de combinaisons de p éléments parmi n. Dans le cas où l'on utilise la programmation dynamique, il devient proportionnel à s(n)2/2, où s(n)2 signifie s que multiplie n à la puissance 2. Ce coût est (10)55 fois moindre que le précédent pour s=n=100.

L'inconvénient majeur de cette approche est qu'elle nécessite la connaissance des relations intermédiaires. Elle retombe donc dans l'éceuil que l'on voulait éviter, c-a-d la prévision statistique ou autre des volumes de tous les résultats.

On utilise ici une approche différente qui évite ces estimations.

On caractérise l'état instantané du système par une variable d'état Si, appelée seuil de transfert. Cette variable est mise à jour à chaque transfert d'information pour tenir compte de l'évolution du système.

Le principe est de lancer un tranfert dès lors qu'il est inférieur au seuil Si. S'il est supérieur, on attend le résultat intermédiaire suivant pour choisir le moins coûteux des deux.

Soit T(i) l'ensemble des tranferts possibles à l'étape i. Le transfert optimal Ti est choisi dans T(i). Soit Ci le coût cumulé de transfert à l'étape i. On a :

(1) Ci = min [c.Ti + Ci-1] pour tout Ti c T(i)

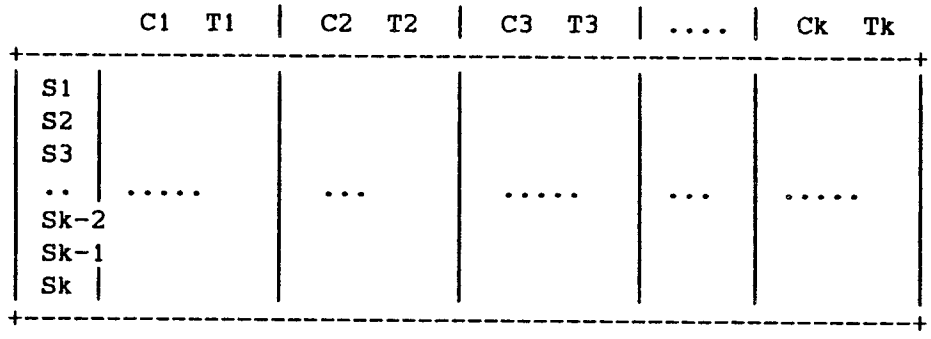
Les contraintes suivantes définissent les relations entre les variables :

(2) S0 = (Somme |Ri|)/s
(3) Si = (Si-1 + Somme Tj)*c/i pour tout 1<= j <= i
(4) Si >= Ti
(5) Sn = (Somme |Ri|)/s

Les valeurs S0 et Sn du seuil sont fixées arbitrairement à la valeur moyenne des cardinalités des relations de base. Elles peuvent être adaptées à chaque cas particulier. La contrainte (4) spécifie que les seuls transferts autorisés sont ceux qui sont inférieurs au seuil courant. La contrainte (3) définit l'évolution de la valeur du seuil à partir de sa valeur précédente et du dernier transfert.

A partir de là, il est possible de calculer un tableau donnant pour chaque valeur possible du seuil Si et chaque étape 1<= i <=k

le transfert théorique optimal Ti.



Comme l'évolution du seuil est fixée à priori par une fonction telle que (3), en cas de conflit on prend comme politique de transférer le résultat intermédiaire dont la cardinalité se rapproche le plus de la valeur du seuil. Ceci induit une erreur Ei que l'on peut calculer à chaque étape i :

$$Ei = ||Ti| - |Ri||$$

où Ri est le résultat intermédiaire réel et Ti la valeur théorique optimale à l'étape i. L'erreur cumulée pour une requête Q donnée et une politique de transferts donnée est donc

$$E = \text{Somme}\ (Ei) = \text{Somme}\ (||Ti| - |Ri||) \text{ pour tout } 1<= i <=n.$$

5.5 Décomposition mixte.

L'une des motivations qui ont présidé à la définition d'une stratégie de décomposition dynamique et adaptative est la nécessité d'avoir un garde-fou et un recours en cas d'estimations erronnées.

Une démarche synthétique consiste à combiner en un seul ensemble l'approche statique et dynamique.

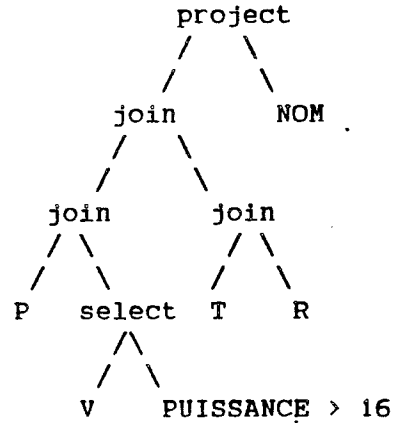
Dans un premier temps toute question est décomposée en sous-graphes locaux évaluables en parallèle. Cette décomposition initiale est basée sur les estimations du coût des opérateurs et de la cardinalité des résultats intermédiaires. Des données statistiques sur les distributions des valeurs d'attributs peuvent être utilisées si elles sont maintenues par le SGBD. Cette première étape aboutit à la définition d'un plan d'exécution statique. Il permet de lancer l'évaluation de la question.

Supposons par exemple que la base de données soit construite sur le schéma suivant :

PERSONNE : relation P(NP, NOM), site S1,

VOITURE : relation V(NV, NP, PUISSANCE), site S2,
TRAVAIL : relation T(NP, QUALIF, LIEU), site S3,
RESIDENT : relation R(NP, VILLE, RUE), site S4.

La question : "Quel sont les noms des personnes qui travaillent sur leur lieu de résidence et qui sont propriétaires d'une voiture de plus de 16 chevaux ?" est interprétée grâce à l'arborescence suivante :



La stratégie d'exécution statique est calculée en fonction de :

- l'estimation du volume des résultats intermédiaires,
- l'estimation du coût des opérations ntermédiaires.

Ceci est fait à l'aide de données statistiques telles que :

- la cardinalité des relations de base, notée card(R),
- la longueur des n-uplets des relations et de leurs constituants, notées long(R) et long(A) respectivement,
- la cardinalité des domaines de valeurs des constituants, notée card(dom(A)).

Les équations utilisées pour estimer la cardinalité des résultats intermédiaires sont (R[A] dénote la projection de la relation R sur le constituant A) :

- card(select(R,A=constante)) = card(R) / card(R[A]), si constante est dans R[A],
- card(semi-join((R1,R2)) = card(R1) * card(R2) / card (dom(A)).

La stratégie d'exécution est alors choisie par comparaison des coûts des différents algorithmes possibles.

Ainsi, le choix entre trois possibilités pour évaluer un opérateur de produit sur les sites S1, S2 et S3 ayant pour opérandes R1 et R2 situés sur les sites S1 et S2 est décidé par comparaison des équations de coûts E1, E2 et E3 suivantes. On

suppose que Cj représente le coût du produit élémentaire entre deux n-uplets. Ct représente le coût de transmission d'un élément unitaire d'information, par exemple un octet. Du fait de l'hypothèse de connexion des sites à travers un réseau local à diffusion, on suppose que ce coût unitaire est égal à 1, quelque soit le site émetteur et le site récepteur. Ceci pondère les coûts d'évaluation et de transmission d'un facteur égal, ce qui répond au besoin exprimé plus haut (paragraphe 5.2) de considérer tous les éléments intervenant dans le processus d'évalution d'une question. Vol(R) dénote le volume de la relation R, c-à-d card(R) * long(R).

E1 : sans semi-join.

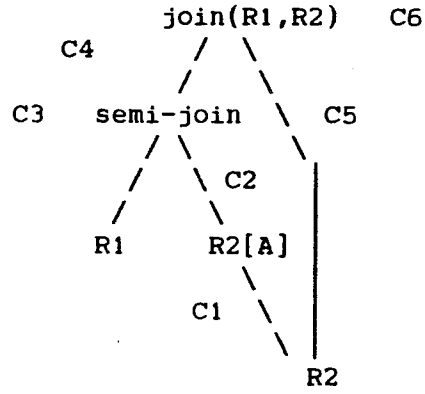
E11 : transfert de R1 et R2 vers un troisième site S3.

Coût(E1) = Ct * (Vol(R1)+Vol(R2)+Cj*(card(R1)+card(R2)).

E12 : transfert de R1 vers R2 et du résultat sur S3.

Coût(E1) = Ct * (Vol(R1)+card(R1)*card(R2)/card(dom(A))) * long(R1) + long(R2) - long(R[A]))) + Cj * (card(R1)+card(R2)).

E2 : avec un semi-join si R1[A] c R2[A]. Cpr dénote le coût de projection d'un n-uplet de relation sur un constituant.



C1 = Cpr * card(R2)
C2 = Ct * card(R2[A]) * long(A)
C3 = Cj * (card(R1)+card(R2[A]))
C4 = Ct * card(R1) * long(R1) * sel(R2,A)
C5 = Ct * card(R2) * long(R2)
C6 = Cj * (card(R2)+card(R1)*sel(R2,A)

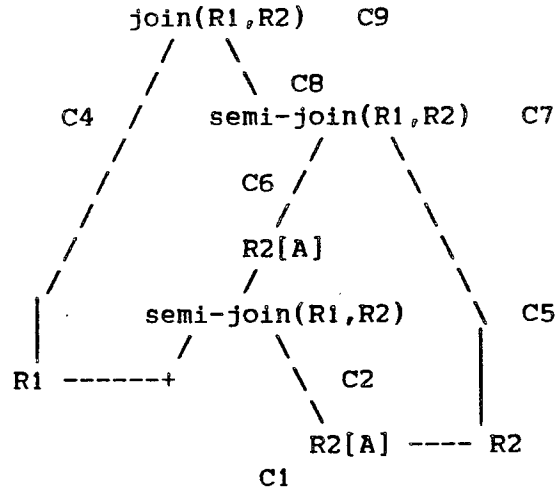
où sel(R,A) dnote la sélectivité du constituant A de la relation R, c-à-d card(R[A])/card(dom(A)).

Le coût total de E2 est donc avec Ct=1 :

Coût(E2) = card(R2[A]) * long(A) + vol(R1) * sel(R2,A) + vol(R2)

```
          + Cj * (card(R1) * (1+sel(R2,A))
          + card(R2) + card(R2[A]))
          + Cpr * card(R2).
```

E3 : deux semi-joins si ni R1[A] c R2[A] ni R2[A] c R1[A].

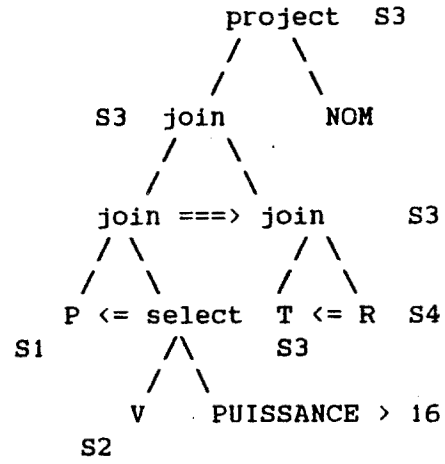



```
C1 = Cpr * card(R2)
C2 = Ct * card(R2[A]) * long(A)
C3 = Cj * (card(R1) + card(R2[A]))
C4 = Ct * vol(R1) * sel(R2,A)
C5 = Cpr * card(R1) * sel(R2,A)
C6 = Ct * ( card(R1[A]) * sel(R2,A)) * long(A)
C7 = Cj * (card(R2) + card(R1[A] * sel(R2,A))
C8 = Ct * card(R2) * sel(R1,A) * sel(R2,A) * long(R2)
C9 = Cj  *  (card(R2)  *  sel(R1,A)  *  sel(R2,A)  +  card(R1)  *
sel(R2,A)
```

Le coût total de E3 est donc avec Ct=1 :

```
coût(E3) = card(R2[A]) * long(A) * (1+sel(R1,A))
           + vol(R1)*sel(R2,A)
           + vol(R2) * sel(R1,A) * sel(R2,A)
           + Cj * (card(R2)*(1+sel(R1,A)*sel(R2,A))
           + card(R1)*(1+sel(R2,A))
           + card(R2[A])*(1+sel(R1,A))
           + Cpr * (card(R1)+card(R2)+sel(R2,A)).
```

La comparaison de E1, E2 et E3 permet de fixer les sites d'évaluation de tous les nœuds de l'arborescence initiale, donc de détermier tous les transferts de données pour la stratégie d'évaluation statique. Par exemple ici :



En utilisant le principe de mise à jour dynamique du seuil de transfert, on surveille en permanence les estimations faites et les résultats réellement obtenus. Si la déviation obtenue est supérieure à une valeur donnée, on bascule à une stratégie de décomposition dynamique (Figure 5.7).

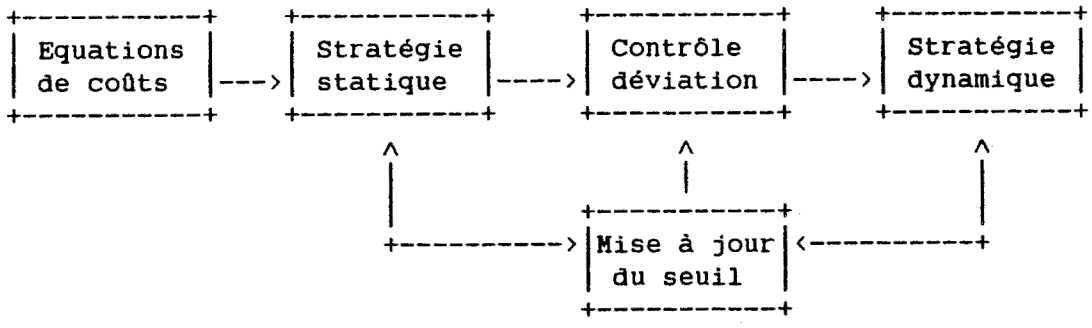


Figure 5.7 Décomposition mixte.

Le schéma complet de traitement d'une question est décrit par la Figure 5.8. On remarquera que si le plan d'exécution statique est satisfaisant, l'ensemble se ramène à une évaluation statique. Les deux sratégies sont donc complémentaires. La restructuration algébrique est basée sur les propriétés classiques des opérateurs relationnels (distributivité, commutativité, etc).

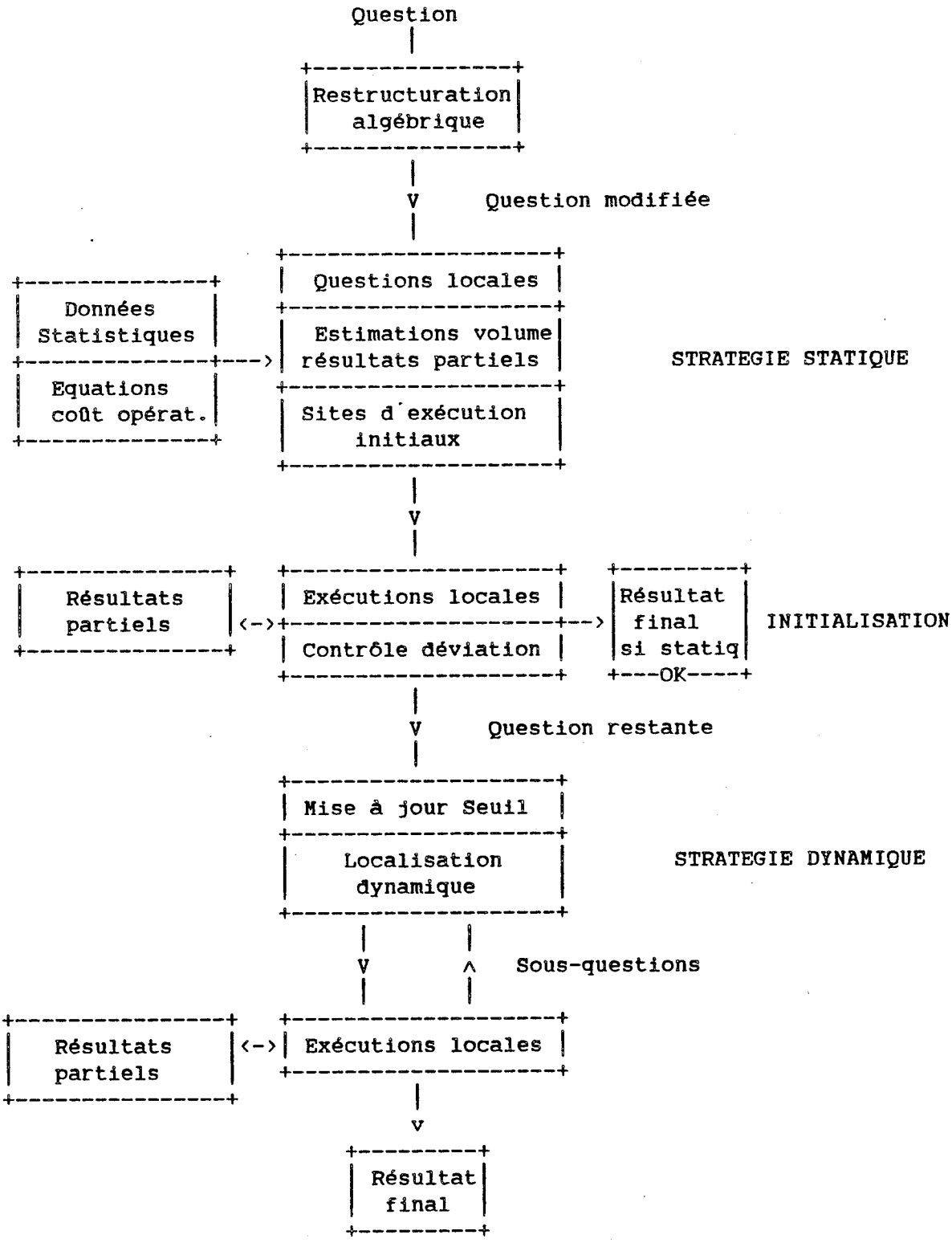


Figure 5.8 Traitement d'une question.

5.6 Limites.

La manipulation d'informations réparties nécessite également le partage des données entre concepteurs travaillant simultanément. Les mises à jour doivent être contrôlées afin d'éviter les interférences entre eux.

On connaît deux grands types de méthodes de partage de données réparties, qui sont en fait des extensions des méthodes de partage centralisées. Ce sont les méthodes de verrouillage et les méthodes d' estampillage.

Dans le premier cas, les informations que l'on désire accéder sont bloquées en écriture pour un seul utilisateur à la fois. Les autres ont éventuellement le droit de les lire. Pour éviter les interblocages, une règle générale impose qu'on ne libère jamais de ressource (c-a-d de donnée) avant d'avoir complètement terminé un travail. Par ailleurs, on peut soit requérir les données au fur et à mesure des besoins, soit en bloc en début de travail. On doit toujours les libérer en bloc en fin de travail.

Les méthodes d'estampillage n'obligent pas à réclamer les informations dont on a besoin à l'avance. Elles sont basées sur l'ordonnancement virtuel des opérations concurrentes. C-a-d que les données mises à jour sont celles qui correspondent aux informations "les plus récentes" eut égard aux estampilles.

La méthode de mise à jour proposée ici est basée sur le verrouillage. Elle est compatible avec la décomposition dynamique en ce sens qu'elle peut l'inclure comme première étape. La deuxième étape est le verrouillage propement dit.

Le point commun est l'ordonnancement en anneau virtuel des sites connectés au réseau. Il permet comme on l'a vu au paragraphe précédent de décomposer dynamiquement les questions sans conflits par octroi séquentiel du droit de décomposition à un seul et par diffusion des décisions à tous. Selon une méthode classique des systèmes répartis, il permet de demander l'usage exclusif des informations.

Ce mécanisme unique présente un intérêt non négligeable car la décomposition adaptative a pour objectif ultime l'optimisation de l'évaluation des questions. Il faut cependant être réaliste et tenir compte de l'accès partagé aux informations. Ceci implique en effet des délais d'attente qu'on sait être seulement finis dans le temps.

L'optimisation en tant que telle est donc limitée par les verrouillages requis pour les accès concurrents.

5.7 Comparaison avec d'autres approches.

L'évaluation de questions sur des informations réparties est l'objet d'un très grand nombre d'études depuis plusieurs années. Il n'est pas question de les passer toutes en revue. On donnera seulement ici quelques éléments de comparaison qualitatifs.

Dans leur quasi totalité, les méthodes d'évaluation reposent sur la génération statique d'un plan d'exécution. C'est-à-dire qu'on détermine à priori une stratégie d'exécution en se basant sur des données statistiques, comme la distribution des valeurs des attributs des relations, et sur une estimation des coûts des opérations élémentaires.

Cette philosophie est basée sur l'hypothèse que les traitements sont stables. C'est -à-dire que le profil des questions est constant, et que les variations des valeurs des données sont négligeables. Moyennant ces hypothèses, des algorithmes d'optimisation ont été proposés qui ont pour objectif de mi.imiser le coût d'évaluation des requêtes.

Ce coût est malheureusement fonction d'un nombre important de paramètres. On peut citer :

- les coûts de transfert de données entre les différents sites,
- les coûts d'exécution des requêtes partielles sur chacun d'eux,
- le type du réseau de transmission utilisé,
- la charge de ce réseau,
- la charge des sites locaux d'exécution.

Ces paramètres ne sont pas indépendants. En particulier, les deux premiers sont fonction des trois derniers. Comme il a été dit au paragraphe précédent, le partage de l'information a également une influence sur les temps d'accès à cause des verrouillages qu'il implique.

Dans la plupart des approches, à l'instar de SDD-1 et MULTIBASE, un réseau à commutation de paquets est supposé servir de moyen de transmission. Ceci implique un coût très élevé en temps de transfert de l'information.

Dans des approches plus élémentaires comme R*, le réseau est un ensemble de lignes spécialisées point à point. Ce qui implique que ces temps de transfert sont négligeables par rapport aux temps de traitements locaux.

L'avènement des réseaux locaux a peu modifié cette hypothèse, si ce n' est que le débit d'information pouvant être entre 100 et 1000 fois supérieur, l'accès au réseau lui-même peut s'en

trouver fortement ralenti.

5.8 Conclusion.

Bien que les traitements puissent être à la longue caractérisés de façon fiable, il demeure qu'aucune des méthodes statiques proposées à ce jour n'offre de palliatif à des estimations erronées sur les valeurs des données ou sur les caractéristiques des transactions.

Une approche dynamique comme celle qui est décrite ici peut alors s'avérer utile pour :

- poursuivre pas à pas une évaluation basée sur des estimations qui apparaissent erronnées en cours d'exécution,
- caractériser le comportement de nouvelles transactions,
- caractériser le comportement de transactions existantes sur des données hautement volatiles.

Cette philosophie a été largement reprise par d'autres auteurs pour l'évaluation de séquences optimales d'opérations dans les bases de données réparties [CHI84, YU86].

Par ailleurs, l'incidence de la charge instantanée des sites d'exécution sur l'évaluation des questions n'a pas fait l'objet de recherche actives en matière de bases de données réparties.

Une approche utilisant la programmation dynamique pour la répartition de cette charge dans un environnement distribué a été proposée dans [KRA80, REU86].

Il y a certainement là une voie de recherche prometteuse pour intégrer les notions de décomposition dynamique et d'optimisation du débit dans un système opératoire distribué.

Si l'aspect dynamique de l'évaluation de questions réparties a été longtemps controversé, il est à postériori largement justifié par les applications avancées comme la bureautique. On sait qu'il est possible de définir dans celles-ci des objets autonomes, dont le comportement, bien que répondant à des spécifications précises, est complètement aléatoire [TSI85]. L'objet répond en effet à des sollicitations de son environnement, et réagit en fonction de sa localisation actuelle, de paramètres internes et de son activité courante.

Ce type d'applications requiert de ce fait des outils évolués de spécification comme la logique, des modèles de données également évolués, comme l'approche orienté objet, et des traitements sophistiqués. L'évaluation dynamique de traitements qui mettent en jeu des informations réparties répond partiellement à ce

besoin.

CHAPITRE VI

CERTIFICATION

6.1 Classe d'équivalence d'un élément.

6.2 Relation d'ordre sur les classes.

6.3 Types et sous-types.

6.4 Types et contraintes : deux approches duales.

6.5 Propriétés de l'ensemble quotient.

6.6 Validité d'une opération.

6.7 Prototypes.

6.8 Certification des opérations.

6.9 Exemple.

6.10 Sous-système de validation.

6.11 Répartition.

6.12 Limites.

6.13 Comparaison avec d'autres approches.

6.14 Application : un Systéme de Base de Données Expert en CAO.

6.15 Conclusion.

CHAPITRE VI

CERTIFICATION

Une grande partie des recherches concernant les bases de données déductives ont actuellement pour but d'étendre les capacités de gestion de mémoire secondaire des systèmes déductifs pour permettre l'accès à des bases de faits ou de règles stockées dans un SGBD.

Les applications de ces systèmes et les nouvelles fonctionnalités que l'on peut en attendre est un domaine qui est très peu abordé.

On pose ici les fondements théoriques nécessaires à l'élaboration d'une méthode originale de gestion et de vérification de contraintes sémantiques dans les applications avancées.

Elle s'appuie sur deux exigences. D'une part les méthodes habituelles de vérification de contraintes proposées dans les SGBD classiques deviennent insuffisantes dans un certain nombre d'applications telles que la Bureautique ou la Conception Assistée par Ordinateur. D'autre part, les faiblesses maintenant reconnues des modèles de données classiques (relationnel et entité-association dans une moindre mesure), requièrent de nouveaux outils. On s'attachera ici à montrer que des méthodes rigoureuses de vérification de contraintes sémantiques peuvent être définies et implantées, en apportant aux systèmes de bases de données classiques des fonctionnalités supplémentaires, quelque soit le modèle de données utilisé.

Le principe utilisé ici consiste à détecter les incohérences potentielles engendrées par les opérations des usagers le plus tôt possible. On décompose pour cela le contrôle en deux phases. La première est la certification des opérations. La deuxième est leur validation.

La certification contrôle les opérations en les appliquant tout d'abord sur un prototype de l'objet à modifier. On rappelle qu'ici un objet est défini par un littéral positif comportant une seule variable (Chapitre IV : Modélisation). La validation rend les opérations effectives dans la base de données.

Ce contrôle à priori, donc pessimiste, permet de bloquer les opérations présumées incohérentes en laissant à l'usager la responsabilité de poursuivre la voie qu'il a choisie.

L'ensemble des prototypes est utilisé comme un résumé de la base de données dans lequel les erreurs concernant la structure des objets sont interdites, alors que les erreurs concernant les valeurs de leurs propriétés sont tolérées.

Une application directe de cette approche concerne la conception assitée par ordinateur, domaine dans lequel l'évolution des objets concerne aussi bien leur structure que leurs valeurs internes.

La logique du premier ordre offre ici un outil intéressant car elle permet de formaliser les concepts utilisés et de prouver leurs propriétés. On se place ici dans le cadre de la théorie de la preuve. On part de la constatation que la théorie du modèle suppose qu'à tout instant, une base de données est un modèle de la théorie définissant son schéma et ses contraintes d'intégrité. Ceci est rarement le cas dans les applications réelles. On utilise donc la théorie de la preuve.

On sait que dans ce cas, une base de données peut-être formalisée comme une théorie du premier ordre. Les axiomes sont constitués des formules qui définissent le schéma et les lois générales (par exemple les contraintes d'intégrité et les règles de dérivation d'informations implicites), ainsi que les faits élémentaires. Une extension permet d'y inclure les valeurs nulles et les informations disjonctives, mais on n'utilise pas cette possibilité ici.

L'inconvénient majeur de cette approche est en effet que la théorie doit rester cohérente à tout instant et que ceci n'est possible qu'en prenant en compte la totalité des axiomes. Cela nécessite donc un balayage systématique de toute la base de données (c-à-d de tous les axiomes de la théorie) lors de chaque mise-à-jour.

L'idée de base que l'on développe ici est de recourir à un balayage de classes d'équivalence des données et non pas de toute la base. Sous réserve que ces classes soient définies de manière appropriée et que les mises-à-jour soient traduites de façon adéquate en opération sur les classes, on évite de contrôler la cohérence de la théorie en examinant tous les axiomes, mais seulement les représentants de ces classes. C'est le rôle des prototypes.

L'avantage de cette approche réside donc dans une classification des axiomes de base (les faits élémentaires), qui permet d'ordonner la théorie du premier ordre. Ceci est à comparer avec l'approche classique en théorie de la preuve où l'ensemble des axiomes (de base ou propriétés générales) est fourni "en vrac".

La notion de prototype proposée ici est donc fondamentalement différente des concepts de vue, de photographie, ou de relation hypothétique proposés auparavant dans le domaine des bases de données [WOO83]. Ceux-ci doivent être en effet définis et manipulés par les usagers, ce qui n'est pas le cas des prototypes définis ici.

Il y a également une autre différence fondamentale entre cette notion de prototype et celle utilisée en Intelligence Artificielle.

Les systèmes de représentation des connaissances, en particulier ceux qui sont basés sur les "frames", y font appel d'une manière ou d'une autre en tant qu'ensemble de valeurs ou de propriétés par défaut [BEN85, ISR84].

Un prototype modélise dans ce cas une connaissance statique.

Tout étudiant est majeur (par défaut) donc est une personne (par défaut), donc possède toutes les propriétés de celle-ci (par défaut).

Au contraire, les prototypes tels qu'ils sont définis ici sont des représentants d'objets qui évoluent sous l'effet des mises à jour de la base de données. A ce titre, ils sont modifiés dynamiquement pour être à tout instant représentatifs d'une certaine classe d'objets.

Comme il a été dit au Chapitre II, la possibilité de définir des exceptions est une particularité que l'on ne retrouve pas dans les modèles de données des SGBD. On n'utilisera pas cette particularité ici.

Ici, la notion de prototype est prise dans son sens littéral. C'est un exemplaire éventuellement inachevé donc évolutif et qui est représentatif, à un instant donné, d'un ensemble d'objets en cours de définition et/ou de construction.

L'utilité pratique de cette approche pour la vérification de contraintes peut être illustrée sur un exemple très simple.

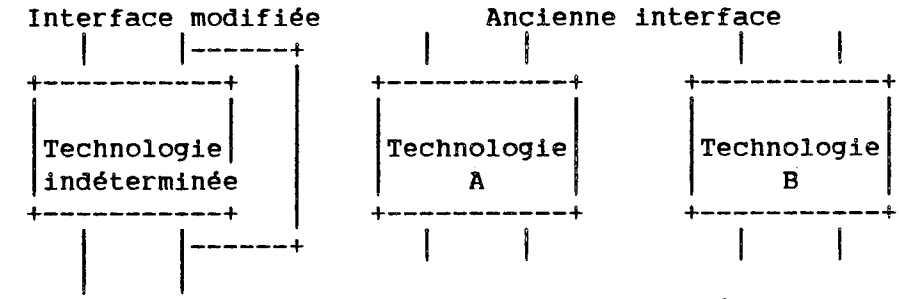
Considérons une base de données pour un système de CAO de circuits VLSI. On sait qu'un même circuit peut être réalisé selon plusieurs technologies différentes. L'interface du circuit constitue un invariant qui permet d'exploiter ces différentes technologies pour minimiser tantôt sa consommation electrique, tantôt sa surface, etc.

La conception de nouveaux circuits réutilise souvent les travaux

de conception de circuits antérieurs. Ceci permet de ne pas repartir de zéro pour chaque nouveau projet.

Si l'on veut par exemple réutiliser le dessin d'un circuit existant qui ne diffère du nouveau que par le nombre ports d' entrée (donc par son interface), on peut tester cette modification en l'appliquant sur un exemplaire particulier de la classe constituée par tous les circuits qui possèdent cette même interface.

Ce représentant, que l'on appellera prototype de la classe, permet de tester immédiatement les modifications envisagées. Si l'on veut par exemple modifier son enveloppe en lui adjoignant une équipotentielle reliant directement une patte d'entrée et une patte de sortie, on peut immédiatement rejeter cette opération sans la tester sur les éléments de la classe.



On conçoit dès à présent sur cet exemple que les prototypes sont bien adaptés à la vérification de contraintes portant sur la structure des objets. Ceci provient également du fait que l'utilisation d'un représentant pour une classe d'objets induit une approximation concernant les valeurs d'attributs. Les contraintes de ce type ne pourront donc être contrôlées que partiellement. On pourra les tester et éventuellement émettre des avertissements lorsque les opérations de mise à jour violeront les contraintes de la classe de l'objet. Ceci est utilisé de manière opérationnelle en temps que garde-fou et c'est particulièrement utile dans les applications du type CAO [RIE86, RIE87].

On utilise dans la suite de ce chapitre la modélisation dont la formalisation en logique est définie au Chapitre IV (paragraphe 4.5).

Reprenons encore l'exemple de la base de données des personnes (Chapitre IV, paragraphe 4.4). Le schéma est constitué des définitions d'objets :

```
(x,i,n,s,a) [ personne(x)  <- ident(x,i)  ∧ nom(x,n) ∧ sexe(x,s)
                              ∧ age(x,a)  ∧ entier(i) ∧ chaîne(n)
                              ∧ (s=F ∨ s=M) ∧ entier(a) ]
      (x) [ homme(x)      <- personne(x) ∧ sexe(x,M) ]
      (x) [ femme(x)      <- personne(x) ∧ sexe(x,F) ]
    (x,p,m) [ enfant(x)   <- personne(x) ∧ homme(p) ∧ pere(x,p)
                                         ∧ femme(m) ∧ mere(x,m) ]
      (x,p) [ homme(p)    <- pere(x,p) ]
    (x,d,a) [ supérieur(d,70) <- homme(x) ∧ age(x,a)
                                  ∧ supérieur(a,50) ∧ poids(x,d) ]
```

où "<-" est l'implication logique et (x) la quantification universelle de la variable x. On suppose ici que pere(x,y), mere(x,y), ident(x,i), age(x,a), entier(i) et chaîne(c) sont définis en extension par des assertions.

Supposons que l'on ait distingué deux enfants x0 et x1, et deux hommes p1 et p2, que l'on appelle prototypes.

Si l'on veut modifier la base de données par les opérations suivantes :

Q1 - le père de x0 initialement inconnu est p1,
Q2 - le père de x1 initialement p2, n'est plus p2,

on va commencer par les appliquer sur les prototypes, ce qui donne

```
enfant(x0) <- personne(x0) ∧ homme(p1) ∧ pere(x0,p1)
enfant(x1) <- personne(x1)
```

On obtient le prototype modifié de enfant(x0) par "connexion" du prototype initial de enfant(x0) avec le prototype de homme(p1).

De même, on obtient le prototype modifié de enfant(x1) par "réduction" de enfant(x1) par pere(x1,p2).

Schématiquement, la réduction permet de modéliser la suppression de propriétés d'un objet et la connexion l'ajout de propriétés à un objet.

Si l'application des opérations sur les prototypes ne viole aucune contrainte d'intrégrité, elles sont certifiées. Dans le cas contraire, elles sont rejetées. La mise à jour de la base de données proprement dite peut ensuite être menée à bien.

L'ensemble des opérations de mise à jour d'une base de données va être modélisé à l'aide de trois opérateurs de produit, réduction et connexion. Dans ce qui suit, toute transaction m est caractérisée par une séquence finie m' = (o1, o2, ..., on) d'opérations de produit, connexion et réduction. Cette séquence m' constitue la signature de m.

On caractérise tout d'abord les propriétés de notre méthode en faisant référence à des concepts élémentaires des mathématiques. Ceci permet de définir une méthode générale de vérification de contraintes sémantiques, basée sur la notion de prototype d'objet et d'échantillon de données. On définit ce que l'on appelle la vérification canonique de contraintes, par analogie avec la décomposition canonique d'une application entre deux ensembles. Ceci fait appel à la notion d'ensemble quotient, définit comme l'ensemble des classes d'équivalence d'un ensemble quelconque par rapport à une relation d'équivalence.

On caractérise également les propriétés de l'échantillon au regard des notions élémentaires de la théorie des ensembles, telles que la structure de groupe, de sous-groupe et d'idéal d'un anneau, pourvus de trois lois appelées ici la "réduction", le "produit" et la "connexion", que l'on définira. Ceci permet la définition précise des règles de contrôle sémantique qui sont nécessaires à la réalisation d'une méthode évoluée de vérification de contraintes dans des applications avancées, telle que la CAO.

L'ensemble des propositions présentées ici a été expérimenté dans le cadre d'une application de conception de circuits intégrés [ADI84, ADI85].

6.1 Classe d'équivalence d'un élément.

On dira qu'une formule f est syntaxiquement dérivable d'un ensemble F de formules (restreintes aux clauses régulières non-récursives) si et seulement si elle appartient à l'ensemble G défini de la façon suivante :

---

1 - initialement G est égal à F,
2 - pour toute formule fi de G, générer itérativement les formules suivantes dont :
    2.1 - la partie gauche ou conclusion est la partie gauche de fi,
    2.2 - les parties droites sont les sous-ensembles (éventuellement vides) des parties droites de fi dont les variables sont reliées aux variables de la partie

gauche par des prédicats impliquant au moins un objet. Si les nouvelles formules ainsi obtenues n'appartiennent pas à G, les ajouter à G.
Dans l'exemple précédent, la variable x est connectée à la variable d à travers l'objet homme(x) et le prédicat poids(x,d). La clause suivante est donc ajoutée à G :

supérieur(d,70) <- homme(x) ∧ poids(x,d),

3 - pour chaque formule générée, les prédicats apparaissant en partie droite sont substitué par les parties droite de règle dans G qui s'unifient avec eux. Les formules résultantes sont ajoutée à G.
Dans l'exemple précédent, la clause suivante est ajoutée :

supérieur(d,70) <- pere(y,x) ∧ poids(x,d),

4 - l'ensemble G ainsi obtenu est à nouveau examiné pour la génération éventuelle de nouvelles formules syntaxiquement dérivables (c-à-d aller en 2). S'il n'y en a pas : fin.

Les clauses étant régulières et en nombre fini, l'algorithme de génération des formules syntaxiquement dérivables d'un ensemble F s'arrête en un temps fini, proportionnel à 2 puissance n, si n est le nombre de formules atomiques de F.

Dans l'exemple précédent, l'ensemble de départ F est le suivant :

```
(x,i,n,s,a) [ personne(x)  <- ident(x,i)  ∧ nom(x,n) ∧ sexe(x,s)
                              ∧ age(x,a)  ∧ entier(i) ∧ chaîne(n)
                              ∧ (s=F V s=M) ∧ entier(a) ]
      (x)   [ homme(x)     <- personne(x) ∧ sexe(x,M) ]
      (x)   [ femme(x)     <- personne(x) ∧ sexe(x,F) ]
  (x,p,m)   [ enfant(x)    <- personne(x) ∧ homme(p) ∧ pere(x,p)
                                          ∧ femme(m) ∧ mere(x,m) ]
    (x,p)   [ homme(p)     <- pere(x,p) ]
  (x,d,a)   [ supérieur(d,70) <- homme(x) ∧ age(x,a)
                                 ∧ supérieur(a,50) ∧ poids(x,d) ]
```

L'ensemble G des formules dérivables est alors :

```
(x)         [ personne(x)  <- 0 ]
(x,i)       [ personne(x)  <- ident(x,i) ]
(x,i,n)     [ personne(x)  <- ident(x,i)  ∧ nom(x,n) ]
(x,i,n,s)   [ personne(x)  <- ident(x,i)  ∧ nom(x,n) ∧ sexe(x,s) ]
(x,i,n,s,a) [ personne(x)  <- ident(x,i)  ∧ nom(x,n) ∧ sexe(x,s)
                              ∧ age(x,a)  ]
(x,i,n,s,a) [ personne(x)  <- ident(x,i)  ∧ nom(x,n) ∧ sexe(x,s)
                              ∧ age(x,a)  ∧ entier(i) ]
........
(x,i,n,s,a) [ personne(x)  <- ident(x,i)  ∧ nom(x,n) ∧ sexe(x,s)
                              ∧ age(x,a)  ∧ entier(i) ∧ chaîne(n)
                              ∧ (s=F V s=M) ∧ entier(a) ]
```

```
   (x) [ homme(x)      <- O ]
   (x) [ homme(x)      <- personne(x) ]
   (x) [ homme(x)      <- sexe(x,M) ]
   (x) [ homme(x)      <- personne(x) ∧ sexe(x,M) ]
 (x,p) [ homme(p)      <- pere(x,p) ]
   (x) [ femme(x)      <- O ]
   (x) [ femme(x)      <- personne(x)  ]
   (x) [ femme(x)      <- sexe(x,F) ]
   (x) [ femme(x)      <- personne(x) ∧ sexe(x,F) ]
   (x) [ enfant(x)     <- personne(x) ]
 (x,p) [ enfant(x)     <- personne(x) ∧ homme(p) ∧ pere(x,p) ]
 (x,m) [ enfant(x)     <- personne(x) ∧ femme(m) ∧ mere(x,m) ]
(x,p,m) [ enfant(x)    <- personne(x) ∧ homme(p) ∧ pere(x,p)
                                      ∧ femme(m) ∧ mere(x,m) ]
 (x,d) [ supérieur(d,70) <- homme(x) ∧ poids(x,d) ]
(x,d,a) [ supérieur(d,70) <- homme(x) ∧ age(x,a) ∧ poids(x,d) ]
(x,d,a) [ supérieur(d,70) <- homme(x) ∧ age(x,a)
                          ∧ supérieur(a,50) ∧ poids(x,d) ]
(x,d,a) [ supérieur(d,70) <- homme(x) ∧ age(x,a)
                          ∧ supérieur(a,50) ∧ poids(x,d) ]
 (x,d) [ supérieur(d,70) <- personne(x) ∧ poids(x,d) ]
 (x,d) [ supérieur(d,70) <- sexe(x,M) ∧ poids(x,d) ]
 (x,d) [ supérieur(d,70) <- personne(x) ∧ sexe(x,M) ∧ poids(x,d) ]
 .........
```

Soit D un ensemble quelconque d'objets, et C un ensemble de contraintes exprimées sous forme de clauses régulières.

A un état particulier de D correspond en théorie une interprétation de l'ensemble C. Dans l'approche classique, les occurrences d'objets constituant D forment un modèle de C, c-à-d que toutes les formules de C sont vérifiées [GAL82].

On adoptera ici une démarche différente. L'ensemble D sera partitionné en sous-ensembles D1, D2, ..., Dn, tels que chaque sous-ensemble Di (1<= i <= n) soit constitué des occurrences d'objets vérifiant seulement une partie de C.

En d'autres termes, l'ensemble C de contraintes est pour chaque état de la base de données scindé en sous-ensembles Cj (non forcément disjoints), tels que Dj soit un modèle de Cj (1<= j <=n). Soit P(C) l'ensemble des parties de C.

> On veut modéliser le fait qu'en général un état E de D n'est pas un modèle de C, car dans les applications réelles, cette situation théorique "idéale" n'est pratiquement jamais vérifiée. C'est évident en CAO par exemple, car la construction d'un objet n'est jamais cohérente vis-à-vis des contraintes qu'il doit vérifier, sauf lorsque sa conception est terminée. On veut par ce biais traiter la notion d'objet incomplet [RIE85], sans faire appel à une théorie particulière impliquant par exemple des valeurs nulles [DEM85].

Soit R la relation définie sur D, telle que deux éléments x et y sont en relation sous R, noté x R y, si et seulement si x et y appartiennent à D et vérifient tous deux la même conjonction de formules C(x). On a :

$$(x) \text{ c } D \text{ , } (y) \text{ c } D \quad x \text{ R } y \iff C(x) = C(y)$$

On sait que la relation R définit sur D une relation d'équivalence [GEL84].

Soit G = (F1, F2, ..., Fg) l'ensemble des formules syntaxiquement dérivables d'un schéma S de base de données.
Si x c D vérifie une conjonction de formules F = F1 ∧ F2 ∧ ... ∧ Fn, où 1<=n<=g) on dira que F définit la classe maximale de x si et seulement si pour toute formule Fj différente de F1, F2, ..., Fn, alors la conjonction F ∧ Fj n'est plus vérifiée par x.

- |F(x)| dénote l'évaluation de la formule F pour un élément x c D, la classe maximale de x est donc définie par la conjonction F1 ∧ F2 ∧ ... ∧ Fn telle que :

$$|F1(x) \wedge F2(x) \wedge \ldots \wedge Fn(x)| = \text{vrai}$$
$$|F1(x) \wedge F2(x) \wedge \ldots \wedge Fn(x) \wedge Fj(x)| \sim= \text{vrai}$$

pour tout Fj différent de F1, F2, ..., Fn.

On appellera <u>classe d'équivalence</u> d'un élément x c D la classe maximale de x.

C'est la classe produit des classes d'équivalence définies par ses propriétés : le père de x0 appartient à la classe produit des classes père, homme et personne.

On dira que F1, F2, ..., Fn est une <u>spécification incomplète</u> d'un objet x si et seulement si c'est un sous-ensemble strict des formules contenues dans les règles de spécification de x et si Fi(x) est satisfaisable pour tout 1<=i<=n.

Ainsi, person(xo) <- O (l'ensemble vide) est une spécification incomplète de x0. C-à-d que l'on sait que x0 est une personne sans avoir aucune autre information sur elle.

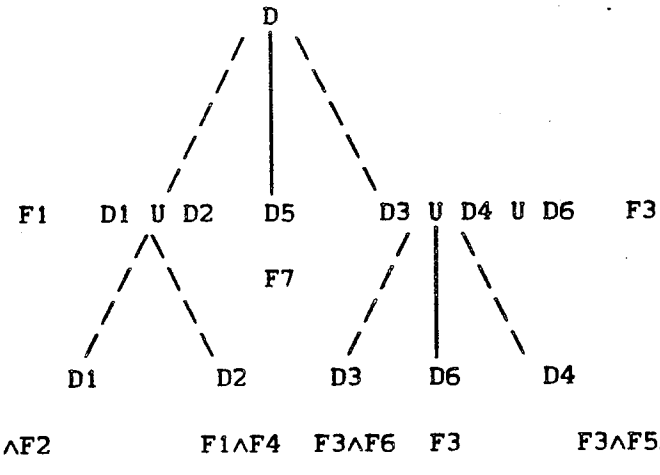
Bien que différentes dans leurs fonctions et dans leur réalisation, un parallèle peut être fait entre les classes d'équivalence et la notion de <u>vue concrète</u> dans le modèle relationnel de données. Elles regroupent en effet des éléments ayant

les mêmes propriétés. En d'autres termes, elles définissent à un instant donné le type de ces éléments (paragraphes 6.3). On définira plus loin trois opérations sur ces classes d'équivalence, appellées réduction, connection et produit (paragraphes 6.6.1 à 6.6.3). Elles correspondent en première approximation aux opérations algébriques de projection d'une vue concrète sur un ensemble d'attributs, de produit et de produit-projection de deux vues concrètes.

Les classes d'équivalence déterminent une partition unique de D, telle que pour tout élément x c D, l'appartenance de x à une classe C(x) peut être déterminée par une recherche arborescente. L'arborescence est construite à la manière d'un questionnaire.

Si l'on ordonne de façon arbitraire les formules, chaque nœud situé sur un chemin de la racine vers une feuille est constitué du sous-ensemble de D qui vérifie une conjonction F1 ∧ F2 ∧ ... ∧ Fp de formules atomiques. Tous les fils d'un même nœud déterminent une partition du sous-ensemble de D représenté par ce nœud. Deux nœuds de même niveau dans l'arborescence, c-à-d situés à égale distance de la racine, représentent donc des sous-ensembles disjoints de D. Les feuilles représentent la partition de D en classes d'équivalence. A un instant donné et pour un ensemble donné de contraintes, cette partition est unique.

Les formules F1, F2, ... F7 déterminent par exemple une partition D1, D2,..., D6 de D :



Lors des modifications apportées aux éléments de la base de données, le parcours de cette arborescence détermine la classe d'équivalence de chaque élément x de D avant et après mise à jour.

L'ajout ou la suppression de propriétés à un élément peut nécessiter une réorganisation de l'arbre. Deux formules Fi et Fj qui déterminaient deux ensembles disjoints Di et Dj peuvent en effet avoir après l'opération un élément commun. Il faut alors

créer un nouveau nœud Fi ∧ Fj dans l'arbre.

Si n est le nombre total des formules dérivables, le nombre maximum de classes d'équivalence est 2 puissance n. A un instant donné, ce nombre est un majorant du nombre réel de classes d'équivalence soit N. L'algorithme de recherche de la classe d'un élément x dans l'arbre est donc dans le pire des cas linéaire et proportionnel à N.

Dans le cas général, la détermination de la classe d'équivalence d'un élément implique l'évaluation de toutes les formules constituant le schéma de la base de données. Ceci permet de caractériser à tout instant sa classe maximale. Cette évaluation exhaustive peut mettre en jeu des définitions récursives de contraintes. L'évaluation de telles définitions est hors du cadre de cette étude [DEL86].

6.2 Relation d'ordre sur les classes d'équivalence.

Soit Fi1, Fi2, ..., Fin les formules dont la conjonction logique définit une partie Ci de C. On a Ci c P(C), où P(C) désigne l'ensemble des parties de C.

On définit une relation d'ordre partiel sur P(C), notée "<", dans laquelle :

(C1) c P(C) et (C2) c P(C)
C1 < C2 <=> (F11, F12, ..., F1n) c (F21, F22, ..., F2m).

6.3 Types et sous-types.

Par abus de langage, on appelera "type" d'un élément x de D l'ensemble C(x) de contraintes que x vérifie et qui est défini par la conjonction de formules :

$$Fi1 \wedge Fi2 \wedge \ldots \wedge Fin$$

On définit alors C(x) par :

$$C(x) <-> Fi1(x) \wedge Fi2(x) \wedge \ldots \wedge Fin(x)$$

A un instant donné, l'ensemble des classes maximales de D définit un modèle "complet" de D au sens de [DEM85], c-à-d que tout élément x de D peut être modélisé sans faire intervenir des valeurs inconnues.

L'ensemble des types des éléments de D définit donc à un instant donné un modèle complet de D.

On appelera "sous-type" Cj du type C(x) d'un élément x de D l'ensemble Cj défini par la conjonction de formules :

$$Fj1 \wedge Fj2 \wedge \ldots \wedge Fjp \quad (p < n)$$

où : (k) (1<= k<=p) (E m) (1<=m<=n) : Fjk = Fim

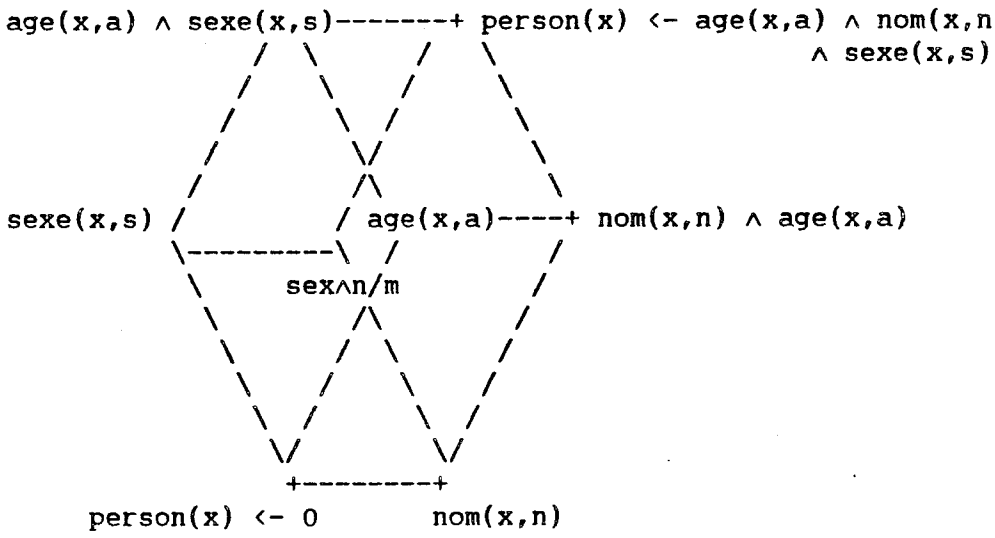
On dira alors qu'un élément y c D est une occurrence d'un sous-type d'un élément x de type C(x) si et seulement si le type C(y) est un sous-type de C(x).

6.4 Types et contraintes : deux approches duales.

Avec les définitions précédentes, il est clair qu'un sous-type Cj d' un élément x de D est défini par la conjonction Fj1, ..., Fjn des formules qui définissent C(x) dans P(C).

La classification des éléments peut alors être vue comme la détermination dans un hypergraphe des sommets associés aux propriétés des objets.

La base de cet hypergraphe est le sommet qui caractérise un objet person(x) par exemple dont on ne connaît aucune propriété. L'opposé de cette base est le sommet qui caractérise un objet entièrement défini. Les voisins immédiats de la base caractérisent chacun une propriété particulière : nom, sexe, age, etc. Les sommets suivants correspondent aux conjonctions des propriétés précédentes prises deux à deux, et ainsi de suite. Chaque sommet est ainsi défini par une conjonction de contraintes prises parmi l'ensemble des formules qui définissent l'objet dans le schéma de la base :



La caractérisation d'un objet revient alors à la détermination de

son sommet dans l'hypergraphe. Celui-ci peut être vu comme un treillis des types où chaque sommet représente un sous-type de l'objet complet.

L'hypergraphe est alors un graphe de spécialisation et de généralisation des types d'objets.

6.5 Propriétés de l'ensemble quotient.

On appelera échantillon E de D l'ensemble des représentants des classes d'équivalence des éléments de D.

Soit S l'ensemble quotient D/R. On va définir sur S trois opérations, dites réduction, connexion et produit. Elles permettent de caractériser les opérations effectuées sur la base de données. A toute transaction m on associe la séquence finie m' = (o1, o2, ... on) d'opérations de réduction, connexion et produit qui la caractérisent dans D/R. Par hypothèse, m et m' sont isomorphes, c-à-d que leurs effets sont identiques quant aux formules satisfaites par les objets après mise à jour. Si s est l'application qui associe à un élément sa classe d'équivalence on aura donc pour tout x et x' :

$$x\ R\ x' \Rightarrow s(m(x)) = s(m'(x)) = s(m(x')) = s(m'(x')).$$

On appellera m' la signature de m.

On définit tout d'abord l'échantillon en termes de structures élémentaires de la théorie des ensembles. On applique ensuite les notions de réduction et de connexion aux représentants des classes d'équivalence. Enfin on définit une méthode de vérification de contraintes basée sur ces concepts.

6.5.1 Opération de réduction.

Soient s1 et s2 deux éléments quelconques de S. On dira que s1 est réduit par s2 si et seulement si :

- s2 < s1, et
- les propriétés de s1 ne vérifient plus les formules qui définissent s2.

"<" est la relation d'ordre définie sur les classes d'équivalence (Figure 6.2). On note cette opération : s1-s2 . Le résultat de s1-s2 est la classe de S dont les éléments vérifient la conjonction des formules définissant s1 à l'exception des formules qui définissent s2.

Cette opération est associative, car tout élément s3 < s2,

où s2 < s1, est tel que :

$$s1-(s2-s3) = (s1-s2)-s3$$

Cette opération a un élément neutre, appelé élément vide, noté 0, tel que 0 < s1 donne s1 :

$$s1-0 = s1$$

Enfin tout élément s a un inverse noté s', tel que :

$$s-s' = 0$$

Cette opération n'est pas commutative.

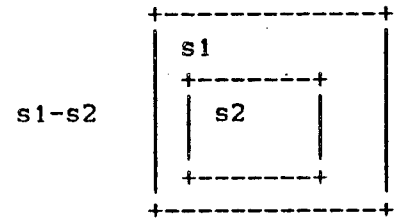


Figure 6.2. Réduction de s1 par s2 : s1-s2.

L'ensemble quotient D/R, muni de l'opération ci-dessus a donc une structure de groupe non commutatif.

La réduction permet de caractériser la différence de deux classes d'équivalence de l'ensemble quotient. Elle peut être assimilée à l'opération algébrique de projection d'une vue concrète sur un ensemble de constituants dans le modèle relationnel de données.

6.5.2 Opération de connexion.

Soit "+" l'opération de connexion de deux éléments s1 et s2 de S (Figure 6.3), définie ainsi :

$$(s1+s2)+s3 = s1+(s2+s3)$$

Le résultat de la connexion de s1 et s2 est la classe dont les éléments vérifient à la fois les formules définissant s1 et les formules définissant s2.

Elle est dotée d'un élement neutre, noté 0', tel que :

$$s+0' = 0'+s = s$$

Cette opération est commutative :

$$s1+s2 = s2+s1$$

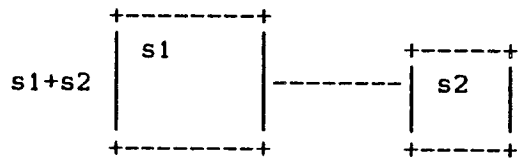


Figure 6.3. Connexion de s1 et s2 : s1+s2.

Tout élément a un opposé défini par la négation des formules qui définissent sa classe. De plus, la réduction est distributive par rapport à la connexion (Figure 6.4) :

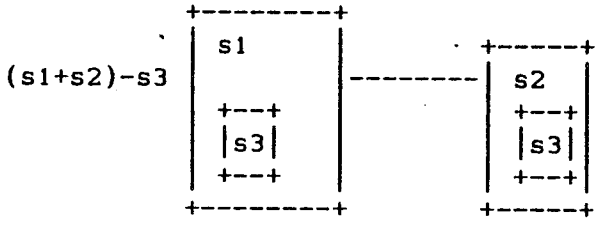


Figure 6.4. Distributivité de la connexion.

$$(s1+s2)-s3 = (s1-s3) + (s2-s3)$$

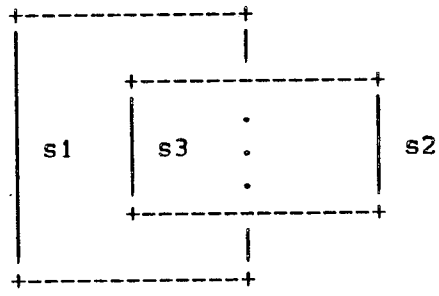
L'ensemble D/R muni des deux opérations définies ci-dessus a donc une structure d'anneau non commutatif. Ce n'est donc pas un corps. De plus, il est stable pour la réduction et la connexion. En ce sens que :

$$(s1) \subset S \ , \ (s2) \subset S : s1-s2 \subset S$$
$$(s1) \subset S \ , \ (s2) \subset S : s1+s2 \subset S$$

La connexion permet de caractériser la fusion de deux classes d'équivalence de l'ensemble quotient. Elle peut être assimilée à l' opération algébrique de produit de deux vues concrètes dans le modèle relationnel de données.

6.5.3 Opération de produit.

Soient s1 et s2 deux éléments de S dont l'intersection est non vide, c-à-d définis par des ensembles non disjoints de formules. Le produit de s1 par s2, noté s1*s2, caractérise un élément s3 de S défini par la conjonction des formules communes à s1 et s2.



Cette opération a un élément nul, l'ensemble vide O, tel que (s) c S s*O=O.

Elle est associative et distributive par rapport à la connexion et à la réduction :

(s1,s2,s3) c S s1*(s2+s3) = s1*s2 + s1*s3
(s1,s2,s3) c S s1*(s2-s3) = s1*s2 - s1*s3.

Cas particulier :

Lorsque s2 est contenu dans s1, c-à-d s2 < s1, on a : s1*s2 = (s1-s2)' où s' est le complément de s dans S.

Le produit de deux élément de S correspond à l'opération de produit-projection de deux vues dans le modèle relationnel de données.

6.5.4 Extension.

On dira par analogie avec la terminologie usuelle concernant la notion d'idéal d'un ensemble que toute partie S' de D/R constitue un "idéal" de l'échantillon.

Intuitivement, ceci signifie que pour tout élément s1 et s2 de l'échantillon, les éléments s1+s2 et s1-s2 appartient encore à l'échantillon.

Ces définitions vont permettre de caractériser une méthode de vérification de contraintes dite "canonique".

6.6 Validité d'une opération.

On sait que pour toute application f d'un ensemble X dans un ensemble Y, il existe une décomposition unique, dite "canonique", de f, telle que pour la relation d'équivalence R définie par x R x' ssi f(x)=f(x'), on ait :

(x) c X et (y) c Y tels que f(x)=y : f(x)=i(h(s(x)))=y,

où i est la relation identité de f(X) dans Y, s l'application qui a tout x c X fait correspondre sa classe d'équivalence et h une application unique de X/R dans f(X) (Figure 6.1). La composition des applications est notée par parenthésage : i(h(...)).

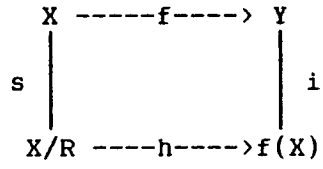


Figure 6.1. Décomposition canonique d'une application f : X -> Y.

On rappelle que l'ensemble quotient X/R est l'ensemble des classes d'équivalence de X pour la relation R.

Soit s l'application qui à tout élément x c D, ensemble des objets explicites, implicites ou hybrides constituant un état de la base de données, fait correspondre sa classe d'équivalence. Soit E l'ensemble des représentants des classes d'équivalence. Soit f l'application de D dans E, qui fait correspondre à un objet x c D le représentant x0 de sa classe d'équivalence. D'après ce qui précède, il existe une décomposition unique de f : D -> E, telle que (Figure 6.5) :

$$(x)\ c\ D\ x0=f(x) \Rightarrow x0=i(h(s(x)))$$

Selon la relation d'équivalence R, x c D et y c D sont équivalents si et seulement s'ils vérifient le même ensemble de contraintes :

$$x\ R\ y \Leftrightarrow f(x)=f(y)$$

Si les contraintes de C sont définies à l'aide de clauses régulières, x et y sont équivalents si et seulement s'ils vérifient la même conjonction de formules.

Vérifier si x c D vérifie une conjonction de formules F équivaut à chercher si s(x) est défini sur le même ensemble de contraintes F par application de i(h(s(x))). On sait que ceci ne dépend pas du représentant choisi pour la classe d'équivalence s(x).

On choisit pour chaque classe d'équivalence un représentant particulier.

On appelle <u>prototype</u> d'un élément x de D le représentant de la classe d'équivalence de x dans D/R.

Ces prototypes sont utilisés comme gardes-fous contre d'éventuelles violations de règles définies dans le schéma de la base de données.

Pour cela, on définit une méthode de certification qui garantit qu'une opération de mise à jour m sur un objet est valide si la signature m' de m est valide (aux constantes près) sur le prototype de l'objet.

La notion de validité d'une opération est définie comme suit.

Si l'on considère un état de la base de données comme une interprétation I d'un sous-ensemble F = (f1, f2, ..., fp) de G, les formules syntaxiquement dérivables du schéma (paragraphe 6.1), à un instant donné, I est un modèle de F :

$$I \models f1, f2, \ldots, fp$$

On dit qu'une opération m sur la base de donnée est valide si et seulement si elle transforme I en une interprétation I' telle que

$$I' \models f1, f2, \ldots, fp, fp+1.$$

où fp+1 peut être éventuellement vide.

Par extension, on dira qu'une opération m sur un objet est valide si et seulement si l'ensemble des contraintes satisfaites par l'objet après l'opération ne décroît pas. Formellement, m est valide si et seulement si $s(x) <= s(m(x))$, où s est l'application qui fait correspondre à un objet x sa classe d'équivalence.

Le symbole "<=" dénote la relation d'ordre entre classes étendue à l'égalité de deux classes. En d'autres termes, si les contraintes de la classe C sont exprimées à l'aide de formules du calcul des prédicats du premier ordre et restreintes aux clauses régulières, elles forment avec les définitions de données une théorie du premier ordre. La règle précédente impose de façon opérationnelle que la logique dans laquelle on se place soit monotone.

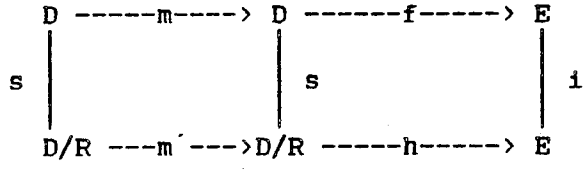


Figure 6.5. Vérification canonique de contraintes.

Si l'application m modélise les opérations de mise à jour d'un schéma ou des éléments d'une base de données, si m' est la signature de m en termes de réduction de produit et de connexion, et si h modélise la bijection qui associe à une classe d'équivalence son prototype, alors l'application h(m'(s(x))) représente la certification de ces opérations sur les prototypes.

Soit m une opération de mise à jour sur un objet x et m' la signature de m. Soit s(x) la classe d'équivalence de x et x0 le représentant de cette classe. Les définitions précédentes de signature d'une transaction et de certification permettent de démontrer les théorèmes suivants.

Théorème 1.

Si m' est une opération de réduction, alors m(x) n'est pas valide.

Démonstration.
Soit x' l'élément par lequel on réduit x. Par définition de la réduction, x'<x. A une permutation près, soit f1, f2, ..., fm les formules satisfaites par x et f1, f2, ..., fn les formules satisfaites par x', avec n<m. Après l'application de m', x0 ne vérifie plus que les formules fn+1, fn+2, ..., fm. Par isomorphisme de m et m' et par définition de la validité, l'opération m n'est donc pas valide.

Théorème 2.

Si m' est une opération de produit, alors m(x) n'est pas valide.

Démonstration.
Suivant la même démarche, soit x' le deuxième élément du produit x*x'. A une permutation près, x satisfait les contraintes f1, f2, ..., fp, fp+1, ..., fm et x' satisfait les formules fp+1, fp+2, ..., fm, fm+1, que les formules fp+1, fp+2, ..., fm. Par isomorphisme de m et m' et par définition de la validité, l' opération m n'est donc pas valide.

Théorème 3.

Dans le cas général, une opération modifiant la valeur d'une propriété d'objet n'est pas valide.

Démonstration.
En effet, une modification de valeur peut être caractérisée par une séquence de réduction et de connexion sur x0. Dans

le cas général, ceci implique que m(x) n' est pas valide : d'une part, la réduction est par le théorème 1 non valide, d'autre part rien ne garantit que la nouvelle valeur de la propriété vérifie bien les contraintes de la classe de l'objet.

Remarque.
Cette restriction est la plus importante de la méthode. Elle ne fonctionne correctement qu'aux constantes près. Si une propriété est contrainte par exemple à des valeurs prises dans un intervalle numérique, rien ne garantit que si l'incrémentation de cette valeur sur le prototype est valide, elle le sera aussi sur l'objet.

Théorème 4.

Si m' est une opération de connexion et si m'(x0) est valide alors m(x) est valide.

Démonstration.
D'après les théorèmes 1, 2 et 3, la seule opération susceptible d'être valide est la connexion. Soit f1, f2, ..., fp les formules satisfaites par x et fi, fi+1, ..., fk les formules satisfaites par x'. Soit x0 et x0' les prototypes de x et x'. Par définition, la connexion de x et x' résulte en un élément qui vérifie à la fois les formules caractérisant x et les formules qui caractérisent x'. La connexion de x0 et de x0' vérifie donc la conjonction de f1, f2, caractérise une classe qui contient les classes s(x0) et s(x0'). Par définition des prototypes, on a s(x0)=s(x) et s(x0')=s(x'). Par définition de la relation d'ordre "<", on a s(x) < s(x+x') et s(x') < s(x+x'). Par isomorphisme de m et de m' et par définition de la validité, il en résulte que m est valide.

Théorème 5.

Une séquence finie d'opérations valides est valide.

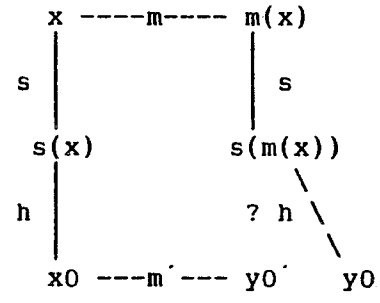
Démonstration.
Raisonnons par récurrence sur le nombre d'opérations définissant la signature de m. Soit o1, o2, ..., ok la séquence d'opérations définissant la signature m' de m. Soit x l'objet à modifier. Soit f1, f2, ..., fp les formules définissant sa classe d'équvalence. L'application de o1 à x étant valide, la classe d'équivalence du résultat o1(x) est définie par la conjonction des formules f1, f2, ..., fp, fp+1, où fp+1 est éventuellement vide. On a donc s(x) <= s(o1(x)).
Soit o1, o2, ..., om une séquence finie d'opérations valides (m<k). Par hypothèse de récurrence, cette séquence est valide. Soit om+1 l'opération suivante. Etant valide, elle

fait passer l'objet en cours de mise à jour d'une classe définie par les formules f1, f2, ..., fm à une classe f1, f2, ..., fm, fm+1 (fm+1 pouvant être vide). On a donc : s(o1(o2(...(om(x)))) <= s(o1(o2(...(om+1(x))))).

Théorème 6 : théorème de validité.

Quelque soit m, si m'(x0) est valide alors m(x) est valide.

Démonstration.
Soit y le résultat de l'opération m(x) : y = m(x). Soit y0 le prototype de m(x) : y0 = h(s(m(x))). Soit y0' le résultat de m'(x0). Pour que la proposition soit démontrée, il faut et il suffit que les classes de y0 et y0' soit identiques : s(y0) = s(y0').



On a : s(m'(x0)) = s(m(x0)) = s(m(x)) par définition de m'. Donc : s(y0') = s(m'(x0)) = s(m(x)) = s(y). Comme s(y) = s(y0) par définition des prototypes, on a bien : s(y0) = s(y0').

Compte tenu du théorème 5, on arrive donc au résultat général suivant.

Pour toute opération de mise à jour m que l'on peut caractériser par sa signature m', la validité de m' sur le prototype x0 d'un objet x entraîne la validité de m sur l'objet x.

6.7 Prototypes.

Le représentant d'une classe d'équivalence peut être fourni par l'utilisateur. Cela nécessite toutefois une interaction entre le système de vérification de contraintes et l'usager, en obligeant celui-ci à donner explicitement des valeurs aux caractéristiques des prototypes, ce qui peut être redondant avec les modifications qu'il effectue sur les objets qu'il manipule.

On préfère donc générer automatiquement ces prototypes, en les faisant évoluer dynamiquement et conformément aux mises à jour opérées sur la base de données. La vérification des contraintes étant indépendantes des prototypes choisis pour chaque classe d'équivalence, ceux-ci peuvent être déterminés par le seul système de vérification, sans intervention extérieure.

De plus, un prototype modélise ici complètement un objet et les liens qu'il peut avoir avec d'autres. Son rôle est d'être représentatif. Alors que dans les systèmes à base de "frame" par exemple, un prototype ne peut représenter qu'un objet primitif (un circuit électronique ne peut pas y être représenté comme la juxtaposition d'un circuit logique, d'un circuit électrique et d'un circuit implanté) [ISR84].

6.7.1 Connexion de prototypes.

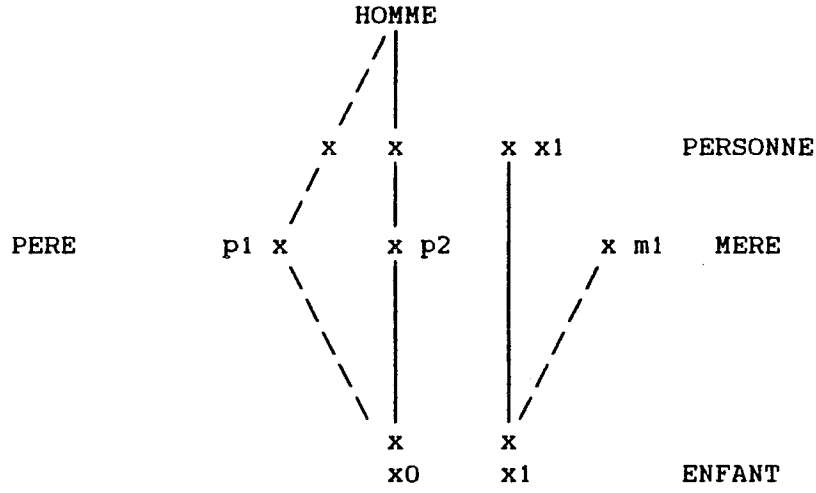
Les deux opérations de réduction et de connexion définies sur l'ensemble quotient D/R vont permettre de prendre en compte les mises à jour de prototypes.

Lors de la mise à jour des objets, on contrôle la validité des opérations en les appliquant tout d'abord sur leur prototype (paragraphe 6.9). Ceci implique une modification de leurs propriétés, et dans le cas général, un changement de classe d'équivalence. Cette "migration" doit être prise en compte de manière formelle. C'est le but de ces deux opérations.

Supposons que l'échantillon correspondant au schéma de la base de données des personnes contienne les prototypes suivants :

```
enfant(x0)    <- personne(x0)                          (1)
enfant(x1)    <- personne(x1) ∧ mere(x1,m1)            (2)
personne(x0) <- ident(x0,i0) ∧ sexe(x0,M)              (3)
personne(x1) <- ident(x1,i1) ∧ sexe(x1,F)              (4)
homme(p1)     <- personne(p1) ∧ sexe(p1,M)             (5)
homme(p2)     <- personne(p2) ∧ sexe(p2,M)             (6)
personne(p2) <- nom(p2,n2)                             (7)
```

Ce qui signifie que l'on connaît l'existence d'un enfant x0 dont on ne connaît que l'identité i0 et le sexe masculin (1)+(3). On connaît également un enfant x1, d'identité i1, de sexe féminin (2)+(4), qui a une mère m1 dont on ne sait rien (2). On connaît par ailleurs l'existence d'un homme p1 dont on ne sait rien (5), et d'un homme p2 dont on ne connaît que le nom n2 (6)+(7). Schématiquement, on a les prototypes liés comme suit :



Si l'on veut modifier la base de données par les opérations suivantes :

Q1 - le père de x0 est p1,
Q2 - le père de x1 est p2,

on va commencer par modifier l'échantillon E, c-à-d modifier les prototypes, ce qui donne :

enfant(x0) <- personne(x0) ∧ homme(p1) ∧ pere(x0,p1) (8)
enfant(x1) <- personne(x1) ∧ homme(p2) ∧ pere(x1,p2) (9)

On obtient le prototype modifié de enfant(x0) par "connexion" du prototype initial de enfant(x0) avec le prototype de homme(p1), soit par abus d'écriture x0 + p1, augmenté de la nouvelle propriété pere(x0,p1).

De même, on obtient le nouveau prototype de enfant(x1) par connexion de enfant(x1) et homme(p2), augmentée de pere(x1,p2).

Formellement, la connexion p1 + p2 de deux prototypes est donc la conjonction logique des formules F1 et F2 qui les définissent et des formules F qui modifient leurs propriétés.

$$p1 + p2 = F1 \wedge F2 \wedge F$$

6.7.2 Réduction de prototypes.

Supposons que p1 soit un prototype de la classe d'équivalence s1 définie par la conjonction de formules F1 ∧ F2 ∧ F3, et que p2 soit un prototype de la classe d'équivalence s2 définie par la conjonction F1 ∧ F2, c-à-d s2<s1.

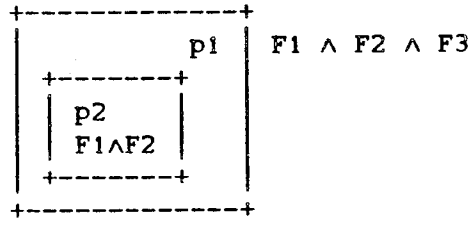


Figure 6.2. Réduction de s1 et s2 : s1*s2.

Si l'on veut modifier un objet x1 de prototype p1 en supprimant la propriété correspondant à F3. Comme s2<s1, on peut appliquer ici l'opération p2*p1 qui produit un prototype vérifiant F1 ∧ F2.

Supposons maintenant que l'on veuille modifier enfant(x0) en lui affectant un nouveau pere, soit p2 :

enfant(x0) <- personne(x0) ∧ homme(p2) ∧ pere(x0,p2)

Comme les prototypes de p1 et p2 existent, il faut et il suffit de

- supprimer la connexion de x0 à p1 par réduction,
- ajouter celle de x0 à p2 par connexion.

D'après ce qui précède, la correspondance entre les opérations d'insertion, suppression et mise à jour dans les bases de données et les opérations de connexion et de réduction de prototypes est la suivante :

bases de données	prototypes
insertion	connexion
suppression	réduction ou produit
mise à jour	réduction ou produit + connexion

6.8 Certification des opérations.

Le terme certification est employé ici par analogie avec les procédures de contrôle utilisées dans l'aéronautique civile lors de la mise sur le marché d'un nouvel appareil de transport. A la sortie des chaînes de montage des premiers prototypes de pré-série, et avant toute mise en service commercial, le constructeur d'un appareil doit obtenir l'agrément des autorités pour que l'appareil soit reconnu conforme à la réglementation en vigueur, et obtienne ainsi son certificat de navigabilité. L'appareil est alors "certifié".

Ce terme est donc sans rapport avec la notion de certification utilisée pour les algorithmes de contrôle d'accès concurrents qualifiés d' "optimistes" [BOK84]. Le terme "confirmation" leur serait sans doute plus approprié. La validité des opérations en conflit y est contrôlée après exécution des transactions, ce qui est à l'opposé de notre approche. Ici, on examine les opérations de mise à jour sur les données en contrôlant à priori leur validité sur des prototypes représentatifs.
Par opposition, la certification est donc ici une méthode pessimiste de vérification de contraintes.

Partant de cette notion de prototype, il est possible d' implanter une méthode de vérification de contraintes basée sur la décomposition canonique décrite plus haut (paragraphe 6.7).

Les avantages de cette approche sont multiples. D'une part elle permet d'effectuer des contrôles de cohérence sémantiques sur les objets constituant la base de données en se limitant à des contrôles sur un représentant et un seul de chaque classe d'équivalence.

D'autre part, elle permet d'implanter efficacement deux parmi les besoins des applications que l'on a qualifiées d' "avancées".

Premièrement, la vérification différée de contraintes [MOR83]. D'autre part, par analogie avec les méthodes de validation de protocoles à deux phases utilisées pour la gestion de données réparties, de scinder la vérification de contraintes en deux étapes successives. Ce sont la certification et la validation proprement dite (Figure 6.6).

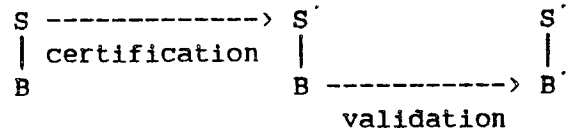


Figure 6.6. Certification et validation.

Dans la figure 6.6, S et S' sont les échantillons de la base B avant et après une opération de mise à jour émise par un usager. B et B' sont les états de la base avant et après modification effective des données.

La certification des opérations a pour objectif de valider les opérations d'un utilisateur, c-à-d de contrôler leur validité, sur les prototypes formant l'échantillon, et non pas sur l'intégralité du contenu de la base de données qu'il manipule.

Cet avantage n'est pas mince lorsque l'on se place comme ici dans le cadre de la logique pour modéliser les informations. On sait en effet que l'ensemble des axiomes d'une théorie du premier ordre comporte (Chapitre II, paragraphe 2.3.1) :

- l'axiome de fermeture du domaine,
- l'axiome d'unicité des noms,
- l'axiome de substitution des termes égaux,
- et les axiomes de complétude qui associent à chaque information les ensembles de valeurs de ses propriétés.

Si l'on considère l'intégralité d'une base de données pour contrôler les opérations de mise à jour, on devra tester chaque opération à partir de l'ensemble des axiomes qui définissent la théorie. Si ni est le nombre de types distincts de la base, pi le nombre d'instances du type i, et ci le nombre d'axiomes qui définissent un type i, le coût de cette vérification est proportionnel à Somme(ni*pi*ci).

Si l'on certifie les mises à jour à partir des seuls prototypes, le coût de l'opération est proportionnel à Somme(ni*ci), car on n'a besoin que d'un prototype par classe d'équivalence. Il faut mentionner de plus que Somme(ni*ci) est un majorant, puisque les prototypes ne sont créés que pour être représentatifs d'objets existant (cf. paragraphes suivants).

La validation, quant à elle, a pour objectif de rendre effectives dans la base de données les opérations qui auront été certifiées.

6.8.1 Définition d'objet.

Lorsqu'une nouvelle définition d'objet D est créée dans le schéma d'une base, on contrôle si elle est correcte vis-à-vis du modèle de données et vis-à-vis des règles existantes du schéma. Ceci ne concerne pas les objets indépendants ou explicites qui, par définition, n'impliquent pas d'autre objet qu'eux-mêmes (Chapitre IV : modélisation).

Initialement, c-à-d lorsque que le schéma S de la base de données est créé, l'échantillon E est vide.

Par recopie, un prototype est créé lors de l'insertion du premier objet correspondant à la définition D.

Il évolue par la suite en fonction des modifications apportées aux objets de la classe d'équivalence.

Soit E l'échantillon et S le schéma de la base de données. On rappelle que E et S sont implantés sous formes de clauses régulières et constituent les axiomes d'une théorie du premier ordre $T = E \wedge S$. Les formules de S qui définissent un objet contiennent des formules atomiques, des termes de comparaison et éventuellement des termes qui font intervenir d'autres objets (cas des objets implicites et hybrides).

Soit D le littéral positif définissant un objet (exemple : D = personne(x)). Il est explicité par une règle de spécification (exemple : personne(x) <- nom(x,n) $\wedge$ age(x,a) $\wedge$ sexe(x,s)).

La procédure qui est appliquée pour contrôler la cohérence d'une nouvelle définition d'objet D vis-à-vis du schéma est :

$$S \wedge E \vdash D$$

c-à-d que D doit être un théorème de la théorie T.

Si la déduction réussit, cela signifie que toutes les propriétés du nouvel objet sont déjà légalement définies dans le schéma. La structure de l'objet est donc licite. La définition D est ajoutée au schéma S.

Cette définition pourra être par la suite augmentée ou réduite de la même façon. C'est ce mécanisme qui permet d'une part de prendre en compte la structure dynamique des objets, d'autre part de tenir compte de leur incomplétude.

S'il n'y a pas de déduction qui permette d'obtenir D la conjonction désirée de propriétés à partir du schéma et de l'échantillon, la définition n'est pas certifiée. Elle n'est pas ajoutée au schéma.

Exemple :

S : personne(x) <- nom(x,n) ∧ age(x,a) ∧ sexe(x,s)
sexe(x,s) <- s=M
sexe(x,s) <- s=F

E : personne(a).

D : homme(x)

Avec la règle de spécification suivante : homme(x) <- personne(x) ∧ sexe(x,M), D est déductible de S, donc à fortiori de S ∧ E. D est donc certifiée.

Par contre avec homme(x) <- personne(x) ∧ sexe(x,M) ∧ adresse(x,a), D n'est plus déductible de S car adresse(x,a) n'est pas définie. Cette nouvelle définition n'est pas certifiée.

Cas particulier :

Les doubles définitions à priori contradictoires ne peuvent pas être contrôlées automatiquement car le mécanisme de généralisation des types permet à un objet d'avoir plusieurs sous-types complémentaires. Exemple : individu(x) avec pour sous-types les personnes mineures et majeures :

individu(x) <- personne(x) ∧ age(x,a) ∧ a < 18,
individu(x) <- personne(x) ∧ age(x,a) ∧ a >= 18.

Une méta-règle est donc nécessaire pour détecter ces doublons et en avertir l'utilisateur.

Soit D1 l'ensemble des définitions d'objets explicites. Soit D2 l'ensemble des définitions d'objets implicites ou hybrides. Soit S = D1 ∧ D2. Initialement, l'échantillon ne contient que des prototypes pour les éléments de D1. Lorsqu'une nouvelle définition D est introduite, trois cas sont à considérer :

- D ne fait référence à aucune définition de S,
- D fait référence à des définitions de D1 (resp. D2),
- D fait référence à des définitions de D1 et D2.

1er cas :

La définition de D est celle d'un objet explicite, ou indépendant. La définition est ajoutée telle quelle au schéma.

2e cas :

Ce cas correspond à la définition d'un objet implicite. Si S ∧ E |- D alors D1 devient D1 U D. Sinon D est rejetée.

3e cas :

D est une définition d'objet hybride. De la même façon, si $S \wedge E \vdash D$ alors D2 devient D2 U D. Sinon D est rejetée.

6.8.2 Création d'objet.

Lors de la création d'une occurrence d'un objet x, sa classe d'équivalence C(x) est examinée. Si elle est vide, une copie de l'objet est utilisée comme prototype, soit P.

Exemple :

Si E=O (l'ensemble vide), la création d'une personne p implique la génération dans E de person(p) et de ses propriétés connues :

S : personne(x) <- nom(x,n) ∧ age(x,a) ∧ sexe(x,s)
sexe(x,s) <- s=M
sexe(x,s) <- s=F

E : personne(p).

Si elle n'est pas vide, le prototype de la classe existe. Il n'y a pas lieu d'en changer. La création est certifiée par :

$$S \wedge E \vdash R(p)$$

où R(p) est l'instance de la règle de spécification concernant l'objet créé p.

Cette création peut ne pas être certifiée parce que le prototype existant a moins de propriétés que l'objet créé. Si elle est maintenue par l'utilisateur, il faut effectuer la connexion de p avec le prototype de sa classe.

Exemple :

Création de l'objet : person(p') <- nom(p',n) ∧ age(p',a) avec :

S : personne(x) <- nom(x,n) ∧ age(x,a) ∧ sexe(x,s)
sexe(x,s) <- s=M
sexe(x,s) <- s=F

E : personne(p).

La création de p' n'est pas certifiée car il n'y a pas de déduction de S ∧ E qui donne person(p') <- nom(p',n) ∧ age(p',a).

Si la création est néanmoins voulue en passant outre la certifi-

cation, il faut connecter p et p' dans E par p + p' ce qui donne

E : person(p') <- nom(p',n) ∧ age(p',a).

Si x est un objet implicite ou hybride pour lequel il existe déjà un prototype P, il est utilisé pour tester l'insertion par : S ∧ E |- R(x).

Exemple :

Si l'on veut créer une occurrence de l'objet individu(x0) défini par :

individu(x0) <- personne(x0) ∧ age(x0,a0).

avec :

S : personne(x) <- nom(x,n) ∧ age(x,a) ∧ sexe(x,s)
sexe(x,s) <- s=M
sexe(x,s) <- s=F
individu(x) <- personne(x) ∧ age(x,a) ∧ a >= 18.

E : personne(p).
individu(p).

la création est certifiée si a0 >= 18 car il existe une déduction de S qui permet d'obtenir individu(x0) si cette condition est satisfaite.

L'algorithme de gestion de l'échantillon est donné dans le paragraphe qui suit.

6.8.3 Modification d'objet.

Les contraintes sont utilisées pour certifier les opérations de mise à jour. En particulier, les effets de bord sont évalués pour éviter ultérieurement la propagation d'erreurs. La décision d'autoriser ou non ces propagations est du ressort de l'utilisateur.

Pour certifier les mises à jour, on évalue les contraintes associées à la définition d'un objet, quelque soit sa nature (explicite, implicite ou hybride), sur le prototype de sa classe d'équivalence dans D/R. La violation de l'une quelconque des contraintes entraîne le rejet de l'opération.

> Le principe appliqué est de rejeter toute modification qui fait passer le prototype x0 d'une classe d'équivalence C dans D/R à une classe C', telle que C' < C (la relation d'ordre est celle définie au paragraphe 6.2).

C'est-à-dire que l'on rejette toute modification sur un objet si l'ensemble des contraintes qu'il vérifie après modification est "plus petit" que l'ensemble des contraintes qu'il vérifiait initialement.

Toutes les opérations de modification doivent donc être valides (paragraphe 6.6).

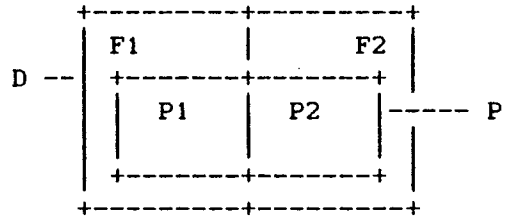
Soit D la définition d'un objet à mettre à jour. Soit F1 la conjonction de formules de la règle de spécification de D contenant des termes ti à modifier et ai leurs nouvelles valeurs ($1 \leq i \leq n$). Soit F2 la conjonction de formules n'en contenant pas. On a :

$$D = F1 \wedge F2.$$

Soit P le prototype de l'objet à modifier. On pose de même :

$$P = P1 \wedge P2$$

où P1 est la conjonction des formules de P contenant des termes à modifier et P2 la conjonction de formules de P n'en contenant pas.



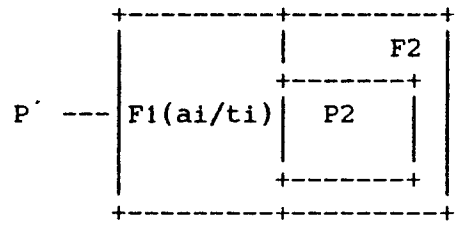
Soit (ai/ti) l'ensemble des substitutions des termes ti par les termes ai. Soit P' le prototype modifié. On a :

$$P' = F1(ai/ti) \wedge P2,$$

Enfin soit :

$$E' = (E - P) \cup (F1(xi/ti) \wedge P2)$$

où xi est une variable non instantiée.



La procédure de certification est : $S \wedge E' \vdash P'$.

Si l'opération est certifiée, le prototype P est remplacé dans

l'échantillon E par le prototype modifié P', ce qui produit un nouvel échantillon E qui est utilisé pour la certification de l'opération suivante.

La construction de l'ensemble E' a pour but de s'assurer que l'échantillon initial E est modifiable. En effet, pour montrer que S ∧ E |- P est un théorème vis-à-vis de la théorie T, E ne doit pas contenir de formule dans laquelle un terme ti à modifier du prototype est déjà instantié. La substitution de E' à E a pour but de permettre cette preuve, sans rien enlever à la généralité de la démarche. P' est le prototype modifié de la classe d'équivalence de l'objet mis à jour.

Exemple :

Reprenons l'exemple donné en introduction. Le schéma est constitué des définitions d'objets :

```
(x,i,n,s,a) [ personne(x)  <- ident(x,i)  ∧ mnom(x,n) ∧ sexe(x,s)
                              ∧ age(x,a)  ∧ entier(i) ∧ chaîne(n)
                              ∧ (s=F ∨ s=M) ∧ entier(a) ]
      (x) [ homme(x)       <- personne(x) ∧ sexe(x,M) ]
      (x) [ femme(x)       <- personne(x) ∧ sexe(x,F) ]
  (x,p,m) [ enfant(x)      <- personne(x) ∧ homme(p) ∧ pere(x,p)
                                          ∧ femme(m) ∧ mere(x,m) ]
```

Supposons que l'échantillon E contienne seulement un prototype P pour l'objet "personne", soit :

```
personne (x0) <- nom(x0,n0) ∧ age(x0,t1) ∧ chaîne(n0) ∧
                 entier(t1)
```

Si l'on veut mettre à jour l'âge t1 d'une personne x, on va tenter de certifier cette opération en l'appliquant d'abord au prototype x0 de l'objet "personne" : S ∧ E' |- D'.

```
Soit ici : P1        = [ age(x0,t1) ∧ entier (t1) ]
           P2        = [ nom(x0,n0) ∧ chaîne(n0) ]
           F1(x1/t1) = [ age(x0,x1) ∧ entier (x1) ]
           E'        = (E - P) U (F1(x1/t1) ∧ P2)
           E'        = [ nom(x0,n0) ∧ age(x0,x1) ∧ chaîne(no)
                                                  ∧ entier(x1) ]
```

Si a1 est la nouvelle valeur de age, on a :

```
           P'        = F1(a1/t1) ∧ P2
           P'        = [ personne (x0) <- nom(x0,n0) ∧ age(x0,a1)
                                        ∧ chaîne(n0) ∧ entier(a1) ]
```

On peut effectivement en déduire que S ∧ E' |- P'. La modification est acceptée si a1 est cohérent avec la définition d'un entier, et éventuellement de âge s'il fait l'objet de contraintes

particulières.

L'algorithme de gestion de l'échantillon E en fonction des opérations est le suivant (on suppose que les opérations et l'état courant de l'échantillon sont indicés chronologiquement de 0 à n) :

```
-----------------------------------------------------------
i := 0
E0 := vide
A : i := i+1
Si l'opération i est une :
     - création d'un objet implicite ou hybride xi alors
          si la classe C(xi) n'a pas de prototype alors
               si S ∧ Ei |- xi alors
                    créer un prototype Pi identique à
                    l'objet xi pour la classe C(xi)
                    Ei+1 := Ei U xi
                    opération certifiée
                    refaire A
               sinon
                    opération rejetée
               finsi
          finsi
     - modification d'un objet xi
       ou création d'un objet dont un prototype Pi existe
       alors
          modifier le prototype Pi de xi, ce qui donne P'i
          modifier Ei en E'i
          si S ∧ E'i |- P'i alors
               Pi+1 := P'i
               Ei+1 := Ei U Pi+1
               opération certifiée
          sinon
               opération rejetée
          finsi
finsi
refaire A.
-----------------------------------------------------------
```

6.8.4 Suppression d'objet.

Lorsqu'un objet explicite est détruit, il y deux actions à réaliser.

Si c'est le dernier élément de sa classe d'équivalence, le prototype correspondant est détruit. Ensuite, tous les objets liés ainsi que leur prototype sont marqués. Ceci est utilisé par la suite de deux façons :

- premièrement pour éviter les propagations cycliques de mises à jour. Les objets marqués ne seront détruits que si les contraintes le spécifient,
- deuxièmement, l'utilisation d'objets potentiellement incohérents sera évitée en contrôlant s'ils sont marqués ou non. La décision finale revenant toujours à l'usager.

Lorsqu'un objet implicite est détruit, les objets implicites et hybrides qui lui sont liés sont marqués. Ceci est appliqué récursivement sur chacun d'eux jusqu'à ce qu'un objet explicite ou un objet déjà marqué soit atteint.

Si c'est un objet hybride qui est détruit, seuls sont marqués les objets implicites auxquels il est lié.

La figure 6.8 résume les opérations effectuées lors de la certification des mises à jour d'objets en fonction de leur nature.

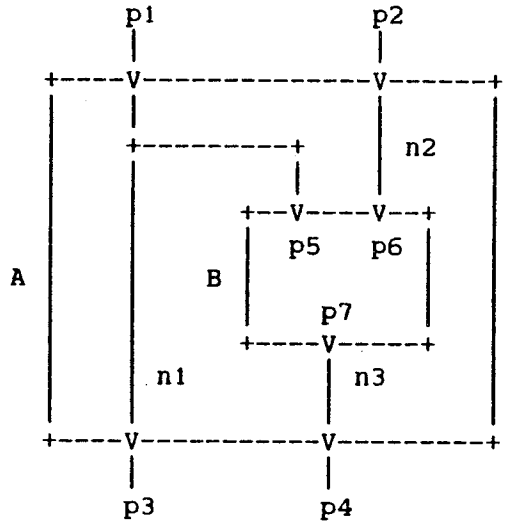
	nature de l'objet		
opération	objet explicite	objet implicite	objet hybride
création	- pas de vérification - créer prototype	- S ∧ E |- objet ou - S ∧ E' |- P'	- S ∧ E |- objet ou - S ∧ E' |- P'
suppression	- supprimer prototype - marquer objets liés	- marquer objets hybrides et implicites liés	- marquer objets implicites liés
mise à jour	- tester les contraintes sur prototype - (*)	- tester S ∧ E' |- P', le prototype modifié - (*)	- tester S ∧ E' |- P', le prototype modifié - (*)

Figure 6.8. Certification des opérations sur les objets.

(*) On a E' = (E - P) U (F1(xi/ti) ∧ P2) où F1 est la conjonction des formules de D contenant des termes ti à modifier.

6.9 Exemple.

A titre d'exemple permettant d'illustrer les éléments qui précèdent, considérons le circuit A représenté ci-dessous:



Il est constitué de trois équipotentielles n1, n2 et n3, d'un composant B dont on ne connaît que l'enveloppe extérieure et les ports d'entrées p5 et p6, et le port de sortie p7. Les ports d'entrée de A sont p1 et p2. Les ports de sortie p3 et p4.

En reprenant la modélisation des circuits à l'aide de formules du premier ordre présentée au Chapitre IV, on obtient pour le circuit A par exemple les définitions suivantes :

```
circuit(A).          schéma_logique(A,LOGA).
                     schéma_électrique(A,ELA).
                     schéma_physique(A,PHYSA).
port_ent(A,p1).      port_ent(A,p2).
port_sort(A,p3).     port_sort(A,4).
équipot(ELA,[n1,n2,n3]).
entrée(n1,p1).       entrée(n2,p2).    entrée(n3,p7).
sortie(n1,p3).       sortie(n2,p6).    sortie(n3,p4).
```

où :
- LOGi, ELi et PHYSi sont les représentations logique, électrique et physique du circuit de nom "i",
- équipot(ELi,nj) est un prédicat qui signifie que nj est une équipotentielle de la représentation électrique ELi,
- port_ent(A,p) signifie que p est un port d'entrée pour le circuit A,
- port_sort(A,p) que c'est un port de sortie pour le circuit A,
- entrée(ni,pj) signifie que le port pj est une entre pour l'équipotentielle ni,
- sortie(ni,pj) que le port pj est une sortie pour

l'équipotentielle ni.

Une équipotentielle est définie par une liste de connexions, des ports d'entrée et de sortie par :

```
equipot(EL,N) :- entrée(N,P1), sortie(N,P2), liste_connexions(N,L),
                 membre(P1,L), membre(P2,L).
liste_connexions(N,L) :- L=[].
liste_connexions(N,[L|L2]) :- L is [P1,P2], connexion(C,N,P1,P2),
                              liste_connexions(N,L2).
```

où membre(P,L) est un prédicat contrôlant que le point P appartient à une des sous-listes de L.

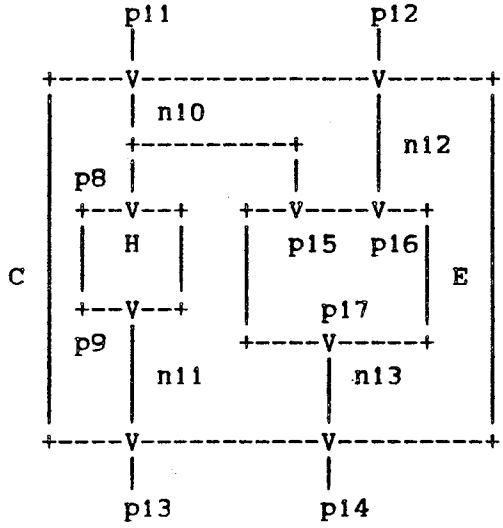
Pour simplifier, supposons que les seules règles de cohérence définies soient :

C1 - aucun port ne doit être inutilisé,
C2 - toute connexion doit aboutir à au moins un port.

Ceci s'exprime sous la forme :

```
C1 : utile(P) :- (entrée(N,P) ; sortie(N,P)), equipot(N).
C2 : connexion(C,N,P1,P2) :- liste_connexions(N,L), membre(P1,L),
                             membre(P2,L), membre(P,L),
                             (entrée(N,P1) ; sortie(N,P1) ;
                              entrée(N,P2) ; sortie(N,P2) ;
                              entrée(N,P) ; sortie(N,P)).
```

Le circuit A vérifie ces contraintes, de même que le circuit C ci-dessous qui n'en diffère que par l'adjonction d'un composant H sans violer les contraintes de connexion :



Indépendamment de leurs fonctions respectives et de leurs représentations, les deux circuits A et C appartiennent à la même classe d'équivalence C(A) puisqu'ils vérifient ici le même

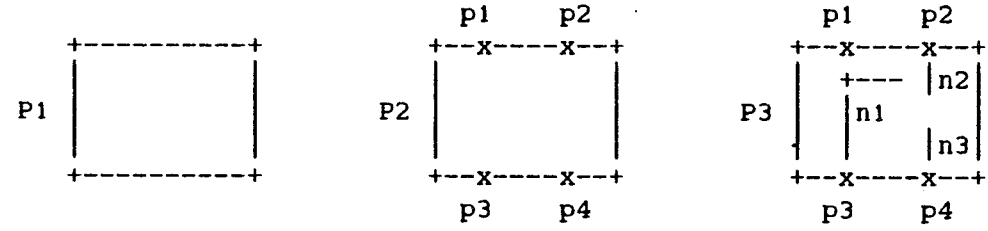
ensemble de contraintes C1 et C2.

Si la classe C(A) est vide avant la création du circuit A, on génère son prototype à partir de A lui-même, au fur et à mesure qu'il évolue.

Les différentes étapes de cette évolution sont les suivantes :

1 - création de l'enveloppe du circuit :

Le circuit est "vide", sa classe d'équivalence également. Ni C1 ni C2 ne sont vérifiées. La certification n'est pas obtenue pour cette création mais l'on passe outre en créant un prototype P1 identique à cette enveloppe.



2 - création des ports d'entrée-sortie p1 à p4 :

La création des ports p1 à p4 est testée sur le prototype du circuit P1. Les contraintes C1 et C2 ne sont toujours pas vérifiées, ce qui est normal. On passe donc outre la certification qui est refusée. On crée un prototype P2 correspondant à ce nouvel objet. Il est défini par :

circuit(P2) :- circuit(P1) ∧ port_ent(P2,p1) ∧ port_ent(P2,p2)
∧ port_sort(P2,p3) ∧ port_sort(P2,p4).

3 - création des équipotentielles n1 à n3 :

On crée de même un prototype P3 correspondant au nouvel objet par connexion de P2 avec lui-même et les formules F qui le modifient. Soit :

circuit(P3) :- circuit(P2) ∧ équipot(P3,n1)
∧ équipot(P3,n2)
∧ équipot(P3,n3).

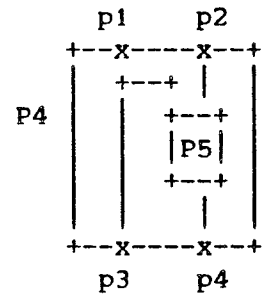
Les contraintes C1 et C2 ne sont toujours pas vérifiées.

4 - création du composant B et connexion aux équipotentielles :

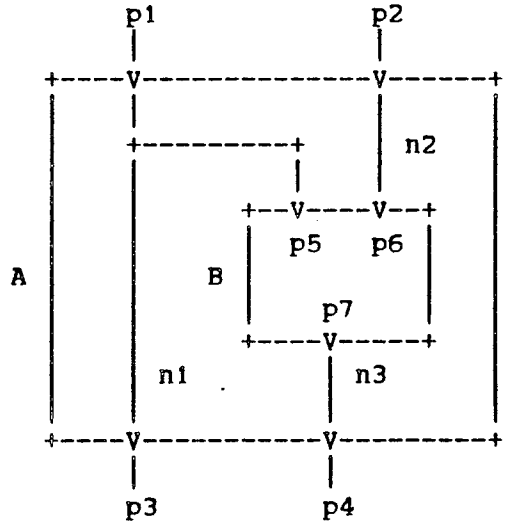
On suppose qu'un prototype P5 existe pour B. Les contraintes C1 et C2 sont vérifiées après adjonction du composant B au prototype P3 par connexion de P3 et du prototype

P5 de B. Cette modification entraîne la création du prototype P4, tel que : P4 = P3 + P5, c-à-d : P4 = P3 ∧ P5. Dès lors, P4 devient le représentant de la classe d'équivalence définie par C1 ∧ C2.

```
circuit(P4) :- circuit(P3) ∧ circuit(P5) ∧
               liste_connexions(n1,[p1,p3,p5]) ∧
               liste_connexions(n2,[p2,p6]) ∧
               liste_connexions(n3,[p4,p7]).
```



P4 est à cet instant le prototype du circuit A ainsi que le représentant de tous les circuits qui appartiennent à sa classe d'équivalence C(A). Ils doivent tous vérifier la conjonction de contraintes C1 ∧ C2.



Supposons que l'on désire créer le circuit C. Conformément à la démarche utilisée habituellement en CAO, la création de C à partir de A se fait par la création d'une copie de A suivie de sa modification. Ici, cela veut dire insérer le composant H dans cette copie, en l'intercalant sur l'équipotentielle n1 entre les ports p1 et p3.

On certifie maintenant les opérations à partir du prototype P4 de la classe d'équivalence de A. Les étapes suivantes sont réalisées

1 - création d'une copie de A.

2 - création ou copie de H s'il existe déjà.

On suppose qu'on ne connaît que l'enveloppe et les ports d'entrée-sortie de H et de P6, qui est défini par :

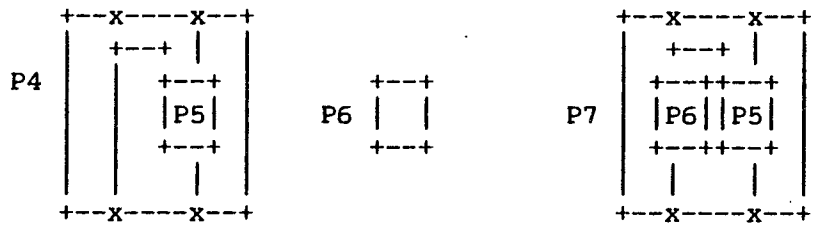
```
circuit(P6).
schéma_électrique(P6,ELP6).
port_ent(P6,p8).
port_sort(P6,p9).
```

3 - connexion du prototype P4 et du prototype P6 de H, qui produit le prototype P7, tel que : P7 = P4 + P6, c-à-d :

circuit(P7) :- circuit(P4) ∧ circuit(P6).

Les connexions entre les ports d' entrée-sortie et les équipotentielles n11 à n13 ne sont pas encore établies, ce qui implique que cette opération n'est pas certifiée. Ceci entraîne en effet une incohérence car les contraintes C1 et C2 ne sont pas vérifiées. Elle est détectée immédiatement sur le prototype P7 du circuit C. C'est bien le but recherché. L' incohérence potentielle du circuit C est détectée dès la modification de P7.

Si le concepteur poursuit normalement la création de C, le prototype sera complété par la connexion des ports p8 et p9 aux équipotentielles n10 et n11. Ceci est constamment contrôlé par la procédure de certification.



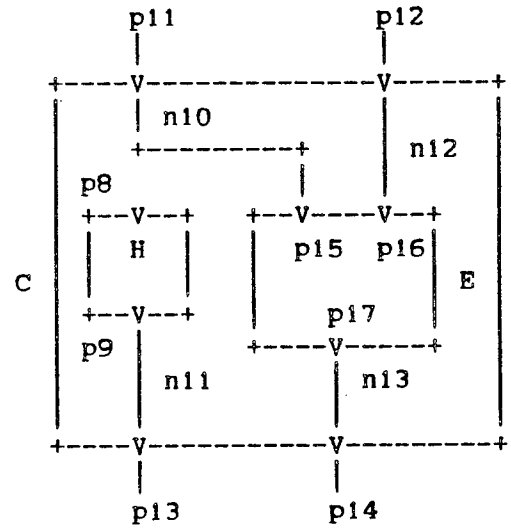
4 - connexion du composant H aux équipotentielles n10 à n11.

On obtient :

```
circuit(P7) :- circuit(P4) ∧ circuit(P6)
                           ∧ liste_connexions(n10,[p8,p11,p15])
                           ∧ liste_connexions(n11,[p9,p13]).
```

Si les connexions sont correctement établies les contraintes C1 et C2 sont vérifiées. La création du circuit C est donc certifiée et peut être validée, c-à-d rendue effective dans la base de données.

Supposons que l'on veuille maintenant modifier le circuit C en supprimant la connexion de l'équipotentielle n10 au port d'entrée p8 du composant H :



Pour certifier cette modification, on l'applique sur le prototype P7 du circuit C. En suivant la démarche du paragraphe 6.9.3, on a :

P = circuit(P7)
P = P1 ∧ P2
P = circuit(P4) ∧ circuit(P6)
∧ liste_connexions(n10,[p8,p11,p15])
∧ liste_connexions(n11,[p9,p13]).

P1 = liste_connexions(n10,[p8,p11,p15]).
F1(ai/ti) = liste_connexions(a1,[a2,p11,p15]).

P2 = circuit(P4) ∧ circuit(P6) ∧ liste_connexions(n11,[p9,p13]).

P' = F1(ai/ti) ∧ P2

E = (circuit(P1), circuit(P2), circuit(P3), circuit(P4), circuit(P5),
circuit(P6), circuit(P7))
E' = (E-P) U (F1(xi/ti) ∧ P2)
E' = (circuit(P1), circuit(P2), circuit(P3), circuit(P4), circuit(P5),
circuit(P6)) U (liste_connexions(x1,[x2,p11,p15]) ∧ P2)

Supprimer la connexion de l'équipotentielle n10 au port p8 est obtenu avec a1=n10 et a2 égal à l'élément "vide", soit :
F1(ai/ti) = liste_connexions(n10,[p11,p15]).

P' = circuit(P4) ∧ circuit(P6) ∧ liste_connexions(n11,[p9,p13]).
∧ liste_connexions(n10,[p11,p15]).

Ceci est certifié si : S U E' |- P' où S est le schéma de la

base de données, qui par définition contient les contraintes C1 et C2.

Il est impossible de prouver P' à partir de E' puisqu'il n'y a pas d' instantiation de la variable x2 qui permette d'atteindre le sous-but : liste_connexions(n10,[p11,p15]).

La contrainte C1 du schéma qui veut qu'aucun port ne soit inutilisé n'est plus respectée.

L'union S U E' ne modifie pas cet état de fait. Il est donc impossible de prouver P', que ce soit à partir de S, de E' ou de S U E'. La modification n'est pas certifiée.

6.10 Validation des opérations.

Dans une architecture répartie telle que celle décrite précedemment (Chapitre V), il est impératif de pouvoir intégrer des outils existant et de disposer de mécanismes de contrôle de cohérence adaptés à la coopération d'outils d'origine diverses, et qui travaillent éventuellement sur des structures de données différentes.

On a adopté une formalisation en logique du premier ordre des données et des modèles afin de disposer :

- d'une part d'un langage commun d' expression des faits élémentaires,
- d'une théorie compatible avec les modèles de données couramment utilisés dans les applications avancées, en particulier relationnel et entité-association,
- des mécanismes de déduction implantés dans les langages basés sur l' interprétation des clauses de Horn,
- de la possibilité d'implanter au dessus des modèles et des outils utilisés une interface de haut niveau, essentiellement orientée objet [ZAN84].

La juxtaposition de divers modèles de données et d'outils de natures différentes implique une coopération contrôlée par des mécanismes nouveaux. C'est la raison pour laquelle on définit un sous-système de vérification de contraintes sémantiques dénommé SYCSLOG, dont Judith Olivares a fait une première réalisation dans le cadre du projet TIGRE [OLI86].

L'architecture générale du sous-système de validation repose sur trois composants :

- un processeur logique de validation,
- une interface utilisateur avec une base de méta-connaissances,

- une interface avec les bases de données et les outils externes.

La figure 6.9 décrit sommairement l'architecture du sous-système de validation.

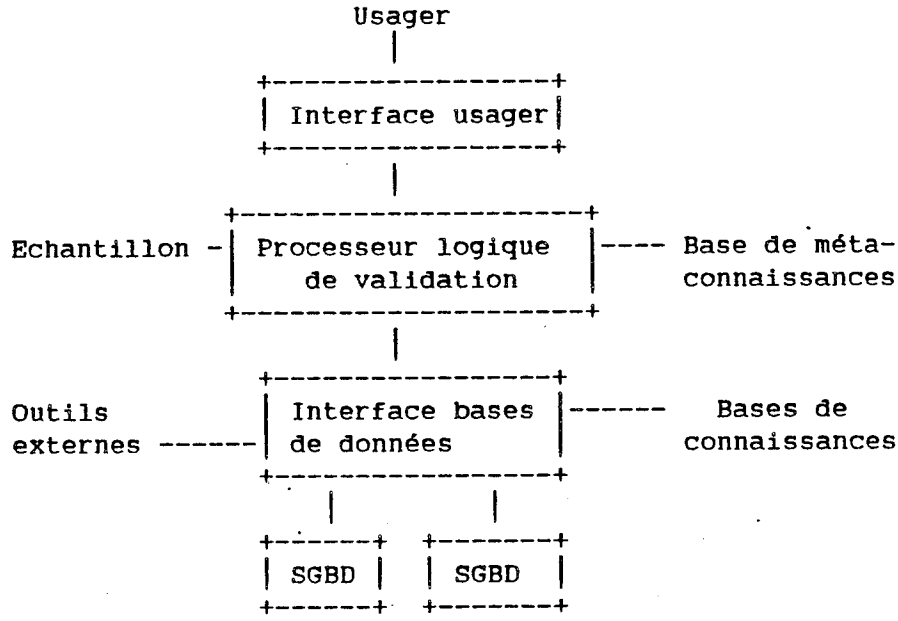


Figure 6.9. Architecture du sous-sytème de validation.

Le processeur logique de validation est le cœur du sous-système. Il pilote l'ensemble des actions entreprises par le SGBDA pour le contrôle des contraintes sémantiques.

On rappelle que ces contraintes sémantiques comprennent :

- la conception des schémas de données,
- leur modification, tant en ce qui concerne la structure des objets (leurs propriétés), que les contraintes associées,
- enfin la modification des instances d'objets proprement dites.

On rappelle ici brièvement ses différents composants.

o Processeur logique de validation.

Ce module utilise une base, dite de "méta-connaissances", pour contrôler la construction et la modification des schémas de données. On rappelle que ceux-ci sont définis comme des ensembles non vides de définitions d'objets, à l'aide de règles de spécification, de déduction de propriétés et de contraintes d'intégrité (Chapitre IV).

La base de méta-connaissances contient un ensemble d'axiomes exprimés à l'aide de clauses de Horn. Ces axiomes constituent un formalisation des concepts des modèles de données utilisés et des opérations licites sur ces concepts.

Le processeur logique de validation utilise un moteur d'inférence, en l'occurrence le mécanisme de résolution utilisé par le langage Prolog, pour contrôler la validité des opérations effectuées par les usagers sur les schémas des bases de données.

Ceci est réalisé par l'activation d'une règle qui spécifie une fois pour toutes la sémantique d'une opération sur un schéma de données. Le nombre d'opérations étant limité (création d'objet, modification de propriété ou de contrainte), le contenu de la base de méta-connaissances peut être défini par un nombre réduit de clauses Prolog, dont l'activation délivre une valeur booléenne correspondant à l'acceptation ou au rejet d'une requête.

Pour contrôler les opérations effectuées sur les objets proprement dits par les programmes d'application, le processeur logique de vérification utilise des bases dites de "connaissances", qui contiennent les informations sémantiques propres à chaque application. On entend ici par bases de connaissances l'ensemble des règles qui sont exprimables à l'aide de clauses de régulières et relatives à un ou plusieurs traitements sur une base de données. On sait que ce type de formalisation peut être effectué pour des domaines aussi complexes que la CAO [CET84].

o Interface bases de données.

L'interface bases de données est utilisée pour fournir au processeur de validation une vision en logique du premier ordre des informations structurelles et élémentaires des bases de données utilisées. Elle peut interfacer indifféremment des SGBD proprement dits ou des fichiers classiques. Cette capacité à accéder à des informations stockées dans des bases de données ou des outils différents est essentielle, car elle fournit un moyen relativement simple pour intégrer des programmes existant ou des données prédéfinies. C'est les cas par exemple de programmes de test ou de simulation en CAO. C'est également le cas pour des librairies de circuits ou de composants prédéfinis, dont l'utilisation est imposée par les méthodes de conception utilisées actuellement.

Elle est implanté à l'aide de systèmes de réécriture. Une réalisation de cette interface pour le SGBD relationnel MICROBE a été réalisée par Pascale Winninger.

o Interface utilisateur.

L'interface utilisateur est dédiée au contrôle des opérations effectuées par un utilisateur, conformément aux règles définies dans les bases de connaissances. Elle fournit une vision des informations basée sur la notion d'objet. Elle est également utilisée pour contrôler la validité des mises à jour des bases de connaissances.

6.11 Répartition.

L'intérêt de la vérification en deux phases est évident dans le cadre d'un système réparti. Cette compatibilité avec les méthodes classiques de verrouillage et de mise à jour dans les systèmes distribués est exploitée lors de l'utilisation d'informations réparties (Chapitre V).

Soit par exemple deux objets OB et OB' liés entre eux (OB' est un sous-objet de OB). Soient AT, AT' et AT'' leurs attributs (Figure 6.7).

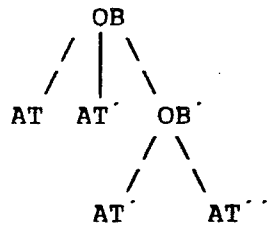


Figure 6.7. Structure des objets OB et OB'.

On suppose ici que les attributs AT' de OB et OB' doivent avoir la même valeur et que AT + AT' de OB = AT' + AT'' de OB'.

Si l'on met à jour l'attribut AT de l'objet OB en lui donnant la valeur V, noté : MAJ (OB, AT, V), trois cas se présentent :

- l'attribut n'a aucun lien, c-a-d aucune contrainte qui le lie à un autre attribut (Cas 1),
- l'attribut AT a un lien avec d'autres attributs, par exemple AT + AT' de OB doit être égal à AT' + AT'' de OB', et la mise à jour MAJ (OB, AT, V) ne viole pas cette contrainte (Cas 2),
- la mise à jour viole au moins une contrainte associée au lien : par exemple l'égalité AT + AT' = AT' + AT'' n'est plus respectée (Cas 3).

Cas 1 : la mise à jour MAJ (OB, AT, V) est testée sur le prototype. L'absence de lien est contrôlée par la phase de certification et la mise à jour est certifiée (CERTIFIE). La validation proprement dite est faite à la demande (VALIDER).

```
(1)   MAJ (OB, AT, V)  -->
(2)                    <-- CERTIFIE
(3)   VALIDER          -->
```

Cas 2 : la mise à jour est contrôlée par l'examen des valeurs de AT' de OB (LIRE (OB, AT')) et de AT'' de OB' (LIRE (OB', AT'')). Ces valeurs sont dans ce cas de figure conformes à la contrainte AT + AT' = AT' + AT'', la modification est certifiée (4), puis éventuellement validée (5).

```
(1)  MAJ (OB, AT, V)  -->
(2)                   <-- LIRE (OB, AT')
(3)                   <-- LIRE (OB', AT'')
(4)                   <-- CERTIFIE
(5)  VALIDER          -->
```

Cas 3 : la phase de certification contrôle la valeur de AT' de OB et de AT'' de OB'. La contrainte d'égalité étant violée, il y a demande de mise à jour des valeurs de AT' et AT'' (MAJ (OB,AT', ?) et MAJ (OB', AT'', ?)). En cas de refus de ces mises à jour, il y a échec de toute l'opération (ECHEC), sinon certification (8' et 9').

```
(1)  MAJ (OB, AT, V)  -->
(2)                   <-- LIRE (OB, AT')
(3)                   <-- LIRE (OB', AT'')
(4)                   <-- MAJ  (OB, AT', ?)
(5)                   <-- MAJ  (OB', AT'', ?)
(6)  MAJ (OB', AT'..) -->
(7)  MAJ (OB', AT''.) -->
(8)  ou ECHEC         -->
(8')                  <-- CERTIFIE
(9') VALIDER          -->
```

Il est clair que ce fonctionnement est compatible avec les méthodes de mises à jour d'informations réparties et partagées [NGU79]. Dans l'exemple précédent, les objets OB et OB' peuvent être réparties sur plusieurs sites différents. Le protocole de mise à jour à deux phases peut alors être "immergé" dans les phases de certification et de validation avec des adaptations minimes.

Dans la réalité, le schéma est complété par la copie de chaque

objet mis à jour. Ce qui veut dire qu'on ne supprime jamais l'information. On doit donc faire intervenir ici un mécanisme de gestion de versions [KIM85].

Selon la nature des objets touchés par une opération de mise à jour, c-à-d s'ils sont explicites, implicites ou hybrides, les règles suivantes peuvent alors être définies pour contrôler la conformité des mises à jour aux contraintes sémantiques définies dans le schéma de la base de données.

6.12 Limites.

La question qui reste en suspens à ce stade concerne la généralité de la méthode de certification proposée. C-à-d concrètement, est-il possible de lever les restrictions imposées par les théorèmes 1 à 3 du paragraphe 6.6 ? Pour cela, on définit la complétude d'une méthode de vérification de contraintes de la façon suivante :

> Une méthode de vérification de contrainte M1 est complète par rapport à une autre méthode M2 si toute contrainte C évaluée à vrai par M1 sur un ensemble d'informations l'est également par M2.

En d'autres termes, il faut démontrer ici que la certification des opérations qui utilise les notions de prototype et d'échantillon est complète par rapport à une méthode classique de vérification de contraintes qui s'exerce directement sur les données de base.

Dans le cas général, la réponse est non car le test de cohérence, c-à-d de non contradiction, d'un ensemble de formules saines du premier ordre est indécidable.

On peut néanmoins particulariser la méthode afin de restreindre son champ d'application à des cas où l'on sait qu'elle s'applique. Cela dépend en particulier du type de contrainte à vérifier.

On distinguer parmi celles-ci les contraintes de valeur et les contraintes de liaison.

A titre d'exemple, une contrainte de valeur peut être pour un attribut AT' de l'objet OB (Figure 6.7), l'obligation d'être inférieur ou égal à la valeur de l'attribut AT et d'être toujours supérieur à une certaine borne inférieure B, c-à-d :

$$B <= OB.AT' <= OB.AT .$$

Dans ce cas, la vérification de contrainte impose la vérification de : B <= OB.AT', ce qui est immédiat aussi bien par la méthode

de certification que par toute autre.

La vérification de contrainte nécessite la comparaison des valeurs de AT et AT'. Si AT et AT' possèdent tous deux des valeurs avant la mise à jour, la certification est possible.

Si par contre AT ou AT' ne sont pas valués au départ, la certification peut ne pas être complète. En effet, la certification a lieu sur les prototypes formant l'échantillon à un instant donné. Il se peut que, par définition, certains attributs ou certaines propriétés d'un prototype soient indéfinis à cet instant. Il est également plausible que l'on veuille modifier AT pour respecter la contrainte AT' <= AT après modification de AT'. Dans ce cas, la réponse est indécidable.

La solution à ce problème consiste soit à délivrer une réponse "indéterminé", soit à générer explicitement une valeur pour cet attribut. Dans ce dernier cas, il faut rechercher une valeur existant déjà dans la base de données. Une politique de génération par "immersion" du prototype dans la base peut alors se révéler efficace.

La génération par "immersion" consiste à augmenter les caractéristiques d'un prototype donné par ajout ou modification de valeurs à partir des données contenues dans la base. En prenant garde que l'ensemble des contraintes reste vérifié sur ce prototype "augmenté", la vérification de contraintes qui resteraient autrement indéterminées devient possible. Cette opération est similaire à l'connexion de prototypes, mais concerne ici un prototype et un objet de la base de données.

6.13 Comparaison avec d'autres approches.

Habituellement, la logique et en particulier celle des prédicats du premier ordre a été utilisée pour l'interrogation des bases de données et le contrôle d'intégrité [DEM82, NIC79, NIC82]. Dans ce dernier cas, les travaux ont surtout porté sur la simplification et l'évaluation de contraintes exprimant des dépendances fonctionnelles et multivaluées. Des travaux plus récents concernent le contrôle de l'intégrité sémantique et la construction de schémas valides [BRY86]. Dans ce dernier cas cependant, seuls sont considérées les règles de formation des schémas par rapport à la formalisation d'un modèle de données en logique du premier ordre. Les répercussions de leur mise à jour sur la cohérence et la complétude des informations de la base ne sont pas abordées. D'autres travaux ont pour but la conception de modèles de données évolués [KUP85, ZAN84].

Avec l'intégration des fonctionnalités des systèmes déductifs et des SGBD, s'ouvrent de nouvelles perspectives d'application. L'un

des domaines qui fait l'objet de travaux intensifs concerne les applications de conception assistée par ordinateur [CET84, FIN85, NGU85].

Une première étape a été franchie avec la réalisation d'outils d'aide à la conception. En particulier, le test de circuits intégrés et l'optimisation automatisée de fonctions implantées à l'aide de PLA ont été réalisés avec des systèmes basés sur la logique des prédicats [HOR84, ZAU83].

Des études plus larges ont pour objectif la réalisation d'architectures réparties plus générales, englobant des SGBD, des systèmes d'aide à la décision, des outils propres aux applications, dans le domaine du génie civil et du génie mécanique par exemple [HOW84].

> L'idée de base des prototypes et de l'échantillon présentés ici est de contribuer à une méthode de conception dans laquelle les contrôles de validité des étapes intermédiaires sont effectués le plus tôt possible.

Pour cela, on évite de balayer tout l'espace des données. On effectue les vérifications à priori sur un prototype qui représente la classe de l'objet à concevoir.

Une approche similaire est prise dans RESIDUE [FIN85]. L'objet à concevoir y est défini par un ensemble de propositions logiques. Un ensemble de règles de résolution particulières est défini pour permettre la génération d'un "résidu", c-à-d d'un ensemble de propositions qui permet de construire l'objet. Cette synthèse progressive se fait par augmentation pas à pas du résidu. Chaque étape caractérise la connaissance partielle que l'on a de l'objet grâce aux clauses qui sont supposées vraies à cet instant et qui forment le résidu.

Les hyptohèses de base sont les mêmes :

- les contraintes auxquelles doit répondre l'objet sont définies à priori,
- la génération se fait par synthèse,
- le mécanisme de déduction est celui de la logique du premier ordre,
- il permet d'évaluer les contraintes à priori, dans un cas pour éviter de générer un prototype ne répondant pas à la question, dans l'autre pour éviter de générer un résidu par des propositions non unifiables.

Les différences notables sont :

- une représentation figée à l'avance de l'objet et de son prototype dans notre approche,
- une représentation non fixée à l'avance dans RESIDUE, ce

qui permet de construire de façon incrémentale l'objet par augmentation de l' ensemble des propositions qui le représentent,
- la possibilité de déduire de nouvelles contraintes par résolution clausale sur ces propositions, ce qui peut réduire l'espace de recherche dynamiquement,
- tandis que l'ensemble des contraintes associées à un prototype est quant à lui fixé une fois pour toutes.

D'autres travaux ont également fait le lien entre la représentation partielle d'informations et la manipulation des bases de données en termes de partitions et de classes d'équivalence [SPY84]. Malheureusement, la mise à jour des données n'y est pas du tout étudiée [LAU86]. Un modèle probabiliste est parfois utilisé pour caractériser les partitions [GEL84]. Dans d' autres, il est fait appel à une algèbre booléenne [KEB86]. Dans ces études, l'objectif est de caractériser les propriétés de l' information à partir d'un découpage de la base de données en sous-ensembles répondant aux mêmes critères. Ceci permet de définir des <u>représentations incomplètes</u> ou <u>résumé</u> des données, à ne pas confondre avec des représentations de données incomplètes.

Les prototypes répondent de manière unifiée à ces deux besoins :

- représenter des informations incomplètes, puisqu'ils sont représentatifs à tout instant des objets appartenant à des classes d'équivalence et qu'ils sont modifiés dynamiquement,
- grâce à l'échantillon qu'ils forment, être considérés comme une représentation partielle (donc incomplète) des éléments d'une base de données, puisqu'il n'y a qu'un représentant par classe d'équivalence.

Si une similitude avec notre approche existe quant à cette caractérisation sous forme de classes d'équivalence, on n'y trouve cependant pas de méthode de contrôle de contraintes d'intégrité ou de contrôle des opérations de mise à jour analogue à la certification. Bien qu'elles fassent appel aux notions habituelles des mathématiques élémentaires, leur étude dans le cadre de la logique du premier ordre et des systèmes déductifs n'est pas abordée.

6.14 Application : un Système de Base de Données Expert en CAO.

Comme il a été vu au Chapitre III, une singularité des applications avancées concerne les structures de données : elles sont évolutives, contrairement aux applications traditionnelles de gestion dans lesquelles elles sont figées. C'est ainsi qu'en Conception Assistée par Ordinateur, un objet à concevoir peut être modifié tant en valeur qu'en structure jusqu'à ce que son modèle soit satisfaisant.

On propose ici d'appliquer les concepts de proptotype logique et de certification dans le cadre des applications CAO. On définit pour cela des règles heuristiques qui permettent d'adapter directement ces notions au cas particulier d'un Système de Bases de Données Expert en CAO.

Le qualificatif "Expert" est utilisé à dessein. Il concerne :

- d'une part la définition de règles d'optimisation qui sont le reflet de la connaissance que l'on a de l'application. Elles sont utilisées ici pour réduire le nombre de classes d'équivalence et donc accélérer la détermination dynamique de la classe d' un objet lors de ses modifications (paragraphe 6.14.1),

- d'autre part la définition de règles opératoires concernant la certification des opérations de mise à jour d'un usager. Elles sont également fortement liées à la sémantique de l'application et au modèle de données utilisé.

Ces règles, qui sont destinées aux bases de connaissance du Système, vont permettre une interaction automatique entre les connaissances gérées par le processeur logique de validation et d'une part les opérations des concepteurs, d'autre part les informations gérées par l'application.

En effet, la certification des opérations est réalisée par l'application sur les prototypes des requêtes utilisateurs. La caractérisation de la classe d'un objet doit donc être réalisée après chaque mise à jour. De son résultat dépend la notion de validité d'une opération (paragraphe 6.8). Il importe donc qu'elle soit très efficace. Ceci est en particulier fonction du nombre de classes à examiner que l'on va réduire grâce aux règles d'optimisation décrites ci-dessous (pragraphe 6.14.1).

Enfin la généralité des propositions de ce chapitre peut être restreinte afin de les adapter à des cadres spécifiques. Les restrictions qui en résulteront seront des facteurs d'optimisation supplémentaires.

De plus des connaissances expertes dans un domaine paticulier seront également des facteurs d'optimisation qui doivent être

pris en compte (paragraphes 6.14.2).

Ces différents aspects sont étudiés sur un cas particulier. L'accent est mis ici sur la notion de règles spécifiques à la CAO. Des exemples simples en montrent à la fois l'utilité et la faisabilité.

6.14.1 Réduction du nombre de classes.

Un calcul direct de la fermeture de l'ensemble des formules syntaxiquement dérivables donne théoriquement un grand nombre d'éléments. Ce calcul peut s'avérer à la fois long et inutile, même s'il est entièrement automatisé.

Un grand nombre de classes n'ont en effet aucune signification. Elles sont soit sans objet car sémantiquement incohérente, soit inutiles dans une application particulière lorsque l'on contraint les objets à respecter certaines configurations. Ceci est par exemple le cas lorsque le modèle de données utilisé autorise certaines formes d'incohérence (contraintes utilisées en règles de cohérence faible) ou utilise au contraire les contraintes d'intégrité en règle de cohérence forte [FAU86, RIE87].

Soit par exemple les classes point et segment définies par les coordonnées (abs et ord) et une origine (org), une extrémité (ext) et une longueur (lg) :

point(p) <- nom(p,n) ∧ chaîne(n) ∧ abs(p,x) ∧ réel(x) ∧ ord(p,y) ∧ réel(y)
segment(s) <- nom(s,n) ∧ chaîne(n) ∧ org(s,o) ∧ point(o) ∧ ext(s,e) ∧ point(e) ∧ lg(s,l) réel(l) ∧ egal(l,dist(o,e)) ∧ not(conf(o,e)).

La longueur d'un segment ne peut évidemment être calculée qu'en connaissant ses points origine et extrémité. La classe d'équivalence définie par la conjonction :
nom(s,n) ∧ chaîne(n) ∧ lg(s,l) ∧ réel(l) est donc sans objet en ce qui concerne les segments.

Les règles suivantes sont utilisées pour caractériser les classes d'équivalence <u>significatives</u>. Elles sont fonction du modèle de données utilisé et de l'application considérée :

R1 - les contraintes décidables sont immédiatement testées.

> Une contrainte est <u>décidable</u> si elle peut être prouvée vraie ou fausse dans le cadre du système considéré. On utilise ici la logique du premier ordre. Cette notion est donc cohérente avec la notion de décidabilité d'une formule. Les classes d'équivalence sont ici étendues pour refléter cette

notion : une contrainte décidable fausse détermine une classe au même titre qu'une contrainte décidable vraie.

R2 - les contraintes décidables sont immédiatement testées et respectées :

Les systèmes CAO mettant en avant des méthodes de conception ascendantes, les classes définies par une propriété et la négation de son type sont éliminées : un segment ne peut pas appartenir à la classe définie par la conjonction
segment(s) <- nom(s,n) ∧ chaîne(n) ∧ org(s,o) ∧ not(point(o)).
De plus, le contrôle du type des objets et de leurs propriétés est effectué systématiquement pour chaque instantiation ou modification. Les classes significatives sont donc celles qui comporte pour chaque objet ou propriété d'un objet son type :
segment(s) <- org(s,o) est éliminé car il manque point(o),
segment(s) <- org(s,o) ∧ point(o) est par contre une classe significative.

R3 - les réalisations d'objet ont toujours des noms instantiés :

Les réalisations d'objets sont toujours identifiées par un nom discriminant. L'instantiation de nom(x,n) est donc obligatoire. Conjointement avec la règle R2, la conjonction nom(x,n) ∧ chaîne(n) est donc obligatoire pour chaque classe d'équivalence. Toutes les autres classes sont éliminées.

R4 - les propriétés calculables par des fonctions sont évaluables si et seulement si tous leurs arguments sont instantiés :

La longueur n'est évaluable que si les points origine et extrémité d'un segment sont connus. Cette règle élimine toutes les classes définies à l'aide de contraintes ou fonctions sans tous leurs arguments :
segment(s) <- nom(s,n) ∧ chaîne(n) ∧ not(conf(o,e))
est éliminée.

R5 - les fonctions sont toujours évaluables :

Cette règle élimine les classes définies à l'aide de la négation d'une fonction. Ceci est dû au fait qu'une fonction est assimilée à une contrainte de cohérence forte.

R6 - les propriétés définies par des fonctions ne peuvent être instantiées qu'à travers ces liens :

Ceci élimine les classes définies à l'aide d'une propriété calculée sans la formule où la fonction intervient explicitement. Ainsi, la classe définie par :

segment(s) <- nom(s,n) ∧ chaîne(n) ∧ org(s,o) ∧ point(o) ∧ ext(s,e) ∧ point(e) ∧ lg(s,l) ∧ réel(l)
est éliminée car le terme : égal(l,dist(o,e)) est absent.

R7 - les contraintes décidables sont immédiatement évaluées et respectées :

Un système d'assistance en CAO doit fournir un mécanisme d'évaluation dynamique de contraintes. Une contrainte décidable, c-à-d ici dont tous les arguments sont instantiés doit donc pouvoir être testée immédiatement. Ceci élimine toutes les classes définies à l'aide des arguments d'une contrainte en l'absence du terme qui la fait apparaître. Ainsi la classe définie par :
segment(s) <- nom(s,n) ∧ chaîne(n) ∧ org(s,o) ∧ point(o) ∧ ext(s,e) ∧ point(e) ∧ lg(s,l) ∧ réel(l) ∧ égal(l,dist(o,e))
est éliminée car la contrainte not(conf(org,ext))) est absente.

R8 - les fonctions évaluables sont toujours évaluées et leurs contraintes respectées :

Ceci est encore dû au fait que les fonctions sont utilisées en règle de cohérence forte. La classe définie par :
segment(s) <- nom(s,n) ∧ chaîne(n) ∧ org(s,o) ∧ point(o) ∧ ext(s,e) ∧ point(e) ∧ not(conf(o,e))
est éliminée car lg(s,l) ∧ égal(l,dist(o,e)) est absent.

En utilisant ces huit règles de réduction de la fermeture, on obtient pour l'exemple des points et des segments les cinq classes suivantes :

C1 - la classe de tous les segments connus :

segment(s) <- nom(s,n) ∧ chaîne(n)

C2 - la classe des segments avec point origine connu :

segment(s) <- nom(s,n) ∧ chaîne(n) ∧ org(s,o) ∧ point(o)

C3 - la classe des segments avec point extrémité connu :

segment(s) <- nom(s,n) ∧ chaîne(n) ∧ ext(s,e) ∧ point(e)

C4 - la classe des segments avec points origine et extrémité connus et confondus :

segment(s) <- nom(s,n) ∧ chaîne(n) ∧ org(s,o) ∧ point(o) ∧ ext(s,e) ∧ point(e) ∧ lg(s,l) ∧ réel(l) ∧ égal(l,dist(o,e)) ∧ conf(o,e)

C5 - la classe des segments avec points origine et extrémité connus et non confondus :

segment(s) <- nom(s,n) ∧ chaîne(n) ∧ org(s,o) ∧ point(o) ∧ ext(s,e) ∧ point(e) ∧ lg(s,l) ∧ réel(l) ∧ égal(l,dist(o,e)) ∧ not(conf(o,e))

Les classes C1 à C3 correspondent à des objets incomplets et cohérents (Chapitre III). La classe C4 correspond à des objets complets et incohérents. La classe C5 correspond à des objets complets et cohérents.

6.14.2 Règles opératoires.

L'adaptation des principes de certification est faite ici sous l'angle opérationnel : on définit des règles qui permettent de contrôler les opérations de mise à jour en fonction des seules complétude et cohérence des objets.

La définition de la relation d'ordre entre les classes d'équivalence est par voie de conséquence modifiée (paragraphe 6.2).

La relation d'inclusion qui permettait d'ordonner les classes est ici remplacée par la notion de nombre de formules décidables pour un objet.

Une classe d'équivalence C1 sera "inférieure" à une classe C2, noté C1 < C2, si et seulement si les objets de la classe C1 vérifient moins de contraintes ou possèdent moins de propriétés instantiées que les objets de C2.

Le nombre de formules atomiques satisfaites par les objets de C1 sera donc moins grand que celui des objets de C2. Les classes C1 et C2 du paragraphe 6.14.1 sont par exemple liées par : C1 < C2, car C2 est définie à l'aide de la formule supplémentaire : org(s,o) ∧ point(o).

C1 : segment(s) <- nom(s,n) ∧ chaîne(n)
C2 : segment(s) <- nom(s,n) ∧ chaîne(n) ∧ org(s,o) ∧ point(o)

Pour affiner les notions de complétude et de cohérence et pour les prendre en compte directement dans le processus de certification des objets, on définit également ici deux notions.

D'une part le degré de complétude d'un objet CAO. D'autre part le degré de cohérence d'un objet.

Degré de complétude d'un objet.

Soit N le nombre depropriétés d'un objet. On dira qu'un objet est P-complet, ou complet de degré P, s'il possède P propriétés instantiées (0 <= P <= N). Le nombre de propriétés indéfinies est alors N-P.
Les objets de la classe C2 du paragraphe 6.14.1 sont donc 2-complets : leur nom et leur origine sont connus.
Un objet est complet s'il est N-complet.

Degré de cohérence d'un objet.

Soit M le nombre de contraintes définies pour un objet. Un objet sera dit L-cohérent, ou cohérent de degré L, s'il possède L contraintes indécidables ou satisfaites. L'objet possède donc M-L contraintes non satisfaites.

On rappelle que les contraintes non satisfaites (c-à-d décidablement fausses) définissent les classes d'équivalence au même titre que les contraintes satisfaites. La notion de degré de cohérence définie ici est donc moins forte que la notion de satisfaction d'une formule en logique du premier ordre. Elle permet de prendre en compte l'incomplétude des objets dans l'évaluation de leur cohérence sans bloquer leur conception incrémentalle (Chapitre III).

Les objets de la classe C4 ci-dessus sont 0-cohérent car la seule contrainte not(conf(o,e)) est violée. Les objets des classes C1 à C3 et C5 sont 1-cohérents car la contrainte not(conf(o,e)) est soit indécidable soit vérifiée.

Un objet est cohérent s'il ne viole aucune contrainte décidable : il est M-cohérent.

Les règles R9 à R11 sont stockées dans la base de connaissances du Système de Bases de Données Expert. Elles déterminent les opérations licites en fonction des degrés de cohérence, de complétude et de la relation d'ordre entre classes.

R9 - Les mises à jour des propriétés d'un objet sont certifiées si le degré de cohérence de son prototype ne décroît pas :

Les mises à jour des propriétés peuvent faire changer l'objet de classe d'équivalence. Cette règle permet permet de maintenir la cohérence liée à une propriété en garantissant que le degré de cohérence de l'objet ne décroît pas.

En d'autres termes, après mise à jour d' une propriété, l'objet doit soit rester dans la même classe soit appartenir à une classe supérieure au sens de la relation d'ordre.

Si Ci(Pi,Li) désigne la classe Ci des objets Pi-complets et Li-cohérents, la mise à jour d'un objet de C1(P1,L1) est certifiée si son prototype passe dans une classe C2(P2,L2) telle que : P1 = P2 et L1 <= L2.

R10 - l'instantiation de propriétés est toujours certifiée :

Un Système de CAO doit permettre des essais et corrections d'objets modifiés. Les concepteurs sont donc autorisés à augmenter un objet sans considération immédiate de leur degré de cohérence.

L'instantiation d'une propriété pour une objet de la classe C1(P1,L1) est toujours certifiée si son prototype passe dans une classe C2(P2,L2) telle que :
C1(P1,L1) < C2(P2,L2) et P1 < P2.

R11 - les propriétés d'un objet peuvent être désinstantiées si le degré de cohérence de son prototype augmente :

Cette règle est complémentaire à la précédente. Si la valuation d'une propriété entraîne des violations de contraintes, le concepteur doit pouvoir défaire sa mise à jour.

La modification est certifiée si la nouvelle classe C2(P2,L2) est telle que :
C1(P1,L1) < C2(P2,L2) où P1 > P2 et L1 < L2.

Ces règles sont des exemples de connaissances à l'aide desquelles il est possible, et souhaitable, d'ajuster les mécanismes de certification dans un cadre particulier.

Leur utilité vient de ce qu'elles permettent un contrôle à priori des mises à jour, tout en calculant automatiquement les effets de bords sur les prototypes des objets. Ceci les distingue de manière essentielle des méthodes classiques de conception car ces dernières ne permettent une évaluation de la complétude et de la cohérence des objets qu'après l'exécution des opérations sur ces objets.

Elles font partie intégrale d'un Système de Bases de Données Expert en CAO.

6.15 Conclusion.

On présente une méthode de contrôle de contraintes sémantiques pour les applications des bases de données avancées. Par opposition aux méthodes traditionnelles, elle ne requiert pas un contrôle exhaustif des opérations sur l'ensemble des informations. Elle ne nécessite pas non plus la détermination du sous-ensemble minimal de données affectées par une mise à jour.

Son originalité réside dans l'évaluation à priori des effets des mises à jour sur un échantillon constitué de prototypes logiques qui sont des représentants de chaque classe d'équivalence de données. Ces classes sont définies par les contraintes vérifiées à un instant donné par les informations contenues dans la base.

On s'appuie sur une formalisation en logique des propriétés de l'échantillon pour définir la certification des opérations.

Une opération est dite certifiée si son application sur le prototype de l'objet à modifier est valide.

Une opération sur un objet est valide si l'ensemble des contraintes vérifiées par l'objet après l'opération ne décroît pas.

Les caractéristiques de cette méthode permettent de la qualifier de pessimiste, car toute opération sur un prototype qui viole une contrainte quelconque de sa classe d'équivalence est rejetée.

A la connaissance de l'auteur, cette méthode est à ce jour inédite.

CHAPITRE VII

CONCLUSIONS

Le mariage harmonieux entre plusieurs domaines de recherche est un exercice difficile et périlleux. Il est difficile car les méthodes de travail et le vocabulaire sont souvent différents. Il est périlleux car les idées simples et génératrices de progrès sont rares, et controversées de tous bords.

Le développement de nouvelles fonctionnalités pour les SGBD s'impose du fait de leur usage dans des domaines d'applications de plus en plus variés. Il est clair que les contraintes imposées par ces dernières, et en particulier la CAO ou la Bureautique, ont conduit à des recherches intensives dans les domaines de la modélisation d' informations complexes, de partage en environnement distribué et de contrôles sémantiques évolués.

Mais alors que la majorité des travaux se sont orientés vers l' utilisation des SGBD comme support de ces applications, peu d'études ont été menées concernant des extensions qui les fassent profiter réellement des mécanismes déductifs.

On propose ici d'utiliser un certain nombre de techniques venant d'une part de la technologie des bases de données pour :

- la mise en œuvre d'informations réparties,
- la certification destinée au contrôle de la cohérence,

et de techniques qui viennent de l'Intelligence Artificielle, et plus particulièrement de la programmation en logique, pour étendre les fonctionnalités des Bases de Données en ce qui concerne :

- la spécification de la sémantique des informations,
- la modélisation des objets,
- le calcul d'effets de bord durant leur manipulation.

Les limites théoriques de cette démarche sont à priori peu contraignantes. La logique des prédicats offre en effet un cadre rigoureux de spécification et de modélisation des informations et des traitements. Elle englobe également la théorie du modèle relationnel, y compris ses extensions aux valeurs nulles et incomplètes. Ses développements plus récents en direction des objets complexes et d'héritage de propriétés rentrent également de façon très naturelle dans son champ d'application.

On sait par ailleurs que dans le domaine de la logique appliquées aux Bases de Données, la théorie du modèle et la théorie de la preuve sont équivalentes en ce qui concerne l'évaluation de formules saines du premier ordre. Contrairement à l'approche largement répandue dans ce secteur qui est d'utiliser la théorie du modèle, on utilise ici la théorie de la preuve pour définir de nouveaux concepts, comme le prototype (logique) d'un objet et l'échantillon d'une base de données. Ces notions ne trouveraient leur place dans la théorie du modèle qu'en utilisant des outils plus évolués, comme la logique modale. De fait, on n'a besoin ici que de la logique des prédicats du premier ordre, ce qui a l'avantage de rester très proche des concepts habituels utilisés en matière de modèles de données.

Un certain nombre de problèmes d'ordre théorique demeurent cependant qui n'ont pas été abordés ici. Ils concernent en particulier les limites imposées par la logique du premier ordre et son interprétation par les méthodes de résolution classiques. Elles restreignent par exemple fortement la généralité des contraintes sémantiques que l'on peut évaluer ici en les cantonnant aux clauses de Horn [DEC86]. De même, la présence de quantificateurs existentiels dans les contraintes d'intégrité soulève des problèmes d'évaluation difficiles à maîtriser [CHA86]. La représentation et l'évaluation d'informations négatives est aussi un problème ouvert, bien que des solutions commencent à apparaître [NAQ86].

Une fois ces problèmes résolus, les limites pratiques de cette approche résideront pour l'essentiel dans la disponibilité de systèmes déductifs performants. Si le langage Prolog est aux dires mêmes de ses pères le "langage machine" de l'Intelligence Artificielle, il ne fait aucun doute que sa valeur opérationnelle va croissant. Autant parce que de nouvelles technologies en rendent l'exécution plus rapide [BER83], que parce qu'on peut s'appuyer sur des méthodes d'évaluation efficaces développées dans le domaine des bases de données [ULL85, ZAN85].

Dès lors que les problèmes d'efficacité et d'interface avec les SGBD seront résolus, et il ne fait aucun doute qu'ils le seront, le mariage des deux domaines aura des effets dévastateurs sur les méthodes traditionnelles de développement des applications informatiques.

La gestion de quantités importantes d'informations deviendra en effet transparente. Elle sera en particulier libérée des contraintes actuelles imposées aux systèmes de représentation des connaissances et aux systèmes déductifs par la gestion de mémoire virtuelle (somme toute limitée) des systèmes d'exploitation.

Enfin, la spécification et la modélisation du comportement des objets et des connaissances manipulées sera unifiée et pourra donc faire l'objet d'interfaces de très haut niveau.

Un effort significatif est fait dans cette direction avec le développement de nouveaux langages de programmation en logique intégrant ces différents aspects.

La démarche utilisée ici a été moins ambitieuse. On s'est borné à rapprocher un certain nombre de techniques venant d'une part des SGBD, d'autre part de l'Intelligence Artificielle. Les réalisations de prototypes qui en ont résulté démontrent sans contestation possible que leur usage conjoint est non seulement réalisable, mais exploitable de manière opérationnelle.

En illustrant la faisabilité de ces propositions sur des exemples inspirés de la CAO et de la bureautique, on a tenté de montrer en quoi elles répondent à des besoins concrets.

Par ailleurs, on s'est volontairement abstenu de citer à un moment quelconque le terme "système expert". Ce domaine était hors du cadre de cette étude. Néanmoins, les propositions formulées ici peuvent à l'évidence être utilisées pour le développement de grandes bases de connaissances et de bases de données intelligentes. C'est pourquo l'on a usé du vocable "Système de Base de Données Expert".

Un domaine particulièrement prometteur à cet égard est celui de la coopération de systèmes experts partageant des bases de données qui sont réparties logiquement ou physiquement.

Les mécanismes de modélisation de données généralisées et d'objets, ainsi que la méthode de certification proposés ici paraissent directement intégrables dans un tel environnement.

Il est impensable aujourd'hui pour les praticiens comme pour les chercheurs en informatique d'ignorer le domaine des Bases de Données ou de l'Intelligence Artificielle. Il est également probable que ces outils feront naître à terme de nouveaux besoins. Le rapprochement de leurs fonctionnalités apparaît comme une voie pleine de promesses. On a tenté ici, avec quelques techniques de base, de poser des jalons pour contribuer à un effort qui ne fait sans doute que commencer.

BIBLIOGRAPHIE

[ADI84] ADIBA M., NGUYEN G.T
Information processing for CAD/VLSI on a generalized data management system.
Proc. 10th International Conf. on Very Large Data Bases. Singapour. Aout 1984.

[ADI85] ADIBA M., NGUYEN G.T
Knowledge engineering for CAD/VLSI on a generalized data management system.
Knowledge Engineering in Computer-Aided Design. J.S Gero Ed.
North-Holland Publishing Co. 1985.

[ADI86] ADIBA M., NGUYEN G.T
Handling constraints and meta-data on a generalized data management system.
Expert Database Systems. L. Kerschberg Ed.
Benjamin/Cummings Publishing Co. 1986.

[BAN85] BANCILHON F.
A naive evaluation of recursively defined predicates.
MCC Report. Austin (Texas). 1985.

[BEN85] BENSAID A.
Un modèle relationnel étendu pour des bases de connaissances centrées objet.
Thèse de Docteur-Ingénieur. Institut National Polytechnique de Grenoble. Mai 1985.

[BER83] BERGER-SABBATEL G., IANESELLI J.C, NGUYEN G.T
A Prolog database machine.
Proc. 3rd International Workshop on Database Machines. Munich (RFA). Septembre 1983.

[BOK84] BOKSENBAUM C. et al.
Certification by intervals of timestamps in distributed database systems.
Proc. 10th International Conf. on Very Large Data Bases. Singapour. Aout 1984.

[BRO84] BRODIE M.
On the development of data models.
On conceptual modelling. Brodie et al. Ed. Springer-Verlag. 1984.

[BRO85] BRODIE M., MYLOPOULOS J.
AI and databases : semantic vs computational theories of information.
New directions for database systems.
Ablex Publishing Co. 1985.

[BRY86] BRY F., MANTHEY R.
Sur la validité des schémas de bases de données.
2e Journées Bases de Données Avancées.
Giens. Avril 1986.

[CAZ83] CAZIN J. et al.
Première maquette du système d' information de l' atelier CONCERTO. L' environnement F1.
Projet Concerto. CNET LAA. Rapport 7199. Novembre 1983.

[CET84] Projet National CAO-CFAO.
ACACIA : spécifications. Vol. 1 : Base de données.
Assigraph-Bull-Cert/Deri-Ensta. Toulouse. Décembre 1984.

[CHA86] CHAKRAVARTY U.S, MINKER J., GRANT J.
Semantic query optimization : additional constraints and control strategies.
Proc. 1st International Conf. on Expert Database Systems.
Charleston (South Carolina). Avril 1986.

[CHI84] CHIU et al.
Processing chained queries using semi-joins operations.
SIAM Journal of Computing. Avril 1984.

[COL83] COLMERAUER A.
Prolog en dix figures.
Séminaire CNET sur la programmation en logique.
Lannion. Mars 1983.

[COM85] Communications of the ACM.
Special Section on Architectures for Knowledge-Based Systems.
Vol. 28. No. 9. Septembre 1985.

[CRO84] CROMARTY A. et al.
Distributed database considerations in an expert system for radar analysis.
Proc. 1st International Workshop on Expert Database Systems.
Kiawah Island (South Carolina). Octobre 1984.

[DEC86] DECKER H.
Integrity enforcement on deductive databases.
Proc. 1st International Conf. on Expert Database Systems.
Charleston (South Carolina). Avril 1986.

[DEM82] DEMOLOMBE R.
Utilisation du calcul des prédicats comme langage d' interrogation des bases de données.
Thèse de Doctorat ès Sciences Mathématiques.
Université de Toulouse. 1982.

[DEM85] DEMOLOMBE R., FARINAS L.
Representation of incomplete information about structured objects.
Proc. Workshop on Knowledge-Based Management Systems.
Crète. Juin 1985.

[DEL82] DELOBEL C., ADIBA M.
Bases de données et systèmes relationnels.
Dunod Ed. Paris 1982.

[DEL85] DELOBEL C.
Formal aspects in databases.
Proc. International Conf. on Foundations of Data Organization.
Kyoto (Japon). Mai 1985.

[DEL86] DELOBEL C.
Bases de données et bases de connaissances : une approche systémique à l' aide d' une algèbre matricielle des relations.
8e Journéees Francophones sur l' Informatique.
Grenoble. Janvier 1986.

[EST84] ESTUBLIER J.
La base de programme ADELE. Manuel de présentation.
IMAG. Laboratoire de Génie Informatique.
Grenoble. Janvier 1984.

[FAU86] FAUVET M.C
CADB : un SGBD pour la CAO.
Rapport de Recherche TIGRE 33.
IMAG. Laboratoire de Génie Informatique. Mars 1986.

[FIN85] FINGER J.J, GENESERETH M.R
RESIDUE : a deductive approach to design synthesis.
Stanford Heuristic Programming Project. Report HPP 85-1.
Stanford University (California). Janvier 1985.

[GAL81] GALLAIRE H.
Impacts of logic on databases.
Proc. 7th International Conf. on Very Large Data Bases.
Cannes. Septembre 1981.

[GAL84] GALLAIRE H., MINKER J., NICOLAS J.M
Logic and databases : a deductive approach.
ACM Computing Surveys. Vol 16, no 2. Juin 1984.

[GAR82] GARDARIN G.
Bases de données. Les systèmes et leurs langages.
Eyrolles Ed. Paris 1982.

[GAR85] GARDARIN et al.
SABRINA : une évolution du SGBD SABRE vers un système intelligent et actif.
Journées IMAG(LGI)-INRIA : Bases de Données Avancées.
St-Pierre-de-Chartreuse. Mars 1985.

[GAR86] GARDARIN G., C. DE MAINDREVILLE
Evaluation des programmes logiques récursifs par des fonctions récurrentes.
2e Journées Bases de Données Avancées.
Giens. Avril 1986.

[GEL84] GELENBE E.
Incomplete representations of information in databases.
New Applications of Databases. Gardarin, Gelenbe Eds.
Academic Press. 1984.

[GRA83] GRAY M.
Databases for computer-aided design.
Proc. ICOD Workshop on New Applications of Databases.
Cambridge (G-B). Septembre 1983.

[HOR84] HORSTMAN P., STABLER E.
Computer-aided design using logic programming.
Proc. 21st Design Automation Conference.
Albuquerque (New Mexico). Mai 1984.

[HOW84] HOWARD H.
Interfacing databases and knowledge-based systems for structural engineering applications.
PhD Thesis. Carnegie-Mellon University.
Pittsburgh (Pensylvania). Avril 1984.

[HUL85] HULL R.
A survey of research on semantic database models.
Proc. International Conf. on Foundations of Data Organization.
Kyoto (Japon). Mai 1985.

[ISR84] ISRAEL D., BRACHMAN R.
Some remarks on the semantics of representation languages.
On conceptual modelling. Brodie et al. Ed. Springer-Verlag. 1984.

[JUL86] JULLIEN Ch., LEBLOND A.
A database interface for an integrated CAD system.
Proc. 23rd ACM/IEEE Design Automation Conference.
Las Vegas (Nevada). Juin 1986.

[KAT83] KATZ R.
Transaction management in the design environment.
Proc. ICOD Workshop on New Applications of Databases.
Cambridge (G-B). Septembre 1983.

[KAT83b] KATZ R.
Managing the chip design database.
Computer. Décembre 1983

[KEB86] KEBATCHI M.A, BERZINS V.
Component aggregation : a mechanism for organizing efficient engineering databases.
Proc. 2nd International Conf. on Data Engineering.
Los Angeles (Ca.). Février 1986.

[KIM84] KIM W. et al.
A transaction mechanism for engineering design databases.
Proc. 10th International Conf. on Very Large Data Bases.
Singapour. Aout 1984.

[KIM85] KIM W., BATORY D.
A model and storage technique for versions of VLSI CAD objects.
Proc. International Conf. on Foundations of Data Organization.
Kyoto (Japon). Mai 1985.

[KLE71] KLEENE W.
Logique mathématique.
Armand Colin Ed. Paris. 1971.

[KOW79] KOWALSKI R.
Logic for problem solving.
North-Holland Publishing Co. 1979.

[KRA80] KRATZER A.M
An analysis of the load levelling problem.
PhD Thesis. Cornell University.
Ithaca (New York). Septembre 1980.

[KUP85] KUPER G.M
The Logical Data Model : a new approach to database logic.
PhD Thesis. Stanford University. Stanford, Septembre 1985.

[LAU86] LAURENT D.
Information incomplète explicite dans le modèle partitionnel de base de données.
2e Journées Bases de Données Avancées.
Giens. Avril 1986.

[LEC84] LECOURVOISIER J.
CASSIOPEE : un système intégré pour la CAO de VLSI.
L'écho des recherches. No 118. 4e trimestre 1984.

[LEO83] Mc LEOD D. et al.
An approach to information management for CAD/VLSI applications.
ACM-IEEE Database week.
San Jose (California). Mai 1983.

[LIE81] LIEN Y. et al.
DSIS : a database system with interelational semantics.
Proc 7th International Conf. on Very Large Data Bases.
Cannes. Septembre 1981.

[LOR84] LORIE R. et al.
User interface and accèss techniques for engineering databases.
IBM Research Lab. RJ 4155. San-Jose (California). Janvier 1984.

[MIC83] NGUYEN G.T et al.
MICROBE : un système de gestion de bases de données relationnelles.
Journées Conception, Implantation et Utilisation de SGBD relationnels sur micro-ordinateurs.
Université Paul Sabatier. Toulouse. Février 1983.

[MIS84] MISSIKOFF M., WIEDERHOLD G.
Towards a unified approach to expert and database systems.
Proc. 1st International Workshop on Expert Databases Systems.
Kiawah Island (South Carolina). Octobre 1984.

[MIT85] MITCHELL T.M et al.
A knowledge-based approach to design.
AI/VLSI Project. Report LCSR-TR-65.
Rutgers University. New Brunswick (New Jersey). Janvier 1985.

[MOR83] MORGENSTERN M.
Active databases as a paradigm for enhanced computing environment.
Proc. 9th International Conf. on Very Large Data Bases.
Florence (Italie). Septembre 1983.

[NAQ86] NAQVI S.A
Negative queries in Horn databases.
Proc. 1st International Conf. on Expert Database Systems.
Charleston (South Carolina). Avril 1986.

[NIC78] NICOLAS J.M, GALLAIRE H.
Databases : theory vs. interpretation.
Logic and Databases. Gallaire, Minker Eds.
Plenum Press 1978.

[NIC79] NICOLAS J.M
Contribution à l' étude théorique des bases de données.
Apport de la logique mathématique.
Thèse de Doctorat ès Sciences Mathématiques.
Université de Toulouse. 1979.

[NIC82] NICOLAS J.M
Logic for improving integrity checking in relational databases.
Acta Informatica. No. 18. 1982.

[NGU79] NGUYEN G.T
A unified method for query decomposition and shared information updating in distributed systems.
Proc. 1st International Conf. on Distributed Computing Systems.
Huntsville (Alabama). Octobre 1979.

[NGU80] NGUYEN G.T
Decentralized dynamic query decomposition for distributed database systems.
Proc. ACM Pacific Conference.
San Francisco (California). Novembre 1980.

[NGU81] NGUYEN G.T
Distributed query management for a local network database system.
Proc. 2nd International Conf. on Distributed Computing Systems.
Paris. Avril 1981.

[NGU81b] NGUYEN G.T, SERGEANT G.
Distributed architecture and decentralized control for a local network database system.
ACM International Computing Symposium.
Londres (G-B). Mars 1981.

[NGU81c] NGUYEN G.T, SERGEANT G.
Local network database.
Computer Communications. Vol 4, no 4. Aout 1981.

[NGU82] NGUYEN G.T et al.
A high level interface for a local network database system.
Proc. IEEE INFOCOM Conference.
Las Vegas (Nevada). Mars 1982.

[NGU82b] NGUYEN G.T
Distributed query processing at Laboratoire IMAG.
Database Engineering. IEEE. Vol 5, no 3. Septembre 1982.

[NGU83] NGUYEN G.T, OLIVARES J., WINNINGER P.
Programmation en logique pour une base de données généralisées.
Rapport de Recherche TIGRE 7.
IMAG. Laboratoire de Génie Informatique. Novembre 1983.

[NGU84] NGUYEN G.T, OLIVARES J., WINNINGER P.
Coopération de Prolog et d' un SGBD généralisé : principes et applications.
Séminaire CNET sur la programmation en logique.
Perros-Guirec. Avril 1984.

[NGU84b] NGUYEN G.T, WINNINGER P.
Prolog et bases de données relationnelles.
Rapport de Recherche TIGRE 25.
IMAG. Laboratoire de Génie Informatique. Décembre 1984.

[NGU85] NGUYEN G.T, OLIVARES J.
Semantic data organization on a generalized data management system.
Proc. International Conf. on Foundations of Data Organization.
Kyoto (Japon). Mai 1985.

[NGU85a] NGUYEN G.T, OLIVARES J.
SYCSLOG : système logique d' intégrité sémantique.
Rapport de Recherche TIGRE 26.
IMAG. Laboratoire de Génie Informatique. Janvier 1985.

[NGU86a] NGUYEN G.T
Semantic data engineering for generalized databases.
Proc. 2nd International Conf. on Data Engineering.
Los Angeles (California). Février 1986.

[NGU86b] NGUYEN G.T
Object prototypes and database samples for expert database systems.
Proc. 1st International Conf. on Expert Database Systems.
Charleston (South Carolina). Avril 1986.

[NGU87] NGUYEN G.T
Logical prototypes for database objects.
Soumis à publication.

[OLI86] OLIVARES J.
Thèse de Doctorat. Institut National Polytechnique de Grenoble. Juin 1986.

[PAR83] PARSAYE K.
Database management, knowledge base management and expert system development in Prolog.
Proc ACM-IEEE Database Week. San-Jose (California). May 1983.

[REH84] REHAK D., HOWARD C., SRIRAM D.
An integrated knowledge-based structural engineering environment.
Proc. IFIP Working Conf. on Knowledge Engineering in Computer-Aided Design.
Budapest (Hongrie). Septembre 1984.

[REI84] REITER R.
Towards a logical reconstruction of relational database theory.
On conceptual modelling. Brodie et al. Ed. Springer-Verlag. 1984.

[REU86] REUTER A.
Load control and load balancing in a shared database management system.
Proc. 2nd International Conf. on Data Engineering.
Los Angeles (California). Février 1986.

[RIE85] RIEU D.
Modèle et fonctionnalités d'un SGBD pour les applications CAO.
Thèse de Doctorat. Institut National Polytechnique de Grenoble. Juillet 1985.

[RIE86a] RIEU D.
Nature, état et dynamicité de l'objet CAO.
2e Journées Bases de Données Avancées.
Giens. Avril 1986.

[RIE86b] RIEU D., NGUYEN G.T
Semantics of CAD objects for generalized databases.
Proc. 23rd ACM/IEEE Design Automation Conference.
Las Vegas (Nevada). Juin 1986.

[RIE87] RIEU D., NGUYEN G.T
An expert database system for computer-aided design.
Soumis à publication.

[SAK83] SAKAROVITCH M.
Techniques mathématiques de la recherche opérationnelle.
Vol. IV : optimisation combinatoire.
Université de Grenoble. Mai 1983.

[SCH85] SCHOLL M.
Architecture matérielle pour le filtrage dans les bases de données.
Thèse de Doctorat ès Sciences Mathématiques.
Institut National Polytechnique de Grenoble. Juin 1985.

[SCI84] SCIORE E., WARREN D.
Towards an integrated database-Prolog system.
Proc. 1st International Workshop on Expert Database System.
Kiawah Island (South Carolina). Octobre 1984.

[SIM84] SIMON E., VALDURIEZ P.
Design and implementation of an extendible integrity subsystem.
ACM SIGMOD Conference. Boston (Mass.). Juin 1984.

[SMA80] GOLDBERG A., ROBSON D.
Smalltalk 80 : the language and its implementation.
Addison-Wesley Publ. Co. 1983.

[SPY84] SPYRATOS N.
The partition model : a deductive database model.
INRIA. Rapport de Recherche no. 286. Avril 1984.

[STO85] STONEBRAKER M.
Thoughts on new and extended data models.
Journées IMAG(LGI)-INRIA : Bases de Données Avancées.
St-Pierre-de-Chartreuse. Mars 1985.

[TIG83] Présentation générale du projet TIGRE.
Rapport de Recherche TIGRE 1.
IMAG. Laboratoire de Génie Informatique. Janvier 1983.

[TSI85] TSICHRITZIS D.
Les outils bureautiques.
Journées IMAG(LGI)-INRIA : Bases de Données Avancées.
St-Pierre-de-Chartreuse. Mars 1985.

[ULL85] ULLMAN J.D
Implementation of logical query languages for databases.
Proc. ACM-SIGMOD International Conf. on Management of Data.
Austin (Texas). Mai 1985.

[VAS83] VASSILIOU Y. et al.
How dœs an expert system get its data ?
Proc. 9th International Conf. on Very Large Data Bases.
Florence (Italie). Septembre 1983.

[WAL83] WALKER A.
Databases, expert systems and Prolog.
IBM Research Lab. RJ 3870. San-Jose (California). Avril 1983.

[WAR81] WARREN D.
Efficient processing of interactive relational queries expressed in logic.
Proc. 7th International Conf. on Very Large Data Bases.
Cannes. Septembre 1981.

[WIE84] WIEDERHOLD G. et al.
Knowledge-based management systems.
Knowledge-based management systems project.
Stanford University (California). Mai 1984.

[WOO83] WOODFILL J., STONEBRAKER M.
An implementation of hypothetical relations.
Proc. 9th International Conf. on Very Large Data Bases.
Florence (Italie). Septembre 1983.

[YAZ85] YAZDANIAN K.
Mise en œuvre simultanée des règles de cohérence et de déduction dans une base de données.
Journées IMAG(LGI)-INRIA : Bases de Données Avancées.
St-Pierre-de-Chareuse. Mars 1985.

[YU86] YU C., et al.
Adaptive techniques for distributed query optimization.
Proc 2nd International Conf. on Data Engineering.
Los Angeles (Ca.). Février 1986.

[ZAN84] ZANIOLO C.
Object-oriented programming in Prolog.
Proc. Logic Programming Symposium.
Atlantic City (New Jersey). Fevrier 1984.

[ZAN85] ZANIOLO C.
The representation and deductive retrieval of complex objects.
Proc. 11th International Conf. on Very Large Data Bases.
Stockholm (Suède). Aout 1985.

[ZAU83] ZAUMEN W.
Computer assisted circuit evaluation in Prolog for VLSI.
ACM-IEEE Database Week. San-Jose (California). Mai 1983.

Composition réalisée sur le VAX 11/780 du Laboratoire de Génie Informatique à l' IMAG.

DERNIERE PAGE DE LA THESE

AUTORISATION DE SOUTENANCE

DOCTORAT D'ETAT

Vu les dispositions de l'article 5 de l'arrêté du 16 avril 1974,

Vu les rapports de Mr. C. Delobel.....

M. G. Gandarun.....

M. J. Nicolas.....

M. Nguyen Gia Toan..... est autorisé à présenter une thèse en vue de l'obtention du grade de DOCTEUR D'ETAT ES SCIENCES.

Fait à Grenoble, le 13 MAI 1986

Transmis avec avis favorable
C. Delobel
Président de la Commission des Thèses 24e Section
le 12 05 85

Le Président de l'U.S.M.G.

UNIVERSITE SCIENTIFIQUE ET MEDICALE DE GRENOBLE