Types abstraits et bases de données : formalisation de la notion de partage et analyse statique des contraintes d'intégrité

Ana Maria Sales

THESE

présentée à

l' Université Scientifique et Médicale de Grenoble

pour obtenir le grade de
DOCTEUR INGENIEUR
«Informatique»

par

Ana Maria SALES

TYPES ABSTRAITS ET BASES DE DONNEES

FORMALISATION DE LA NOTION DE PARTAGE ET ANALYSE STATIQUE DES CONTRAINTES D'INTEGRITE

Thèse soutenue le 24 avril 1984 devant la commission d'examen

C. DELOBEL	Président
D. BERT	
J.P. FINANCE	Examinateurs
Ph. JORRAND	
J.M. NICOLAS	

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

Remerciements

Je tiens à remercier tous ceux qui matériellement et moralement m'ont permis de mener à bien ce travail. En particulier, je remercie Mr Rui Barbosa pour la confiance qu'il a déposée en moi et pour le virus "recherche" qu'il m'a transmis. Je remercie également la direction du département d'informatique de l'Universidade Nova de Lisboa pour son appui et sa compréhension.

Je remercie Mr Philippe Jorand de m'avoir accueilli dans son équipe, de m'avoir proposé une direction de travail et pour ses conseils; Mr Didier Bert pour avoir accepté de suivre mon travail, pour ses conseils et pour son aide à la mise en forme de ce document.

Je suis aussi très reconaissante à Mr C. Delobel D. Bert, J-P. Finance, Ph. Jorand et J-M. Nicolas qui on bien voulu juger mon travail. J'ai apprécié leurs critiques et les ouvertures qu'ils m'ont proposées.

Je remercie Mirella Bello pour son "agréable" aide à la frappe de ce document.

Je tiens à remercier particulièrement Mr Jean-Pierre Finance pour l'intérêt qu'il a porté à mes idées, pour son soutien et sa compréhension

Ana Sofia Simonet

T A B L E D E S M A T I E R E S

0 - INTRODUCTION

0.0 - INTRODUCTION

Le concept de base de données est apparu vers les années 62/63. Les difficultés rencontrées dans la gestion des données d'un univers à l'aide de plusieurs fichiers sont une des principales causes qui ont favorisé l'émergence de ce concept. En effet, l'utilisation d'un fichier soulève le problème de l'interdépendance totale entre celui-ci et les programmes qui l'exploitent; dans le cas de plusieurs fichiers, à ce problème s'ajoutent ceux résultant de la multidéfinition possible d'une même donnée et sa multilocalisation qui en est une conséquence.

Lors d'une manipulation des données, des problèmes supplémentaires apparaissent. Ainsi, l'actualisation d'une donnée exige l'actualisation de n fichiers, n étant le nombre de fichiers où la donnée est susceptible d'être présente. La modification des besoins d'un utilisateur peut avoir comme conséquence la modification de la définition d'un fichier: tous les programmes qui l'exploitent doivent être modifiés. Enfin, l'évolution de l'univers lui-même rend nécessaire la modification des définitions de certains fichiers; plusieurs programmes doivent être modifiés par la même occasion.

Pour essayer de résoudre ces difficultés, les bases de données utilisent le principe fondamental de la séparation entre la spécification d'un univers et les valeurs qui sont stockées en mémoire; ce principe introduit un nouveau niveau d'abstraction dans la gestion d'un univers. Ainsi, toute base de données est composée par:

- une spécification de l'univers selon un modèle de données. Dans cette spécification, dénommée SCHEMA, seules sont décrites les propriétés qui ont un intérêt pour les utilisateurs;

- une collection de valeurs qui représente les données de l'univers restreint par le schéma. Cette collection de valeurs, abusivement considérée en elle-même comme la base de données, n'est accessible qu'à travers un schéma.

Les propriétés spécifiées dans un schéma sont des objets, les règles auxquelles ces objets sont soumis, et des relations entre ces objets.

Comme un schéma n'est qu'une abstraction d'un univers [BRO78], la spécification d'un schéma, de la même manière que la spécification de n'importe quelle abstraction, exige l'"intime conviction" qu'il représente bien l'univers présumé. Si, dans un cas général, ceci pose déjà problème ("représente bien" n'est pas défini de façon totale et unique), dans le cas particulier des bases de données, la difficulté est augmentée. En effet, une base de données sert plusieurs catégories d'utilisateurs ayant chacune sa propre façon de voir l'univers, ce qui peut conduire à plusieurs abstractions différentes.

Par ailleurs, si dans la plupart des cas la spécification d'un objet (ses attributs) ne pose plus de problème à l'heure actuelle, la spécification des règles n'est faite que d'une manière incomplète et souvent "ad hoc": les règles sont soit dispersées dans les programmes des utilisateurs (et non pas dans le schéma) [TSI76] [TAY76], soit centralisées au niveau du schéma sans toutefois que la preuve de leur non redondance et de leur cohérence soit effectuée [AST76]. Dans les deux cas la vérification qu'un objet satisfait les règles est totalement dynamique. Dans le premier cas, à ce problème s'ajoute celui de la garantie de cohérence de la base de données. Ceci est une conséquence de l'absence de mécanismes automatiques de vérification de cohérence de l'ensemble des règles définies par tous les utilisateurs dans tous leurs programmes.

La cause directe de cet état de choses est l'absence de consensus sur les catégories de règles ayant un intérêt pour les bases de données et sur l'endroit où elles doivent être spécifiées. Les causes indirectes sont éventuellement liées au pragmatisme initial dans les bases de données, et à la difficulté de spécification des règles. Ainsi, le pragmatisme a favorisé les grands thèmes de recherche suivants:

- La modélisation d'un univers. Sur ce point deux voies sont considérées:

a) Quelles caractéristiques doit avoir un modèle de données pour permettre une définition logique d'un univers. On voit naître les modèles réseau, hiérarchique, relationnel-binaire et n-aire, entity- relationship, etc.

b) Etant donné un SGBD fondé sur un modèle de données particulier,

et un univers à informatiser pour un ensemble d'utilisateurs, quel est le "meilleur" schéma possible.

- La confidentialité: étant donné un schéma et un ensemble d'utilisateurs, comment définir les droits d'accès pour chaque catégorie d'utilisateurs.

- La concurrence : c'est-à-dire la gestion efficace de l'accès simultané de la base de données par plusieurs catégories d'utilisateurs.

- La récupération d'une base de données, c'est-à-dire le rétablissement d'un état cohérent à partir d'un état incohérent. L'état incohérent peut être dû à une défaillance du système (matériel ou logiciel), ou à une défaillance du programme d'un utilisateur; quel que soit le cas, l'état cohérent recherché doit être obtenu par déduction de données coherentes à partir de données incohérentes.

Par ailleurs, parmi les difficultés rencontrées dans la spécification des règles, nous soulignons:

- La difficulté de transcription d'une règle informelle, normalement définie en langue naturelle, dans une règle formelle.

- La diversité des méthodes possibles de spécification des règles (formelles).

- L'inexistence de mécanismes automatiques de preuve de non-redondance (la redondance signifiant plusieurs spécifications d'une même règle) et de cohérence d'un ensemble de règles.

- L'inexistence de mécanismes automatiques qui, à partir d'un ensemble de règles, déduisent l'ensemble minimum de vérifications dynamiques à effectuer lors d'une modification des règles ou lors d'une modification des valeurs stockées en mémoire.

0.1 - LE PROBLEME

Comme tout domaine pragmatique, les Bases de Données ont évolué vers la "théorisation". Cette nouvelle approche a suscité de nouvelles orientations pour la recherche, parmi lesquelles nous soulignons:

- l'étude des langages presque naturels pour spécifier et manipuler les bases de données: ce thème a pour but l'ouverture des Bases de Données à des utilisateurs non informaticiens;

- la spécification d'univers complexes et évolutifs pour plusieurs catégories d'utilisateurs ayant chacune ses propres intérêts.

Notre travail se situe dans le deuxième volet. Etant donné que spécifier une entité signifie spécifier ses attributs et ses règles, et que la maximisation des vérifications statiques est un objectif commun à tous les domaines de l'informatique, nous considérons que les principales caractéristiques à exiger d'un langage de spécification adapté aux bases de données sont:

- la modularité, permettant entre autres l'étalement dans le temps de la spécification d'une catégorie d'objets;

- l'enrichissement aisé d'une spécification pré-existante;

- la vérification que l'on a spécifié toutes les propriétés dont on a besoin;

- l'analyse statique des règles spécifiées.

0.2 - NOTRE APPROCHE

"Il existe une relation d'inter-dépendance totale entre un modèle de données et son langage de spécification/manipulation" [DEL78]. Ainsi, le langage associé à un modèle réseau est différent de celui d'un modèle relationnel et aussi de celui d'un modèle "Entity-relationship", etc.

Avant de présenter notre approche du langage, nous allons présenter

le modèle de données qui lui est sous-jacent. En effet, notre but initial - un langage de spécification pour le modèle relationnel - a été dépassé très vite. Parmi les causes directes de cette évolution, nous soulignons l'utilisation du concept fondamental de type abstrait, ainsi que, la répercussion de ce concept sur un modèle de données [GOI80] [MOI82] [SCH77] [LIS74].

Seuls les univers acceptant une approche ensembliste peuvent être modélisés à l'aide du modèle proposé. Dans ce cas, nous avons considéré qu'un univers peut être modélisé par une famille d'ensembles d'objets, tel que:

- tous les objets d'un ensemble ont un même type,

- les types des éléments de plusieurs ensembles peuvent être les mêmes,

- des associations peuvent être établies entre un ou plusieurs types d'objets.

La principale conséquence des deux dernières conditions, est le fait qu'un même objet puisse appartenir à plusieurs ensembles: dans ce cas, la cohérence de l'information associée à l'objet doit être garantie.

D'une façon semblable à celle de [CHE76], nous avons choisi de représenter les attributs d'un objet par des fonctions. D'autre part, nous avons décidé d'utiliser la structure algébrique, comme structure mathématique de base du langage de spécification. Cette structure est caractérisée par:

- un ensemble S de noms de domaines ou noms de types,

- un ensemble F d'opérateurs à paramètres et à résultat dans S,

- un ensemble d'équations E qui définissent le sens des opérateurs.

Le besoin de spécifications modulaires (cf 0.1), nous a dirigés vers l'utilisation de la spécification algébrique des types abstraits proposée par [GUT78]. Ainsi, dans un environnement de progammation [BER83],

spécifier un nouveau type abstrait équivaut à spécifier une algèbre, en général hétérogène. L'algèbre spécifiée est associée à s ∈ S,où s est le type d'intérêt de la spécification. Puisqu'elle dépend intrinsèquement de l'environnement de programmation, et plus précisément de ses domaines (sorts) et de ses opérateurs, des algèbres différentes sont associées à un même type abstrait dans des environnements différents, pourvu qu'ils soient tous obtenus par des enrichissements d'un environnement initial. L'algèbre associée à l'environnement initial est l'algèbre minimale de la famille d'algèbres associées au type.

Notre choix de représenter les attributs d'un objet par des fonctions impose que les types des codomaines des fonctions attribut appartiennent, eux-aussi, à S. D'autre part, parmi les équations d'un type abstrait, nous soulignons celles qui représentent les catégories particulière des règles qui sont les Dépendances Inter valeurs d'Attributs (DIA) (cf. chap. III) et les Dépendances Inter Objets (cf. chap.VI). En sachant qu'un objet est communément perçu de n façons différentes, nous concluons que la spécification d'un univers revient à spécifier :

1 - ses objets significatifs à l'aide des types abstraits. Pour chaque famille d'objets, un type abstrait doit être spécifié. Les différentes vues selon lesquelles on doit percevoir les objets d'une sorte sont spécifiés par des enrichissements successifs d'un environnement initial, c'est-à-dire environnement auquel est associée l'algèbre minimale du type abstrait;

2 - les ensembles manipulables par des utilisateurs. La spécification de ces ensembles est faite par instantiation du type abstrait générique ens [i].

0.3 - TRAVAIL EFFECTUE

Dans ce paragraphe, nous faisons une présentation du travail effectué. Nous donnons donc un résumé, par chapitre, de notre travail, dont la ligne directrice est:

i - Etude de l'existant et propositions alternatives [Chapitre I]
ii - Solutions formelles [Chapitres II, III et IV]
iii - Solutions techniques [Chapitres III, IV et V].

CHAPITRE I

Ce chapitre est divisé en trois parties.

Dans la première partie [I-2], nous présentons les possibilités de spécification offertes par trois catégories de SGBDs, ainsi que les extensions souhaitables.

Dans la deuxième partie [I-3], nous faisons un rappel des notions de la théorie algébrique, dont nous avons directement besoin. Nous y présentons aussi les types génériques et l'enrichissement des types. Finalement, au paragraphe [I-4], nous présentons informellement les principales caractéristiques du modèle de données qui est sous-jacent à tout notre travail, et plus précisément les caractéristiques des "objets" ou "p-types".

CHAPITRE II

Nous formalisons le modèle proposé à l'aide de la théorie algébrique. Dans une première partie, nous définissons les notions d'"objet reel ", d'"objet" (ou de "p-type") et d'attribut. Ensuite, nous décrivons les concepts de clé, de type nucléaire et de vue d'un p-type. Puis nous proposons qu'un schéma soit défini par les types d'intérêt d'un univers. Enfin, nous décrivons la modification d'un schéma par enrichissement.

CHAPITRE III

Au chapitre III, nous étudions les règles qui définissent la dépendance entre les valeurs attributs d'une occurrence d'un p-type : nous les appelons des dépendances inter-attributs (abrégé DIAs). L'objectif du travail sur les DIAs est double : d'une part, aider l'utilisateur qui fait une spécification, en détectant les d-inconsistances et les multi-spécifications dans un ensemble de règles, et d'autre part, minimiser les vérifications dynamiques lors d'une modification de la base (insertion ou modification d'une occurrence du p-type), ou lors d'une modification de l'ensemble des DIAs du p-type. Le concept de base du chapitre est celui de sous-domaine stable. En effet, il permet de réécrire, dans un système de formules de la logique propositionnelle, les dépendances inter-attributs spécifiées initialement en logique du premier ordre. De plus, ce concept est à la base des notions de semi- domaine et de D-classe : toute occurrence d'un p-type appartient à une et une seule D-classe; selon que la D-classe est valide ou non-valide, l'occurrence est elle aussi valide ou non-valide. Finalement, nous proposons une méthode ad hoc pour détecter les d-inconsistances et redondances d'un ensemble de DIAs, et pour minimiser les vérifications dynamiques lors de la modification de la base ou de l'ensemble de DIAs. Cette méthode est basée sur un graphe dont les noeuds sont des sous-domaines stables généralisés et dont les arcs représentent les DIAs.

CHAPITRE IV

Dans ce chapitre, nous proposons que les dépendances inter-objet puissent être spécifiées au niveau d'un p-type qui contient d'autres p-types. La méthode consiste à associer, de façon automatique, une pré-condition à chaque opération primaire (insertion, modification, suppression) d'un p-type en association. Une transition d'états, déterminée par l'application d'une de ces fonctions, est valide ssi sa pré-condition est satisfaite.

CHAPITRE V

Le chapitre V présente des solutions techniques aux propositions faites en chapitre II. Il est divisé en trois parties. Dans la première [V-1], nous proposons un environnement de programmation minimal capable d'accepter la spécification de p-types et d'ensembles de p-types. Dans une deuxième partie [V-2], nous proposons une représentation abstraite du support commun aux différents algèbres d'un p-type. Nous proposons aussi une représentation abstraite des ensembles manipulables par les utilisateurs. Enfin en [V-3], nous proposons le schéma d'une base de données ad hoc chargée de la gestion des noms significatifs d'une base de données particulière.

I - BASES DE DONNEES ET TYPES ABSTRAITS

I.0 - INTRODUCTION

En [CHE76] il est souligné qu'un des principaux besoins de tout utilisateur des bases de données est de pouvoir spécifier et manipuler sa vision logique d'un univers. Parmi les obstacles les plus importants à la réalisation de cet objectif nous soulignons l'apparente diversité des concepts nécessaires pour réaliser la dépendance intrinsèque entre un utilisateur - sa formation culturelle et technique, son expérience antérieure, etc. - et sa vision logique, la multiplicité des modèles de données, ainsi que la dispersion des concepts du modèle à l'intérieur de ceux du langage informatique.

De plus, en sachant qu'une même application est susceptible d'être définie par plusieurs schémas équivalents [BEE80], nous avons décidé, dans un premier temps, de cerner les concepts fondamentaux qui permettent à la majorité des utilisateurs d'exprimer leur vision logique. Cette première partie du travail a été réalisée à l'aide d'une étude systématique des principaux modèles existant dans les bases de données, ainsi qu'à l'aide du concept de type abstrait. Le résultat de ce travail est le modèle de données proposé. Bien qu'il n'ait pas été conçu comme une extension des modèles déjà existant, ses concepts de base contiennent ceux des modèles réseaux, relationnel et "entity-relationship": chacun de ces modèles peut être vu comme un cas particulier du modèle proposé. Nous le montrons à la fin de ce chapitre.

Auparavant, nous présentons les concepts que nous considérons comme fondamentaux dans les trois principaux modèles de données, ainsi que ceux de la spécification algébrique et de la généricité, puis les grandes lignes de notre modèle.

I.1 - MODELES DE DONNEES: CONCEPTS DE BASE POUR LA SPECIFICATION

Dans ce paragraphe, nous récapitulons certaines notions des modèles réseaux, relationnel et entity-relationship dont l'importance nous semble

essentielle pour la spécification d'un univers.

I.1.1 - Modèle réseaux

La grande majorité des SGBD(s) (Système(s) de Gestion des Bases de Données) réseaux aujourd'hui disponibles, satisfont les recommandations du CODASYL (Conférence of DATA System Language) et plus précisément celles du rapport de 1973 [ANS75] [TAY76]. C'est le cas du DMS 1100 (UNIVAC), DBMS (DEC) IDS (CII-HB), etc.

Dans ces systèmes, la description d'un univers se fait sur deux niveaux: le SCHEMA et les SOUS-SCHEMAS.

Au niveau du schéma, on fait la description globale des données de l'univers, ainsi que la description interne du modèle (interface avec les structures internes de la base de données).

Dans ces modèles, les deux concepts de base dans la description d'un univers sont ceux d'enregistrement et de "set".

Un enregistrement est une collection nommée de constituants; un constituant est lui-même une donnée nommable et typée. Les différentes occurrences d'un même enregistrement sont différenciables à travers leur clé. Une clé peut être externe (= constituant) ou interne (= emplacement mémoire de l'occurrence).

Un "set" est une relation nommable entre deux catégories d'enregistrements: l'une est le père (owner) du "set" et l'autre est le fils (member) du "set". Une généralisation de ce concept est faite en considérant plusieurs catégories d'enregistrements comme "fils" d'un "set". Une occurrence d'un "set" est une collection d'occurrences d'enregistrements dont une seule est de type père: un "set" implante donc

une relation de type 1 x N entre deux catégories d'enregistrements. L'implantation des relations de type N x M et de type récursif (le fils et le père sont d'un même type) est plus difficile, voire impossible.

Chaque catègorie d'utilisateurs accède à la base de données à travers son propre sous-schéma qui est composé des enregistrements et des "sets" du schéma auxquels elle a droit; en général, l'accès à un enregistrement ne signifie pas l'accès à tous ses constituants: le DBA - utilisateur privilégié qui spécifie et gère la base - a le pouvoir de décider de la protection à assurer aux constituants des enregistrements.

Dans les futurs SGBDs, satisfaisant donc les recommandations de 1978 [TSI78] , la description d'un enregistrement comportera en plus la description procédurale des contraintes d'intégrité que les occurrences de l'enregistrement doivent satisfaire, ainsi que la description des droits d'accès. Cette innovation est fondamentale même si elle n'est pas suffisante. En effet, elle met au même niveau la description des constituants et des contraintes d'intégrité, qui sont les deux composants de la même réalité: l'enregistrement. Néanmoins, cette innovation n'est pas complètement satisfaisante puisqu'elle n'implique pas une vérification de cohérence de l'ensemble des procédures.

I.1.2 - Modèle relationnel

Le concept qui est à la base du modèle relationnel [COD70], [COD79] est celui de relation. Formellement une relation R définie sur les ensembles E1, E2, ... En, non obligatoirement distincts, est un sous-ensemble du produit cartésien E1 x E2 x ... x En. Les ensembles E1, E2, ... En sont appelés domaines de la relation et n son degré.

La relation R est donc un ensemble de n-uplets ordonnés <e1, e2,... en> où e1 $\in$ E1, e2 $\in$ E2,... en $\in$ En. Presque toujours, on associe à R un prédicat qui caractérise les éléments valides de R; dans ce cas, l'appartenance d'un n-uplet <e1, ... en> à R n'est possible que si

le prédicat est satisfait par les valeurs e1, ..., en.

En général, on associe à chaque domaine un nom désignant son rôle dans la relation. Ce nom, couramment appelé attribut ou constituant, permet à l'utilisateur d'oublier l'ordre des domaines dans la définition de R, en ne se rappelant que des noms significatifs pour lui. Une clé est nécessaire pour différencier deux n-uplets. Elle peut être constituée d'un sous-ensemble des attributs (clé externe) ou cachée à l'utilisateur (clé interne).

Les principales caractéristiques du modèle relationnel - l'existence d'un concept de base unique, et son caractère fortement mathématique - sont à la fois des avantages et sources de problèmes. Ainsi, ces caractéristiques ont permis au modèle relationnel de servir de support au travail théorique sur la modélisation d'un univers et plus particulièrement sur les propriétés qui peuvent ou doivent être exigées des différentes relations pour garantir la cohérence de la base dans un contexte particulier [COD70] [FAG77] [DEL78] [BEE80] [FAG81]. Parmi les problèmes, nous soulignons:

- la difficulté presque générale de définir le prédicat d'une relation. L'inexistence de ce prédicat, qui permettrait de décider de la validité d'un n-uplet, rend difficile la garantie de cohérence d'une relation;

- la généralité du concept de relation, qui permet la définition de relations sans aucune signification sémantique pour l'utilisateur.

I.1.3 - Modèle entity-relationship

Le modèle "entity-relationship" [CHE76] [BEN81] [CAT83] a été proposé en 1976 par P. CHEN. La base mathématique de ce modèle est constituée par la théorie ensembliste et la théorie relationnelle. A notre avis, ce modèle est très intéressant, puisqu'il contient en embryons des notions que nous considérons aujourd'hui comme fondamentales. Ainsi, il contient des notions de:

A- valeurs et ensembles de valeurs. Exemple

DECLARE	VALUE-SET	REPRESENTATION	ALLOWABLE-VALUE
	NO-EMP	entier (4)	(0,2000)
	PRENOM	chaîne (8)	TOUS

Cette déclaration effective permet que l'on puisse explicitement exiger qu'une opération de l'algèbre relationnelle (division, jonction, etc...) ne puisse être exécutée que si les attributs sont définis sur un même ensemble de valeurs; d'autre part, elle contient le germe de la distinction entre une entité et une valeur: leur définitions séparées permet de supposer qu'il y a une différence entre elles. A notre avis, cette différence n'a pas été complètement mise en évidence.

B- représentation fonctionnelle des attributs.

Un attribut est une fonction qui a comme domaine un ensemble d'entités (ou ensemble d'associations entre entités) et comme codomaine un ensemble de valeurs ou le produit cartésien d'ensembles de valeurs

$$f : Ei \text{ ou } Ri \longrightarrow Vi \text{ ou } Vi1 \times Vi2 \ldots Vin$$

La représentation fonctionnelle des attributs d'une entité ou association entre entités permet de supposer l'idée de "valeur unique" associée à un attribut d'une entité. Cependant, les restrictions que le type du résultat de f soit un type simple ou un type composé sont la conséquence de l' implantation des ensembles-d'-entités (ou ensembles d'associations) par des relations n'acceptant comme domaines que des ensembles dont les éléments sont de type simple ou composé. Nous considérons que cette restriction sur f n'est ni nécessaire ni souhaitable. En effet, elle exclut le cas courant où le type de f est ensemble; par exemple, les "enfants" d'une personne sont représentables par une fonction attribut de type "ensemble [personne]".

D'autre part, le modèle E-R, comme son nom l'indique, permet la spécification de:

1 - entités et ensembles-d'-entités. Par exemple:

```
DECLARE REGULAR ENTITY RELATION employé /*declaration simultanée
                                          d'un ensemble et d'un type
                                          d'entité*/

        attribut/value-SET

        EMP-NO/EMP-NO  /*l'attribut peut avoir le même nom
                         que son codomaine*/
          .
          .
          .

        PRIMARY KEY

        EMP-NO /*la clé primaire peut être aussi une composition
                 de fonctions attributs ou encore une valeur interne */
```

Comme il est dit en [CHE76] plusieurs ensembles d'entités peuvent repérer le même "type" d'entités ; de plus, ces ensembles ne sont pas forcément disjoints. Cependant, comme la déclaration d'un ensemble (RELATION) est faite en simultanéité avec la déclaration du type de ses éléments, la spécification explicite que plusieurs ensembles ont un même type d'éléments est difficile.

2 - associations entre entités et ensembles d'associations entre entités. Par exemple:

```
DECLARE REGULAR-RELATIONSHIP-RELATION TRAVAILLEUR-DEPARTEMEN
T-VILLE

                                    /*déclaration de l'association des en-
                                      tités EMPLOYES-DEPARTEMENT-VILLE */
        ROLE/ENTITY-RELATION/PK/MAX-NO-OF-ENTITES

    TRAVALLEUR /EMPLOYE  /  EMP-NO/  m

        ATTRIBUTE/VALUE-SET

            /*attributs propre à l'association*/
```

Les mêmes remarques faites à propos de la déclaration simultanée de type et d'ensemble sont reproduisibles ici. Si la séparation entre entité et association est (peut être) au niveau conceptuel importante, son implantation, qui exige que l'on ne puisse associer que des entités, est très restrictive: elle interdit qu'une association soit elle-même partenaire d'une nouvelle association. De ce point de vue, le modèle relationnel est plus puissant: des relations, définissant des "objets", peuvent entrer en association avec d'autres objets; la relation qui implante cette association peut elle-même (par sa clé) appartenir à de nouvelles relations, etc.

I.1.4 - Conclusion

Le tableau ci-dessous résume ce paragraphe.

MODELE	PERMET DE DEFINIR	AVANTAGES	PROBLEMES
RESEAUX	ENREGISTREMENT : NOM CONSTITUANTS : NOM (=selecteur) TYPE : elementaire /vecteur /agregat CLE : externe /interne		
	FUTUR: CONTRAINTS D'INTEGRITE :procedures	communs a tous les utilisateurs	verifications dynamiques
	SET : ENRG PERE ENRG FILS ordre des fils independance des fils FIXED MANDATORY OPTIONAL INS. AUTOMATIQUE INS. MANUEL	relation de type 1xN facilement implantee et controlee	dificulte d' implanter des relations -NxM -recursives

RELATIONNEL	RELATION : NOM DOMAINE : NOM TYPE /*non decomposable */ CLE : externe /interne PREDICAT /*definissant les elements valides */	- concept unique -base mathematique	trop general n'imposant pas des limites qui garantissent la signification dans une definition	
E-R	ENSEMBLES-DE-VALEURS NOM TYPE PREDICAT ENSEMBLES-D-ENTITES NOM FONCTION ATTRIBUTS NOM CODOMAINE ensemble-de-valeurs /produit cartesian d' ensembles-d-valeurs */ ENSEMBLES-D-ASSOCIATIONS : NOM ENTITE : ROLE (=fonction) NB MAX (1 ou N) ATTRIBUTS : NOM CODOMAINE PREDICAT	-declaration explicitement distincte des ensembles-d-entiés -representation fonctionnelle des attributs -accepte des associations 1xM, MxN, MxPxQ,etc	-restriction aux types simples -declaration simultanee du type des elements et de l' ensemble lui-meme -declaration simultanee des types d' association et de l' ensemble-d-associations -choix de la cle automatique : composition des cles des entites; ce choix n'est pas toujours valable.	

I.2 - TYPES ABSTRAITS ET GENERICITE

Tout langage informatique destiné à la spécification et à la manipulation d'univers complexes et changeants doit avoir un nombre limité de concepts fondamentaux. Ces concepts doivent être simples et indépendants afin de faciliter l'apprentissage du langage et de permettre à l'utilisateur de se concentrer sur son problème et non sur des questions annexes "de langage". De plus, ces concepts doivent permettre de spécifier clairement et succinctement un problème, en ne considérant dans un premier temps que sa sémantique; les décisions d'implantation étant réservées pour une phase ultérieure.

Pour satisfaire cette dernière exigence, le concepteur du langage peut essayer d'y introduire toutes les primitives dont tous ses utilisateurs virtuels auront besoin, ou bien il peut fournir des mécanismes qui leur permettront de définir eux-mêmes leurs propres primitives. La première hypothèse n'est pas réaliste, puiqu'aucun concepteur ne peut prévoir a priori tous les besoins de tous les utilisateurs du langage. Par contre, la seconde approche est intéressante et son développement a donné naissance aux types abstraits.

Dans ce travail, nous nous intéressons exclusivement aux types abstraits algébriques [GUT78] [BUR81] [GOG77]. Notre choix est lié au fait que la spécification algébrique des types abstraits est constructive, c'est-à- dire qu'elle permet de spécifier des nouveaux types à partir de types pré-existants, ce qui n'est pas possible dans la spécification axiomatique [BER79].

- Types abstraits

Au départ, la notion de type abstrait prétendait combler le manque d'abstraction des données existant dans les langages. En effet, les possibilités de programmation modulaire, que les notions de procédure et de fonction ont permises, n'avaient pas leur équivalent pour les données: de la définition d'une donnée demandait le choix et la définition simultanés de la structure d'implémentation [JOR77] [LIS74].

Le concept de type abstrait est une extension (généralisation) de celui des types de base (entier, bool, car, etc) pré-définis dans les langages. De ce point de vue, l'utilisateur éventuel d'un type abstrait peut le considérer comme la définition des fonctions que lui permettent de manipuler (construire et sélectionner) les objets du type. Quand il travaille sur les entiers, avec l'opération d'addition, l'utilisateur ne s'occupe ni de l'implantation ou de la preuve de l'opération, ni de la représentation d'un objet de type entier. De la même manière, l'utilisateur d'un type abstrait n'a besoin de connaître que la définition formelle des fonctions auxquelles il a droit. Ainsi, diverses implantations et leurs preuves peuvent être fournies dans différents contextes sans que l'utilisateur en ait conscience. Cette caractéristique des types abstraits leur permet d'implanter directement et facilement le besoin d'indépendance entre un objet et sa représentation ("data-indépendence" physique).

Cependant pour la personne qui fait une spécification formelle, cette connaissance est insuffisante. En effet, même si des langages tels que CLEAR [BUR77] [BUR79b] et LPG [BER82] [BER83] [BER83b] peuvent être équipés avec des outils d'aide à la spécification [MUS80] [RAB82b] [BER82b], l'utilisation performante de toutes les potentialités d'une spécification algébrique exige la connaissance de la théorie mathématique qui lui sert de support [STA77] [GOG77] [BUR78] [BUR79b]. Cette connaissance est aussi souhaitable pour quiconque a besoin de faire des

spécifications. Jusqu'à la fin de ce paragraphe, nous présentons un rappel des définitions dont nous avons directement besoin. Les notations utilisées seront proches de celles de LPG, dans la mesure où c'est le langage dont les concepts nous ont le plus marqué. Ces concepts sont aussi présents en CLEAR, sous une forme différente.

DEFINITIONS (RAPPELS)

[def I-1]
Une signature est une paire $\Sigma = (S,F)$ où S est un ensemble de noms de domaines (ou noms de types) et F est un ensemble d'opérateurs à paramètres et résultats en S.

Une signature est une définition syntaxique; la sémantique (signification des domaines et des opérateurs) est une algèbre.

[def I-2]
Soit la signature $\Sigma = (S,F)$, une Σ-algèbre est la donnée d'un ensemble $A = \{As | s \in S\}$ et d'un ensemble FA d'applications telles que

$\forall f \in F: \quad f:s1 \times s2 \times \ldots \times sn \longrightarrow s$

alors $fA \in FA: \quad fA: \quad As1 \times \ldots \times Asn \longrightarrow As$

"A" est appelé l'ensemble "support " de l'algèbre

Plusieurs algèbres peuvent être associées à une même signature. D'une manière générale, on ne s'intéresse pas à l'ensemble de toutes les algèbres d'une signature, mais seulement à une de ses parties. La caractérisation de ce sous-ensemble est faite à l'aide d'un ensemble d'équations E sur les opérateurs de la signature.

[def I-3]
Une présentation de type est un triplet <S,F,E> où (S,F) est une signature et E est un ensemble d'équations qui sont les axiomes de la présentation.

Une Σ-équation "e" est un triplet <X,i1,i2> où

- X est un ensemble de variables x1, ..., Xn sur les domaines de **Σ**.

- i1 et i2 sont des termes, écrits à l'aide de X et appartenant à un même domaine s ∈ S de **Σ**= (S,F).

En général, on écrit:

$$\forall X \qquad i1 == i2$$

Ou plus simplement

$$i1 == i2$$

[def I-4]
Une Σ-algèbre A satisfait une présentation <Σ,E> ssi elle est un modèle de E.

L'algèbre A est un modèle de E si elle satisfait toutes les équations de E. De la même manière que [BUR79] [BUR81] nous notons E* l'ensemble des algèbres qui satisfont E; et nous notons E** l'ensemble de <u>toutes</u> les équations que toutes les algèbres de E* satisfont. L'ensemble E** peut être obtenue directement à partir de E par l'application d'un système d'inférence cohérent et complet qui engendre toutes les conséquences logiques de E.

[def I-5]
Une algèbre de termes est une algèbre dont les éléments de ses domaines sont des termes.

Chaque terme a un type défini; soit il est une constante de l'algèbre, soit il à la forme "opérateur (terme 1, ...,terme n)" où n est le nombre de paramètres de "opérateur". Dans ce dernier cas, le type du terme est celui du résultat d'"opérateur". D'autre part, deux termes sont égaux ssi ils sont équivalents par $\underline{=}$,où $\underline{=}$ est la relation de congruence engendrée par E.

La notation W_Σ est employée pour dénoter l'algèbre des termes d'une signature Σ, et la notation $W\Sigma,E$ pour celle d'une présentation $\langle\Sigma,E\rangle$. Or $W\Sigma,E = W_\Sigma/=\!\!E$.

[def I-6]
$W\Sigma,E$ est un modèle d'une présentation de type abstrait.

Dans la plupart des cas, il est le modèle choisi pour définir la sémantique du type abstrait. Ce choix est dû à son adaptation aisée aux preuves dont on a besoin.

On note $\overline{E}$ l'ensemble d'équations vraies dans $W\Sigma,E$. (Techniquement: $\overline{E}$ est la théorie associée à l'algèbre initiale de la catégorie des Σ-E-Algèbres).

Il existe un homomorphisme [BUR81] unique entre $W\Sigma,E$ et tout Σ-E-Algèbre:

$$h : W,E \longrightarrow A\langle\Sigma,E\rangle$$

L'image homomorphe de $W\Sigma,E$ selon h peut être une sous-algèbre de A.

[def I-7]
La Σ-E-algèbre A est une Σ-sous-algèbre de A' ssi A' est une $\mathfrak{E}$-E'-algèbre et il existe un morphisme de signature entre $\mathfrak{E}$ et $\mathfrak{E}$ ($\sigma: \Sigma \longrightarrow \Sigma'$).

[def I-8]
Un morphisme de signature σ entre une signature $\Sigma = (S,F)$ et une signature $\Sigma' = (S',F')$ est une paire de fonctions $\langle f,g\rangle$ telle que:

$$f : S \longrightarrow S''$$

g est une famille d'application:

$$gus : Fus \longrightarrow F'f^*(u)f(s)$$

où

- $u = s1 \times s2 \times \ldots \times sn$

- $f^*(u) = f(s1) \times f(s2) \times \ldots \times f(sn)$

On note $\varphi(s)$, $\sigma(u)$ et $\sigma(w)$ à la place de f(s), f*(u) et gus(w).
Par rapport à Σ, Σ' ($\sigma : \Sigma \longrightarrow \Sigma'$) peut contenir des domaines ou des opérateurs supplémentaires.

Etant donnés un morphisme de signature $\sigma : \Sigma \longrightarrow \Sigma'$ et une Σ'-algèbre A', on engendre une Σ-algèbre A, par:

- A chaque domaine s $\in$ S de $\Sigma = (S,F)$ est associé l'ensemble de A' lié à $\sigma(s) \in S'$ de $\Sigma' = (S',F')$.

- Les fonctions de la Σ-algèbre A liée aux opérateurs f $\in$ F de Σ sont les fonctions de la Σ'-algèbre associées aux opérateurs $\sigma(f) \in F'$ de Σ'.

[def I-9]
Soient les signatures $\Sigma = (S,F)$ et $\Sigma' = (S',F')$ et le morphisme $\sigma : \Sigma \longrightarrow \Sigma'$, on définit la fonction qui fait passer d'une Σ'-algèbre A' à une Σ-algèbre A, notée $\underline{\sigma}$, par:

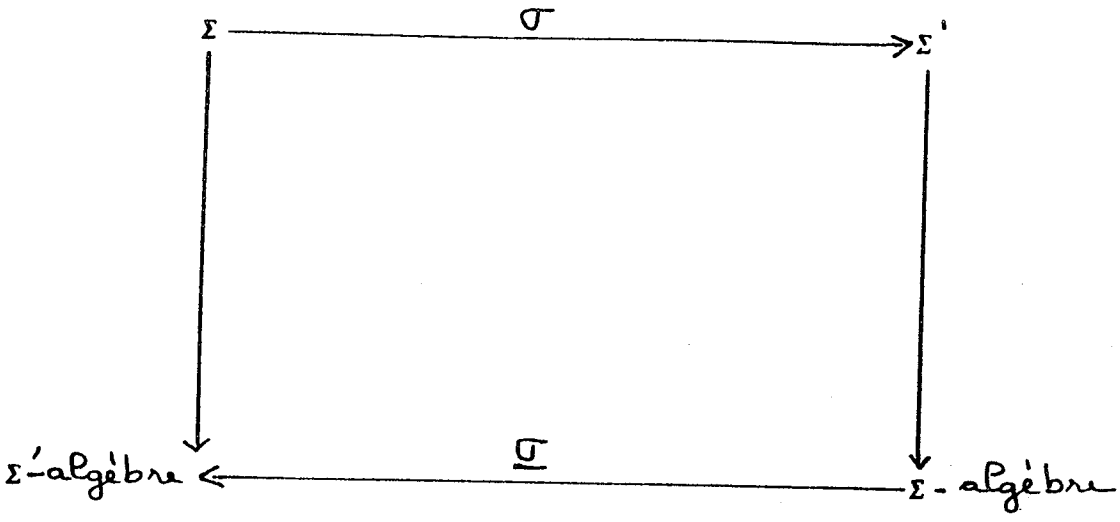
$$\underline{\sigma}(A') = A$$

où

- $|A|_s = |A'|_{\sigma(s)}$ /* la notation |X| est employée pour indiquer le support de l'algèbre X. */

- f(a1 ... an) de A est $\sigma(f)$(a1 ... an) en A'.

- On appelle A une <u>réduction</u> de A'



Un cas particulier de σ est le morphisme d'inclusion: les noms des domaines de Σ, ainsi que ceux de ses opérateurs sont conservés en Σ' (f et g sont des identités).

[def I-10] Soit les signatures $\Sigma=(S,F)$ et $\Sigma'=(S',F')$ et le morphisme σ: $\Sigma \dashrightarrow \Sigma'$, on definit la fonction:

$$\sigma^* : W\Sigma(X) \longrightarrow W\Sigma'(X')$$

où, X est un ensemble de S-variables, et X' est un ensemble de S'-variables telles que:

i- $\forall x \in X$, $\sigma^*(x) = f(x)$
avec f: X —> X' injective;

ii- $\forall f \in F$, f : —>s
$\sigma^*(f) = (f)$

iii- $\forall f \in F$ f: s1 x...x sn —>s
$\sigma^*(f(a1,\ldots,an)) = \sigma(f)(\sigma^*(a1), \ldots \sigma^*(an))$

Pour une Σ-équation e = ($\forall$ X) i1 = i2 on note $\sigma^*(e)$ pour la Σ'-equation: ($\forall$ X') σ^* (e1) = *(e2)

Contrairement à la spécification d'une présentation de type abstrait, où un nouveau type (le type d'intérêt et son algèbre, à un isomorphisme près) est spécifié, l'enrichissement d'une signature a comme unique conséquence la définition de nouvelles fonctions. Pour comprendre cette affirmation, il faut se rappeler que, formellement, tous les types d'un environnement de programmation appartiennent au support d'un type t, même si l'habitude fait que l'on ne nomme que les types dont on a besoin explicitement.

En général, un enrichissement a plusieurs types abstraits d'intérêt. Ces types sont implicitement déterminés par les types qui interviennent dans les nouvelles fonctions. Toutefois, si lors d'un enrichissement l'utilisateur ne prétend "enrichir" qu'un type bien déterminé, il a la possibilité de le nommer explicitement. Dans le contexte des bases de données, c'est surtout la dernière alternative qui nous intéresse.

[def I-11] Une présentation <S,F1,E1> enrichit un type t:<S,F,E>, ssi

i-il existe un morphisme d'inclusion σ entre $\Sigma=(S,F)$ et $\Sigma'=(S,F \cup F1)$;

ii- il existe, eventuellement, une restriction R sur les termes de W_Σ : $R(|W_\Sigma|) \subseteq |W_{\Sigma'}|$,où $|W|$ signifie l'ensemble support de l'algèbre W;

iii- toute assertion (équation) qui est vraie pour $R(|W_\Sigma|)$ est vraie dans $W\Sigma'$. Autrement dit: $\forall e \in \overline{E}$, e=<X,i1,i2> et i1,i2 $\in R(|W_\Sigma|)$ alors, $\sigma^*(e) \in \overline{E}'$, où E'=E UE1.

On appelle "enrichissement de vues", ou simplement "enrichissement", un morphisme d'inclusion qui satisfait (ii) et (iii).
L'enrichissement du type t:<Σ,E> pour obtenir le type t':<Σ',E'> induit les relations suivantes:

i-la présentation Σ est incluse en Σ', noté $\Sigma \subseteq \Sigma'$;

ii-la Σ-E-algèbre $W\Sigma,E$ inclut la Σ'-E'-algèbre $W\Sigma',E'$, en tant que Σ-E-algèbre, noté $\sigma(W\Sigma',E') \subseteq W\Sigma,E$

Généricité. Le besoin de généricité est fondamental: il est implanté dans tous les langages évolués sous une forme plus ou moins efficace [JAC78]. Dans le cadre d'une spécification algébrique, il est aussi important. En effet, si le langage permet la spécification de types abstraits génériques (ou paramétrés), quiconque a besoin de n types "semblables", peut les spécifier à l'aide d'un seul type: par exemple, au lieu de spécifier les types "ensemble-d'-entiers", "ensemble-de-personnes", "ensemble-de-cours", disposant chacun de fonctions semblables (l'insertion, la modification, etc), l'utilisateur peut spécifier uniquement le type ensemble [t]. Les fonctions d'insertion, de modification, etc du type ensemble [t] deviennent des fonctions génériques au sens structural. De plus, un de leurs paramètres est le type "ensemble [t]" lui-même. Cependant, le traitement statique d'une spécification générique implique que l'on puisse définir les caractéristiques qu'un type doit posséder pour être paramètre effectif du type générique [BER77]. Cette définition est faite à l'aide du concept de propriété.

RAPPELS [BUR77] [BER83]

[def I-12]
Une présentation de propriété est la donnée d'un triplet <S,F,E> où

- S est un ensemble de noms de domaines (ou noms de types);
- F est un ensemble d'opérateurs;
- E est un ensemble d'équations.

Dans S on distingue plusieurs domaines t1,..., tn qui sont appelés domaines d'intérêt de la propriété. D'autre part, il n'y a pas parmi les opérateurs de F des opérateurs générateurs : S et F sont considérés comme formels. Contrairement à une présentation de type abstrait où l'on définit l'algèbre des termes par son type d'intérêt, l'algèbre des termes d'une présentation est constituée par les éléments de ses domaines d'intérêt, t1,..., tn, considérés comme des constantes sur lesquelles on définit les opérateurs de F.

Plusieurs types en général non isomorphes peuvent satisfaire une même

propriété.

[def I-13]

Un type abstrait t décrit par une présentation $\langle \Sigma, E \rangle$ satisfait une propriété p décrite par la présentation $\langle \Sigma', E' \rangle$ ssi il existe un morphisme de signature entre le Σ' et Σ (c'est-à-dire $\sigma : \Sigma' \longrightarrow \Sigma$) et $E' \subseteq \overline{E}$, c'est-à-dire pour toute équation $e \in E'$ alors $\sigma^*(e) \in \overline{E}$)

Par ailleurs ,un type t satisfaisant une propriété p satisfait toute propriété p' incluse en p.

[def I-14]

Une propriété $p' = \langle \Sigma', E' \rangle$ est incluse en $p = \langle \Sigma, E \rangle$ ssi il existe un morphisme de signature entre Σ' et Σ (c'est-à-dire $\sigma : \Sigma' \longrightarrow \Sigma$) et $\forall e \in E' : \sigma^*(e) \in E^{**}$.

Ci dessous, nous présentons la façon utilisée en LPG pour établir la relation entre un type abstrait paramètré et les types qui peuvent être ses paramètres effectifs. Cette relation est établie à travers une propriété. Ainsi, on écrirait:

type t exige p [t1,..., tn opns fi...fj]

pour indiquer que seuls les types abstraits satisfaisant la propriété p peuvent être paramétres formels du type t. Lors de la spécification de t, ti,..., tn et fi,..., fj sont des types et des opérateurs formels.

Une instance de t est décrite, par exemple, comme:

t avec p [i1,... in opns g1... gj]

où encore plus simplement t[i1,..., in] si l'on peut déduire que (i1,..., in opnsg1...gj) est un modèle de p. Dans cette notation, i1,..., in et g1...g2 sont des types et des opérateurs effectifs qui satisfont p.

D'autre part, pour indiquer au système qu'un type satisfait une propriété p, l'utilisateur dispose, en LPG, des clauses :

i - satisfait p [t1,..., tn opns f1... fj]

ii - hérite p [t1,..., tn opns f1... fj]

La spécification d'une de ses clauses lors d'une spécification de type t donne au système l'indication que t satisfait, par l'intermédiaire de ses fonctions f1... fj, la propriété p. La différence entre i) et ii) étant dans l'ensemble d'équations de t. Ainsi, tandis que dans le premier cas (satisfait) l'utilisateur, ou un système de déduction, a fait la preuve de satisfaisabilité, (cf [CHA73]), dans le deuxième cas, le sytème prend les équations de p et les ajoute aux équations spécifiées pour t.

Ces deux mêmes clauses sont aussi employées pour indiquer au système qu'une propriété inclut une autre. Ainsi, lors de la spécification de la propriété p on pourrait "écrire":

i - satisfait p' [.....]

ii - hérite p' [.....]

pour indiquer que p' est incluse en p. La différence entre la première et deuxième clause étant la même que dans le cas des types.

I.3 - MODELE DE DONNEES BASE SUR LES TYPES ABSTRAITS

Dans ce paragraphe, nous présentons d'une façon intuitive les principaux points du modèle que nous proposons. Ce modèle n'est que la conséquence dans le contexte des bases de données de l'application du concepts d'abstraction à la spécification d'un univers. Son principe de

base est le principe d'IMAGE:

"Tout utilisateur doit pouvoir spécifier des objets tels que chaque occurrence d'une spécification soit l'IMAGE d'un "objet réel", c'est-à-dire d'un objet de l'univers", selon sa perception logique.

Cette notion de spécification dont les occurrences sont des images d'"objets réels" est essentielle. Son application systématique à différents univers nous a permis d'extraire trois caractéristiques des objets dans le contexte des bases de données:

- Un même "objet" peut être perçu selon plusieurs facettes. Par exemple une personne p1 peut être perçue comme "salariée", et/ou "élève" et/ou "parent" et/ou "propriétaire", etc. C'est la facette à travers laquelle on a besoin de percevoir l'objet qui détermine les caractéristiques (attributs) dont la connaissance est fondamentale : la connaissance d'autres caractéristiques de l'objet étant sans importance, voire préjudiciable ou interdite. Ainsi, si la connaissance du salaire de p1 est importante quand la facette significative est "salarié", sa connaissance est sans importance pour la facette "élève".

- Un même objet peut appartenir à plusieurs ensembles. Ainsi, la personne p1 peut appartenir à l'ensemble des salariés de la ville de Grenoble, à l'ensemble des élèves de la faculté X, à l'ensemble des étudiants internationaux, etc. Pour chacun des ensembles, il n'y a qu'une facette significative; plusieurs ensembles peuvent être concernés par la même facette.

- Un objet peut être associé avec d'autres objets de plusieurs

manières différentes. Ainsi, la personne p1 peut être propriétaire de la maison m1, et/ou professeur de la personne p2 pour le cours c1, et/ou marié avec p4, etc.

Notre deuxième principe, le principe d'UNICITE découle de ces trois propriétés spécifiques des objets dans le contexte des bases de données. Son énoncé est:

"A toute caractéristique (attribut) d'un "objet réel" est associée une valeur unique, indépendante du nombre de facettes selon lesquelles on peut le percevoir, du nombre d'ensembles auxquels il appartient, ainsi que du nombre d'associations qu'il établit avec d'autres "objets réels".

L'implantation de ce principe permet à l'utilisateur de raisonner comme si l'ordinateur ne contenait qu'une représentation unique de chacun des "objets réels" de l'univers. Ainsi, tous les utilisateurs ayant droit à l'attribut ATTRj d'un "objet", ont la connaissance d'une même valeur pour chacune des occurrences ("objet réel") de l'objet; en conséquence, une correction faite par un utilisateur sur la valeur d'un attribut d'un "objet réel" doit être automatiquement ressentie par tous les autres utilisateurs connaissant l'attribut, même si l'utilisateur responsable de la modification ignore leur présence.

Pour définir des objets satisfaisant le principe d'UNICITE, nous utilisons la spécification algébrique des types. Classiquement, les occurrences d'un type ne satisfont pas le principe d'UNICITE. Nous distinguerons donc les "valeurs" declarées par des types comme dans les langages de programmation habituels, et les "objets réels", déclarés par des p-types, pour lesquels le système se charge de préserver l'UNICITE.

Etant donné notre choix de spécification algébrique, la définition d'un p-type est faite d'une façon analogue à celle des types classiques: on doit définir les fonctions du p-type et sa sémantique.

Parmi les fonctions, nous soulignons:

- les fonctions attribut dont le paramètre est le p-type que l'on définit et dont le type du résultat est soit un type classique, soit un p-type.

- la clé qui permet de distinguer deux occurrences différentes du p-type. ELle peut être une fonction attribut ou une composition de fonctions attributs, eventuellement privés, du p-type.

Pour l'expression de la sémantique du p-type, nous mettons en évidence l'expression des dépendances intrinsèques entre valeurs d'attribut (DIA) et, dans le cas de p-type dont certaines des fonctions attributs sont des p-types, l'expression de dépendances inter-objets (DIO).

Exemple:

p-type personne /*peut être vue comme la

définition du type nucléaire */

attributs :

NOM : personne —> chaîne

SEXE : personne —> (M,F)

ADRESSE : personne —> chaîne

AGE : personne —> entier

ENFS : personne —> ens[personne]

NSS : personne —> entier

SM : personne —> chaîne

clé : NSS

assertions :

$\forall p$ AGE(p) $\geq$ 0 $\wedge$ AGE(p) < 120

$\forall p$ SM(p) $\in$ {"OUI", "NON", "SURS", "EXEMPTE"}

$\forall p$ AGE(p) > 18 $\wedge$ SEXE(p) = M —> SM(p) $\in$ {"OUI", "SURS"}

fin

Pour satisfaire la propriété de multifacettage normalement associée aux objets bases de données, nous permettons la spécification modulaire du p-type. Au départ, on spécifie le noyau (type nucléaire) du p-type, c'est-à-dire tout ce qui est commun à toutes les facettes, puis les différentes facettes; la spécification d'une facette est faite par enrichissement (fonctions et/ou règles) d'une facette spécifiée auparavant ou du type nucléaire.

Exemple

vue personne-du-3e-âge enrichit personne

/*quand il n'y a pas d'ambiguïté possible nous donnons le même nom au p-type et à son type nucléaire */

assertions

$\forall p$ AGE(p) $\geq$ 65

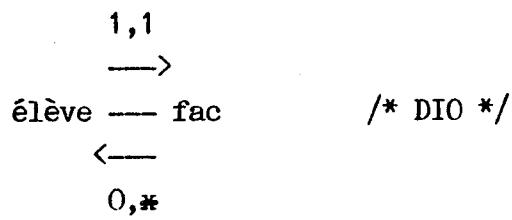
fin

vue élève enrichit personne

attributs

INSCRIPTION : élève —> fac
N-CARTE-ETUD : élève —> entier

assertions



fin

D'une manière naturelle, une nouvelle facette hérite de toute la définition de la facette à partir de laquelle elle est obtenue.

A un instant donné et en présence d'un p-type, l'utilisateur peut spécifier autant d'ensembles qu'il souhaite en utilisant le type générique ens [t].

Exemple

élèves : ens[élève]
élèves-internationaux : ens[élève]
service-gériatrie : ens[personne-du-3e-âge]

Par ailleurs, la spécification d'une association entre objets est faite en spécifiant un p-type dont certaines fonctions attributs sont des p-types et les autres représentent les informations propres à l'association. Ainsi, supposons que l'on spécifie les p-types cours et professeur:

p-type cours

. . .

fin

vue professeur enrichit personne

attributs

DIPLOME : professeur —> diplôme

fin

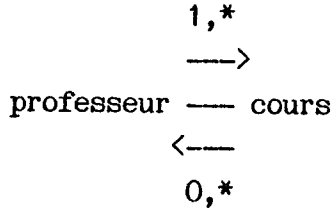
On peut définir l'association "professeur-de" par:

p-type professeur-de

attribut

ENSEIGNANT : professeur-de —> professeur
MATIERE : professeur-de —> cours
NB HEURES : professeur-de —> entier
/*attribut propre de la relation*/
SEMESTRE : professeur-de —> chaîne

assertions



/*un professeur fait au minimum un cours;un cours
peut n'être fait par aucun professeur*/

fin

En conclusion, nous dirons que les principales caractéristiques du modèle proposé sont:

- L'égale importance des attributs et des règles pour la spécificaion d'un objet.

- L'existence du concept d'objets satisfaisant le principe d'UNICITE. Ce concept, qui est l'unique concept du modèle, est implanté à travers les p-types. Il fait la distinction entre une valeur et un "objet-réel".

- La possibilité de spécification modulaire d'un objet (le type nucléaire et les facettes). Pour profiter au maximum de cet avantage, l'utilisateur est obligé d'approfondir sa connaissance de l'univers. En plus, elle facilite la preuve d'une spécification et l'évolution dans le temps des besoins des utilisateurs, ainsi que de l'évolution de l'univers lui-même.

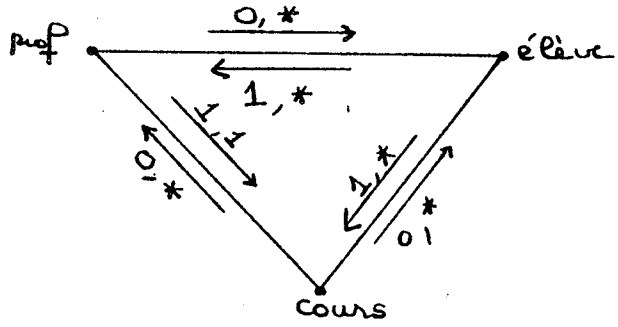
Le tableau ci-dessous fait le résumé des principales caractéristiques du modèle.

TYPES :	
NOM	
FONCTION:	
NOM	
TYPE	
EQUATIONS /*axiomes*/	
P-TYPES :	
TYPE-NUCLEAIRE :	
NOM	
FONCTION ATTRIBUT:	
NOM	
TYPE	
CLE /*fonction attribut ou composition de fonctions attributs */	
FONCTION :	
NOM	
TYPE	
ASSERTIONS /*axiomes du type nucleaire */	
/FACETTE	
NOM	
FONCTION ATTRIBUT	
FONCTION	
ASSERTIONS	
ENSEMBLE :	
NOM	
TYPE PARAMETRE /*p-type*/	
	- distinction entre valeurs et "objet reel"
	- representation fonctionnelle des attributs
	- type des fonctions attributs banalise
	- concept unique: p-type
	- separation de la specification d'un type de l'utilisation de ses occurrences
	- meme niveau de specification pour les attributs et les regles
	- mutifacettage ce qui permet d' introduire la "confidentialite" au niveau du schema
	- la totale centralisation des regles est parfois restreignante

I.4 - CAS PARTICULIERS DU MODELE

Les cas particuliers du modèle sont présentés à travers un exemple et puis sur forme de restrictions à imposer.

L'exemple choisi est l'exemple classique [CAD76] [NIC79] résumé par:



MODELE RESEAUX	MODELE RELATIONNEL
QUATRE ENREGISTREMENT : prof eleve cours "fictif"	CINQ RELATIONS: prof eleve cours T1(prof,cours) T2(prof,eleve,cours)
TROIS "SETS": s1 :(enr. pere = cours enr. fils = prof) s2 :(enr. pere = prof enr. fils = "fictif") s3 :(enr. pere = eleve enr. fils ="fictif")	ASSERTIONS : $\forall$ x y z T1(z,y,x) v T (x,v))
A SOULIGNER : -la creation d'un enregistrement supplementaire "fictive" pour implanter la relation (MxN).	A SOULIGNER : -les domaines des relations sont des ensembles de valeurs simples. AVANTAGES : CONCEPT UNIQUE

MODELE ENTITY-RELATIONSHIP	MODELE PROPOSE
TROIS ENSEMBLES-D-ENTITES: prof eleve cours DEUX ENSEMBLES-D-ASSOTIATIONS T1 (eleve,cours prof) avec (MxNxP) T2 (prof,cours) avec (1xN) A SOULIGNER -specification des ensembles de valeurs. -representation fonctionnelle des attributs. -correspondance entre la specification et l'idee que dans l'univers il y a des objets et des relations entre objets. DESAVANTAGES -restriction du type des fonctions attributs . -specification du type des objets faite en simultaneitee avec les ensembles. -choix de la cle d'une association arbitraire; en effet, la cle de T1 (composition des cles des entites) n'est pas une cle primaire. -un ensemble-d-associations ne peut etre defini sur des ensembles-association.	CINQ (ou QUATRE) P-TYPES : prof ou deux facettes d'un eleve meme p-type cours R1 (prof,cours) avec DIO :prof $\overset{1,1}{\underset{0,4}{\rightleftarrows}}$ cours R2 (R1,cours) avec DIO: R1 $\overset{1,n}{\underset{1,*}{\rightleftarrows}}$ eleve CINQ ENSEMBLES /*parametres par les 5 p-types*/ A SOULIGNER -permet de specifier prof et eleves comme deux facettes d'un meme p-type: necessaire pour une base de donnees de l'enseignement superieur (un individu peut etre professeur et eleve); non necessaire dans une base de donnees de l'enseignement primaire . - concept unique ,ce qui permet a un p-type de rentrer dans la composition d'autres p-types. DESAVANTAGES - l'existence d'un concept unique peut etre "non naturelle".

Le tableau ci-dessous résume comment exprimer dans le modèle proposé les concepts des autres modèles.

MODELE	POUR DEFINIR	CARACTERISTIQUES DU P-TYPE
RESEAUX	enregistrement	p-type n'ayant aucune fonction a resultat p-type.
	set (1xN)	p-type ayant -deux seules fonction attributs. -fonctions attributs de type p-type. -DIO p1 $\overset{f}{\underset{f^{-1}}{\rightleftarrows}}$ p2
RELATIONNEL	relation	p-type ayant des fonctions attributs a resultat simple.
E-R	ensemble-d-entite	p-type ayant -des fonctions attributs a resultat simple ou compose. -des fonctions attributs que ne sont pas de p-types. -ens[p-type] demande une specification explicite.
	ensemble-d-associations	p-type ayant - n$\geq$ 1 fonctions attributs a a resultat p-type -les p-types resultat des fonctions attributs precedentes ne peuvent pas avoir des fonctions attributs a resultat p-type.
	ensemble-de-valeurs	type abstrait

II - FORMALISATION ALGEBRIQUE

II.0 - INTRODUCTION

Dans ce chapitre, on fait une présentation formelle des principaux concepts et opérateurs du langage de spécification d'un schéma. Cette présentation, qui utilise la structure algébrique, met en relief des restrictions sur cette structure, nécessaires à la spécification d'univers dans le domaine des bases de données. D'autre part, la structure algébrique permet d'unifier un certain nombre de concepts, ainsi que d'en étendre d'autres. Parmi eux, nous soulignons l'unification des concepts d'entité et d'association entre entités, ainsi que l'extension du concept d'attribut.

II.1 - "OBJET" ET OBJET REEL

Les principes d'IMAGE et d'UNICITE (cf. chap. I.4) ont permis de dégager les notions d'"objet" et d'objet réel. Ces notions, qui y sont volontairement laissées floues, sont à la base du modèle proposé. En conséquence, le "noyau d'un langage qui permet leur spécification est au centre de notre travail. Nous définissons la notion d' "objet réel" par :

[def II-1]

Un "objet réel" est un objet, une situation ou un évènement caractérisé par un certain nombre d'attributs, et tel que :

i - il est identificable de façon unique
ii - les attributs sont représentables par des fonctions ayant toutes un même domaine. Celui-ci est composé de tous les "objets réel"s caractérisés par les mêmes attributs.

Cette définition souligne la possibilité de distinguer deux objets quelconques, pourvu qu'ils soient considérés comme des "objets réels". Ceci est important surtout quand ils sont caractérisés par la même collection d'attributs. D'autre part, la capacité des attributs d'être représentables par des fonctions dont le domaine est un ensemble d'éléments identifiables de façon unique, assure aux objets la

satisfaction du principe d'unicité (cf. chap. I.4). En effet, le résultat d'une fonction dépend, uniquement et exclusivement, de la valeur du paramètre ; il est donc complètement indépendant de la personne qui l'utilise, pourvu qu'elle en ait le droit. C'est seulement lors de la représentation en mémoire d'un objet que la preuve de représentation fonctionnelle des attributs doit être effectuée. Cette représentation ne doit pas conditionner la spécification. Au chap. V.2 nous proposons une représentation possible d'"objet réel".

[def II-2]

Un "objet" est l'abstraction d'une famille d'"objet réel"s caractérisés par les mêmes attributs.

POINT IMPORTANT

Pour éviter d'autres interprétations subjectives des notions d'"objet" et d'"objet réel", nous les nommerons jusqu'à la fin de ce travail p-type et p-occurrence. Le préfixe p provient de "partage" et il a été choisi dans le sens où un "objet réel" (ou p-occurrence) peut être partagé par plusieurs ensembles.

A la lumière du principe d'IMAGE, nous proposons la (re)définition du concept d'attribut d'un p-type :

[def II-3]

Un attribut ATTR d'un p-type p est une fonction totale qui fait correspondre à chaque p-occurrence de p une valeur unique du codomaine de ATTR :

$$ATTR : p \longrightarrow v$$

Le type des éléments du codomaine est unique et banalisé. Dans le contexte des bases de données, les types les plus fréquents sont le type simple (v), le type composé (v1, v2, ... ,vn) et le type ensemble (P(v) ou P(v1, v2, ... ,vn)).

D'autre part, si une fonction attribut représente une fonction

partielle, la valeur "indéfinie" appartient explicitement à son codomaine.

La définition proposée n'est pas complètement nouvelle. En effet, elle est équivalente à celle du modèle "SET-THEORETIC" [ENG76] :

"Attributs are relations mapping a specific objet type into either simple types or structured types".

[Proposition II-1]

Une sous-collection, non-vide, d'attributs d'un p-type, permet d'identifier de façon unique toute p-occurrence.

Ainsi, en utilisant le mot courant de "CLE" pour désigner cette sous-collection, nous pouvons dire que :

i - La clé est soit une fonction attribut, soit une construction [BAC78] de n fonctions attributs :

$$CLE == ATTRi \mid ATTRi \times ATTRj \times \ldots \times ATTRn$$

ii - Deux p-occurrences différentes ont des clés différentes:

$$\forall pi, pj \quad pi \neq pj \longrightarrow CLE(pi) \neq CLE(pj)$$

iii - La valeur de la CLE d'une p-occurrence est la même pendant toute la vie de la p-occurrence.

Il ne faut pas oublier que la définition d'attribut n'interdit pas que la fonction soit privée ; dans ce cas, la CLE est donc une clé interne [COD79].

II.2 - P-TYPE ET TYPE ABSTRAIT

La spécification d'un p-type demande la spécification de ses fonctions attributs, et plus précisément, du type du codomaine de chacune d'elles (cf. def. II-3), ainsi que la spécification des règles que les p-occurrences du p-type doivent satisfaire.

Le résultat d'une spécification de p-type est un "ensemble virtuel" de p-occurrences caractérisées par les mêmes fonctions attributs. Nous appelons ensemble support de t, ou encore support de t l'ensemble virtuel

des p-occurrences du p-type p.

[Proposition II-2]

Spécifier un p-type t équivaut à spécifier une algèbre hétérogène dont le type d'intérêt est t.

Et, comme "spécifier un type abstrait équivaut à spécifier une algèbre" [GOG80] :

[Proposition II-3]

La spécification d'un p-type t, caractérisé par n fonctions attributs, est la spécification d'un type abstrait algébrique t, tel que :

i- n de ses fonctions sélectrices sont les fonctions attributs ;
ii- il existe une fonction génératrice déduite implicitement à partir de ses n fonctions attributs :

CONSTRUCTEUR : (t1, t2, ... tn) —> t

où t1, t2, ... tn sont les types de ses n fonctions attributs.
iii- un certain nombre de ses équations traduisent les propriétés de la "CLE", c'est à dire l'égalité d'éléments de Wt (cf. Proposition II-1).

Dans cette proposition, nous avons utilisé la convention habituelle qui consiste à donner le même nom au domaine principal d'une présentation de type abstrait et au type lui-même [BER79]. D'autre part, le support de t est l'ensemble associé au domaine principal de la présentation du type t : l'ensemble support (cf. def. I-2) de l'algèbre décrite par le type t contient donc le "support de t".

Le mot-clé "p-type" introduit lors de la spécification du p-type personne (cf. def. I-4) pourrait donc être remplacé par "type". Cependant, nous le maintenons, d'une part pour indiquer que les éléments associés à son domaine principal sont des p-occurrences, et, d'autre part, pour introduire un langage d'expression plus proche de celui des bases de données.

L'algèbre des termes du p-type personne (cf. déf. I-4) contient tous

les n-uplets (n=7) satisfaisant les axiomes du p-type. En particulier :

<Ana, F, Uriage, 31, (280 ... , ...), 252 ABC, NON>

<Isabelle, F, Grenoble, 33, (0), 25 ... , NON>

sont deux de ses termes, c'est-à-dire qu' ils satisfont les axiomes du p-type personne. Cependant, en présence des deux termes précédents, le 7-uple :

<Ana, F, Grenoble, 31, (280 ... , ...), 252 ABC, NON> n'est pas un terme valide : les axiomes sur la clé ne sont pas satisfaits.

D'autre termes de l'algèbre des termes du p-type personne sont par exemple :

```
NOM (<ANA, ...>)  /*terme du type de la fonction NOM*/
ADRESSE (<JEAN, PARIS, ...>)  /*terme du type de la
                                fonction ADRESSE*/
```

L'importance de la spécification des règles peut être vue sur ce petit exemple. Ainsi,

<MARIE, F, ... , SURS>

est un terme valide de personne, même si dans la réalité, on peut garantir que, en France, aucune femme n'est jamais sursitaire. L'interdiction des termes appartenant à :

$$\{p \mid SEXE(p) = F \text{ et } SM(p) = \text{"SURS"}\}$$

demande la spécification de la règle :

$$\forall p \; SEXE(p) = F \longrightarrow SM(p) = \text{"NON"}$$

L'extrême limite dans la spécification des règles étant la spécification de chaque p-occurrence par une règle.

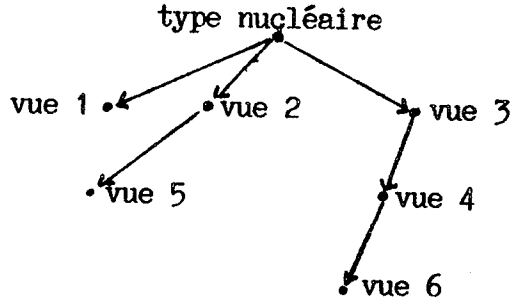
[Proposition II-4]

La spécification d'un objet ou d'une association entre objets est formellement identique.

Cette proposition découle naturellement de la définition d'attribut (cf. def. II-2) conjuguée au fait que spécifier un p-type revient à spécifier un type abstrait : les codomaines de plusieurs (éventuellement une seule) fonctions attributs peuvent être eux-mêmes composés de p-occurrences.

II.3 - TYPE NUCLEAIRE ET VUES D'UN P-TYPE

Nous considérons qu'il est fondamental qu'un utilisateur puisse spécifier un p-type de façon modulaire et évolutive. En conséquence, nous avons décidé de lui permettre de spécifier, dans un premier temps, le type nucléaire, ou vue minimale du p-type, puis ses vues. En sachant qu'une vue est obtenue par enrichissement (cf. def. I-10) du type nucléaire ou d'une autre vue, déjà spécifiée, nous concluons que la spécification d'un p-type est une spécification modulaire et hiérarchique :



[Proposition II-5]

Spécifier le type nucléaire d'un p-type t équivaut à spécifier la vue "minimale" selon laquelle toute p-occurrence de t peut être perçue.

[def II-4]

Le type nucléaire d'un p-type t est un type abstrait t : <S,F,E>.

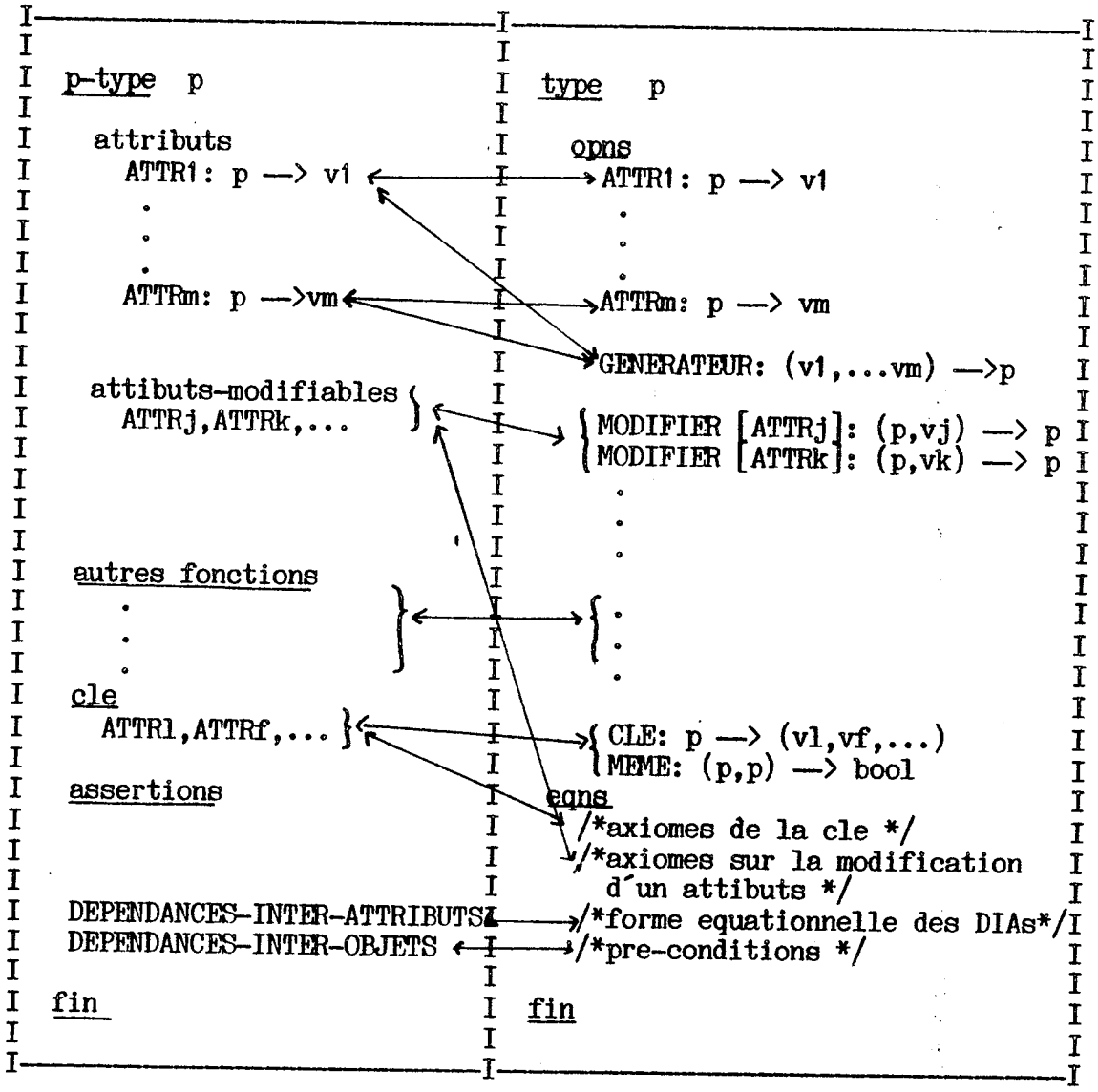
Sauf confusion possible, le nom du type nucléaire d'un p-type est le même que celui du p-type. Ainsi, le type nucléaire du p-type "personne" est lui-même dénommé "personne".

Pour la spécification d'un type nucléaire, nous utilisons une forme

externe proche de celle couramment employée en bases de données:

```
p-type t

  attributs

    /*spécification du plus petit nombre de
      fonctions attributs nécessaires à
      caractériser les p-occurrences de t*/

  attributs modifiables

    /*spécification des fonctions attributs
      dont la valeur du résultat peut
      changer avec le temps*/

  autres fonctions

    /*spécification d'autres fonctions*/

  clé

    /*spécification de la composition de
      la clé*/

  assertions

    /*spécification des règles*/

fin
```

Cette unité syntaxique peut être automatiquement transformée dans une unité type abstrait. Nous présentons ci-dessous une première version de cette transformation. Une version plus élaborée est déductible des résultats du chapitre V.


p-type p
attributs
ATTR1: p —> v1
ATTRm: p —>vm
attibuts-modifiables
ATTRj,ATTRk,...
autres fonctions
cle
ATTR1,ATTRf,...
assertions
DEPENDANCES-INTER-ATTRIBUTS
DEPENDANCES-INTER-OBJETS
fin
type p
opns
ATTR1: p —> v1
ATTRm: p —> vm
GENERATEUR: (v1,...vm) —>p
MODIFIER [ATTRj]: (p,vj) —> p
MODIFIER [ATTRk]: (p,vk) —> p
CLE: p —> (v1,vf,...)
MEME: (p,p) —> bool
eqns
/*axiomes de la cle */
/*axiomes sur la modification d'un attibuts */
/*forme equationnelle des DIAs*/
/*pre-conditions */
fin

Exemple :

```
p-type personne
  attributs
    NOM: personne --> nom-pers
    SEXE: personne --> {M,f}
    ADRESSE: personne-->adresse
    AGE: personne --> entier
   ENFS:personne-->ens[personne]
    NSS: personne-->ident
    SM: personne-->{OUI,NON,
                    SURS,EXEMP}

  attributs-modifiables

    AGE, ENFS, SM

  cle

    NSS

  assertions

Vp AGE(p)>0 n AGE(p)< 120
Vp AGE(p)>18 n SEXE(p)="M"-->

fin
```

```
type personne
  opns
    NOM: personne --> nom-pers
    SEXE: personne --> {M,F}
    ADRESSE: personne --> adresse
    AGE: personne --> entier
    ENFS:personne-->ens[personne]
    NSS: personne --> ident
    SM:  personne --> {OUI,NON,
                       SURS,EXEMP}
    GENERATEUR:(nom-pers,{M,F},
                adresse,entier,
                ens[personne],
                ident,{OUI,NON,
                SURS,EXEMP}) -->
                        personne

    MODIFIER [AGE]:(personne,
                    entier)-->per-
                              sonne
    MODIFIER [ENFS]:(personne,
                    ens[personne]--> 
                              personne
    MODIFIER [SM]: (personne,{OUI,
                    NON,SURS,EXEMP})
                   --> personne

  CLE: personne-->ident
  MEME: (personne,personne)-->
                        personne

  eqns
/*axiomes de la cle
  CLE(p)==NSS(p)
  Vx,x' MEME(x,x')==CLE(x)=CLE(x')
  Vx,x' MEME(X,x')==V ATTRi
                (ATTRi(x)=ATTRi(x'))
/*axiomes de la modification d'
  attributs */
Vx,Vy AGE(x,MODIFIER [AGE](x,y))==y
Vx,Vy ENFS(x,MODIFIER [ENFS](x,y))
                        == y
Vx,Vy SM(x,MODIFIER [SM](x,y))==y
/*axioms des DIAs*/
<*,M,*,]18,120],*,*,{OUI,SURS}>==v
<*,M,*,]18,120],*,*,{NON,EXEMP}>==f
/*l' "*" signifie n'importe laquel-
 le valeur */

fin
```

Une vue d'un p-type t est construite à partir de son type nucléaire ou d'une autre vue de t spécifiée auparavant. Ainsi :

[def. II-5]

Soit le type t' : <S',F',E'> correspondant au type nucléaire ou à une vue du p-type t ; la spécification de la vue t" à partir de t' est la donnée d'une présentation p = <S',F'',E''>, telle que p enrichit t'.

Le type t'' est le type t' enrichi par p. De la conjugaison de (déf. II-5) avec (déf. I-10) on peut conclure que les algèbres décrites par t' et t'' ont un même "ensemble support" (cf. def. I-2). Ainsi :

[Proposition II-6]

Associé à un p-type t, pour lequel n vues ont été spécifiées, il existe une famille de n + 1 algèbres fondées sur le même ensemble support.

En particulier, les n + 1 algèbres, une pour le type nucléaire de t et une pour chacune de ses vues, ont en commun le support de t. Celui-ci est associé au domaine principal de leurs présentations.

D'autre part, par définition même, si la présentation p = <S',F'',E''> enrichit le type t' : <S',F',E'> alors, il existe un morphisme d'inclusion entre $\Sigma' = (S', F')$ et $\mathfrak{L}' = (S', F' \cup F'')$, c'est à dire $\sigma : \Sigma' \longrightarrow \mathfrak{L}''$. De plus,

$$\underline{\sigma}\,(W\,\Sigma'',E'') \subseteq W\,\mathfrak{L},E'$$

Puisque $W\Sigma',E'$ est l'algèbre des termes de t' et $W\,\mathfrak{L}',E''$ est celle de t'', nous pouvons conclure que tout terme satisfaisant t'', est, "en oubliant" les fonctions attributs ajoutées lors de l'enrichissement, un terme de t'. Dans un "langage" plus proche des bases de données, ce résultat signifie que toute p-occurrence qui peut être perçue selon la vue t'', peut aussi être perçue selon t', où t'' est obtenue à partir de t'.

En utilisant une forme externe proche des bases de données, la spécification d'une vue est faite par:

vue t'' enrichit t'

attributs

/*spécification des fonctions attributs
de la présentation p qui enrichit t' */

attributs modifiables

/*spécification des attributs de p ou de
t' dont la valeur peut changer */

fonctions

/*autres fonctions de p */

assertion

/*règles propres a p */

fin

Les fonctions et les règles explicitées lors de la spécification de t'' par enrichissement de t' sont les fonctions et les règles de la présentation p qui enrichissent t'. Par commodité, nous les dénommons fonctions propres et règles propres à t''. Selon la forme externe proposée, on nomme explicitement le type qui va être enrichi (t') et le type résultat de l'enrichissement (t'').

D'une façon plus précise on devrait dire que l'on nomme le type d'intérêt de l'enrichissement(t') et l'environnement de programmation (t'') où le type t' dispose des nouvelles fonctions spécifiées par l'enrichissement.

Par définition, si t' : <S',F',E'> a été enrichi par la présentation p:<S',F'',E''>, la présentation de t'' est:

t" :<S,F' U F'',E' U E'' >

Ainsi, la définition automatique du type abstrait

t´´ : <S´,F´ U F´´,E´ U E´´> obtenue par la mise en commun des présentations <S´,F´,E´> et <S´,F´´,E´´>, est semblable à celle d´un type nucléaire.

Exemple :

Soit la spécification de la vue "personne-du-3e-âge" par enrichissement de "personne" :

vue personne-du-3e-âge enrichit personne

assertions

¥p AGE(p) ≥65

fin

Le nouveau type "personne-du-3e-âge" a les fonctions de "personne" ; cependant, son assertion propre (¥p AGE(p) ≥ 65) modifie les équations correspondant aux DIAs. Ainsi, les deux premières équations de "personne" [cf. exemple précédent] sont remplacées par :

<*,*,[0,65[,*,*,*> == faux

<*,M,[65,120],*,*,{CUI,SURS}> == vrai

<*,M,[65,120],*,*,{NON,EXEMP}> == faux

Les résultats de ce paragraphe permettent de conclure que toute p-occurrence satisfaisant "personne-de-3e-âge" satisfait "personne" ; et, donc, toute p-occurrence de type "personne-du-3e-âge" peut être paramètre effectif d´une spécification paramétrée par personne.

II.4 - SPECIFICATION D'ENSEMBLES

[def II-6]

Un ensemble en base de données est une instance du type générique "ens" dont les seuls paramètres effectifs doivent être des p-types (c'est-à- dire, satisfaire la propriété d'"égalité-par-CLE" : deux instances d'un p-type représentent la même p-occurrence si la valeur de leur clé est la même (cf. chap. V-1).

Une première version du type ensemble est proposée ci-dessous :

```
type ens exige égalité-par-clé [t opns CLE]

  opns

      privé EVIDE : —> ens[t]

      INSERER     : (ens[t],t) —> ens[t]

      SUPPRIMER   : (ens[t],t) —> ens[t]

      REINSERER   : (ens[t],t) —> ens[t]

      EST-VIDE    :  ens[t]    —> bool

      APP : (ens[t],t) —> bool

  qns

/*voir chapitre IV*/

  fin
```

Des instances du type ensemble sont, par exemple :

caisse-allocation-familiale : ens[personne]

malades-du-3e-âge : ens[personne-du-3e-âge]

II.5 - SPECIFICATION D'UN SCHEMA

La spécification d'un schéma est la spécification des types d'intérêt d'un univers, et en particulier des p-types et des ensembles paramétrés par des p-types.

II.6 - MODIFICATION DE LA SPECIFICATION D'UN UNIVERS

La modification d'un schéma (cf. chap. 0-1) se différencie de sa spécification initiale par l'existence de valeurs déjà mémorisées. Dans ce travail, les seuls cas de modification d'un schéma que nous abordons sont :

- créations de nouveaux ensembles
- spécification d'une vue par enrichissement d'une autre vue pré-spécifiée
- modification d'une vue par ajout de nouvelles fonctions et/ou équations.

Plus concrètement, nous n'abordons que la troisième opération, puisque les deux premières sont indépendantes de l'existence de valeurs en mémoire. En général, la modification d'une vue n'a de signification que dans un contexte de modification d'un schéma.

[def.II-7]

La modification d'une vue t' est un enrichissement du type t' : <S',F',E'> par une présentation <S',F'',E''>. Après enrichissement, le seul type disponible est t' : <S',F' U F'',E' U E''>.

La modification d'une vue, dans un contexte de modification d'un schéma, exige la vérification supplémentaire que toutes les p-occurrences, modèles de la vue avant la modification, continuent à l'être après la modification. Cette vérification supplémentaire découle du contexte

spécifique des bases de données, où le besoin de rémanence des p-occurrences est fondamental. Ainsi,

[Proposition II-7]

Soit la vue t´ : <S´,F´,E´> avant modification et la vue t´ : <S´,F´ U F´´,E´ U E´´> après modification. La mise à jour de t´ est valide ssi :

i - la présentation <S´,F´´,E´´> enrichit t´

ii - toute p-occurrence de t´ déjà connue de la base de données satisfait t´ après la mise à jour des nouvelles fonctions spécifiées dans la modification.

L´unité syntaxique qui permet de modifier une vue t´ est :

```
mis-a-jour t´
  attribut
    /*attributs propres à la présentation qui enrichit t´ */
  attributs modifiables
    /*le résultat de l´application de l´attribut sur une p-occurrence
      n´est pas figé dans le temps*/
  fonctions
    /*fonctions propres à l´extension */
  assertions
    /*nouvelles règles pour le type t´ */
fin
```

Nous appelons fonctions propres à la mise à jour, les fonctions attributs, les attributs modifiables et les fonctions de la présentation qui enrichit t'. De la même façon, nous appelons assertions (ou règles) propres à l'extension, les nouvelles règles spécifiées lors de la mise à jour de t'.

L'écriture formelle de l'opération de mise-a-jour est équivalente à celle de création d'une nouvelle vue.

Exemple

mise-a-jour personne-du-3e-âge

attributs

PAYS : personne-du-3e-âge —> pays

fin

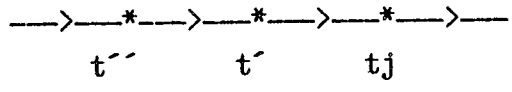
Après cette opération, le type "personne-de-3e-âge" dispose d'un sélecteur supplémentaire (PAYS) ; en plus, son constructeur est lui aussi modifié :

* CONSTRUCTEUR : (nom-de-personne,(M,F), adresse-de-personne.
entier, ens[personne], identificateurs,
{"OUI","NON","SURS","EXEMPTE"}, pays)
—> personne-du-3e-âge.

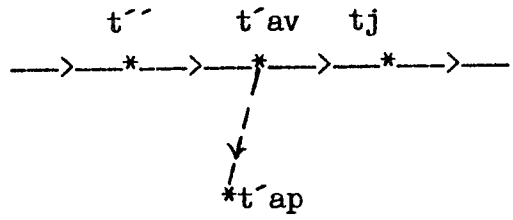
La mise-à-jour d'une vue soulève des problèmes au niveau de l'inclusion des présentations de types. Ainsi, supposons que $t'' \subseteq t'$ et que $t' \subseteq tj$; supposons aussi que l'on modifie t' à l'aide de la présentation <S',F'',E''>. Avant la mise-à-jour de t' : <S',F',E'> on avait les relations d'inclusion suivantes entre les présentations des types

$$t'' \subseteq t', \quad t' \subseteq tj \quad \text{et} \quad t'' \subseteq j$$

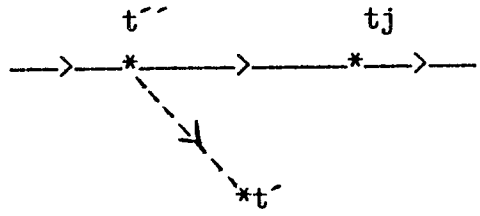
Ces relations peuvent être représentées graphiquement par:



Après, l´extension t´ a comme présentation <S´,F´ U F´´,E´ U E´´>, graphiquement représentable par



ainsi, étant donné que t´ap : <S,F´ U F´´,E´ U E´´> remplace t´av : <S´,F´,E´>, les seules relations d´inclusion, maintenues automatiquement, sont t´´ ⊆ tj et t´´ ⊆ t´ap :

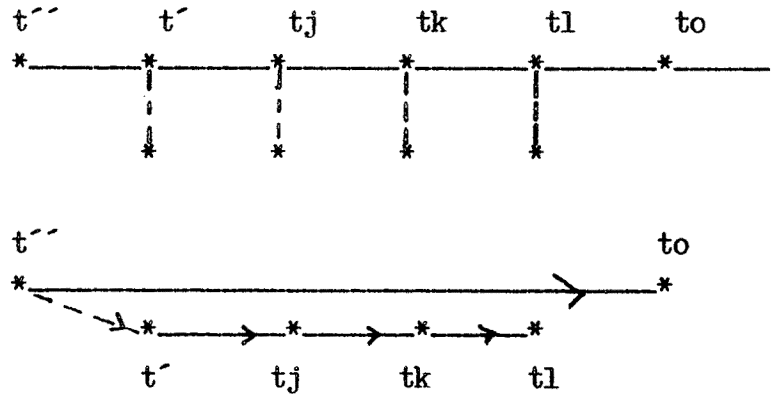


Or, la relation d´inclusion de présentation de types est importante pour la maximisation des vérifications statiques des types. En conséquence, il peut être nécessaire de déterminer si après la mise-à-jour, t´ continue à être inclus dans tous les types ti, tels que, t´av ⊆ ti.

[Proposition II-8]

Soient t´ : <S,F´,E´> et ti : <S,Fi,Ei>, deux types tels que t´av ⊆ ti : soit p une présentation <S,F´´,E´´> qui enrichit t´ [cf. déf. II-7]. Après l´enrichissement, la relation t´ap ⊆ ti ssi la présentation p enrichit aussi ti, ou, plus précisément p met-à-jour ti (cf. proposition II-4).

En général, on arrive à une situation représentée graphiquement par :



Si le type to ne peut pas être enrichi par p, aucun autre type tp tel que to tp le peut.

Les effets secondaires d'une mise-à-jour d'un type exige qu'elle soit utilisée avec précaution. Ainsi, la modification du type nucléaire de personne par :

mise-à-jour personne

assertions
    ∀p AGE(p) ≤ 50
fin

aurait comme conséquence que personne-du-3e-âge n'est plus contenu dans personne: les axiomes de personne-du-3e-âge en conjonction le nouvel axiome (∀p AGE(p) ≤ 50) est un système d'axiomes incohérent.

Pour éviter la situation ou une vue n'est pas un enrichissement du type nucléaire, nous proposons que la modification du type nucléaire soit interdite.

III - DEPENDANCES INTER-ATTRIBUTS

III.0 - INTRODUCTION

La spécification algébrique d'un p-type exige la donnée de ses n fonctions attributs [chap 2].Cependant, ceci reste incomplet si l'on ne donne que les noms et les codomaines des fonctions,c'est à dire leur syntaxe.En effet,l'absence d'une spécification sémantique de ces fonctions soulève des problèmes (classiques) de correspondance entre les occurrences d'une abstraction et les p-occurrences de l'univers.Ainsi,considérons le p-type personne:

p-type personne

attributs

NOM : chaine

AGE : entier

SM : chaine

NSS : nss

modifiables

AGE,SM

cle

NSS

assertions

∀p AGE(p) ∈ [0,120]

∀p SM(p) ∈ {OUI,NON,SURS,EXEMPTE}

fin

En l'absence d'autres règles traduisant la sémantique des p-occurrences du p-type,l'algèbre des termes du type abstrait personne contient tous les éléments du produit cartésien des 4 codomaines de ses fonctions attributs.Ainsi,

1) <paul ,0,oui, 183AB>
2) <remi ,0,non, 183AG>
3) <paul ,0,surs, 183AT>
4) <henri ,0,exemp,183AS>
5) <jean ,0,oui, 183AM>
6) <jean ,0,non, 183AN>
7) <pierre,0,oui, 183AR>
8) <pierre,30,oui, 183BL>

sont des termes "syntaxiquement" valides.

On peut y distinguer deux familles de termes: ceux qui ne peuvent pas être des images de p-occurrences et ceux qui sont des images de p-occurrences réelles ou potentielles.

La premiere famille est constituée des termes:
-1,3,4,5,7 :une personne d'âge zero ne peut avoir fait de service militaire,ni être sursitaire,ni être exemptée.
-8 :une personne née en 1983 ne peut pas avoir 30 ans en 1983 ni avoir effectué son service militaire.

La deuxième famille,complément de la première,est composée des termes 2 et 6.

Dans ce chapitre nous nous intéressons à la spécification des règles qui permettent de circonscrire avec le plus de précision possible chacune des deux familles. Dans l'exemple présenté, il nous aurait suffi d'exprimer que:

1) l'âge de quelqu'un est soit
   ANNEE-ACTUELLE - ANNEE-DE-NAISSANCE soit
   ANNEE-ACTUELLE - ANNEE-DE-NAISSANCE - 1

2) quelqu'un ayant un âge inférieur ou égal à 18 ans n'a pas encore effectué son service militaire.

Dans une spécification plus précise du type personne d'autres lois seraient sûrement nécessaires. Cependant la spécification des règles exprimant l'interdépendance entre les valeurs des attributs d'une p-occurrence est, en général, rendue difficile par l'absence ou la non connaissance de règles faisant intervenir explicitement tous les attributs du p-type. Dans la plupart des cas, une combinaison de valeurs d'attributs est susceptible d'exister dans l'univers si et seulement si elle satisfait toutes les lois exprimées. On se trouve donc devant la contradiction suivante: d'une part avoir besoin d'un grand nombre de lois pour permettre une spécification fine et d'autre part les minimiser pour limiter les vérifications à effectuer. Pour cela, il faudrait que l'utilisateur puisse garantir que les lois qu'il a exprimées sont les seules nécessaires à la spécification de son type. Ceci n'étant pas envisageable en général, les redondances, c'est à dire la multi-spécification d'une loi, et les "incohérences" doivent être éliminées a posteriori.

Dans ce chapitre nous nous intéressons aux dépendances entre valeurs d'attributs; ces dépendances sont exprimées par l'utilisateur à l'aide d'un sous-ensemble de la logique du premier ordre. Dans un premier temps, nous proposons une méthode basée sur la notion de partition d'un ensemble, qui permet d'une part de définir des classes d'équivalence des occurrences d'un type par rapport aux dépendances exprimées, et d'autre part de transformer les assertions, écrites en logique du premier ordre, en formules de la logique propositionnelle. Ces transformations sont à elles seules très importantes puisqu'elles permettent d'utiliser les méthodes décidables de la logique propositionnelle pour mettre en évidence les redondances et les "incohérences". Puis, dans un deuxième temps, nous proposons une méthode ad hoc basée sur la sémantique des assertions exprimées en logique propositionnelle.

III-1- DÉFINITIONS ET NOTATIONS

Dans ce chapitre nous reprenons des résultats présentés en [SAI84].

Dans un type t, nous dénotons $\Delta(Xi)$ l'ensemble fini des éléments du codomaine de la fonction attribut Xi. Comme tout ensemble, $\Delta(Xi)$ peut être spécifié par énumération de ses éléments ou par un prédicat. Dans ce travail, nous avons besoin de distinguer les ensembles ordonnés ([0,120], entier, etc.) des ensembles non ordonnés (chaine, {BLEU,ROUGE}, etc.). De plus nous introduisons trois identificateurs génériques INF,SUP et AUTRES:

-INF et SUP représentent respectivement la borne inférieure et la borne supérieure d'ensembles ordonnés dont les limites inférieure ou supérieure n'ont pas été explicitement spécifiées.

Ces deux identificateurs permettent de réécrire des assertions faisant intervenir des ensembles ordonnés dont la ou les bornes ne sont pas explicites. Ainsi, soit AGE une fonction attribut spécifiée uniquement par:

AGE : entier /* Δ(AGE) est fini mais non explicitement limite */

avec l'assertion écrite en logique du premier ordre:

$\forall p$ AGE(p) < 30 v AGE(p) >65

En utilisant INF et SUP, dont la valeur exacte est sans importance, [cf III-2], nous pouvons réécrire l'assertion précedente:

$\forall p$ AGE(p) $\in$ [INF,30[v AGE(p) $\in$]65,SUP]

-AUTRES repésente l'ensemble composé par tous les éléments non explicitement énumérés d'un ensemble non ordonné. Ainsi, soit la fonction attribut VILLE spécifiée uniquement par:

VILLE : chaine /* Δ(VILLE) est un ensemble non

ordonné et non énuméré */

Soient Grenoble, Marseille et Paris des villes explicitement énumérées dans les assertions. Nous écrivons:

Δ(VILLE)={GRENOBLE,MARSEILLE,PARIS} U AUTRES

"AUTRES" groupe donc toutes les autres villes dont l'existence n'a pas été explicitée. Sa composition exacte est sans importance pour la spécification des règles.

DEFINITIONS ET RAPPELS

[def III-1]

Un prédicat sur la valeur de la fonction attribut Xi du type t est un prédicat de la forme:

$$\forall tj\ Xi(tj)\ R$$

où

-tj est de type t

-R est soit $\in$ (appartient) soit = ou $\neq$ ou $\geq$ ou > ou < ou $\leq$

- $C \subseteq \bigwedge(Xi)$

[def 3-2]

Un prédicat élémentaire sur la fonction attribut Xi du type t est un prédicat de la forme

$$\forall tj\ Xi(tj) \in \delta$$

où

-tj est de type t

- $\delta \subseteq \bigwedge(Xi)$

-le seul symbole de prédicat est $\in$

A partir de maintenant nous considérons que tj est implicitement quantifié universellement et nous omettons le symbole $\forall$.

[proposition III-1]

Tout prédicat sur les valeurs d'une fonction attribut (cf def III-1) peut être réécrite sous la forme d'un prédicat élémentaire sur la même fonction attribut.

Ainsi, considérons les assertions suivantes:

a1) $X1(t) \leq 20 \wedge X1(t) \geq 10 \longrightarrow X3(t)$="OUI"

a2) $X1(t) \geq 10 \wedge X1(t) \leq 15 \longrightarrow X3(t)$="NON"

a3) $(X3(t)$="OUI" $\vee\ X3(t)$="SURS") $\wedge\ X2(t) \geq 100 \longrightarrow 20 \leq X1(t) \leq 30$

elles se réécrivent:

X1(t) ∈ [10,20] —> X3(t) ∈ {OUI}
X1(t) ∈ [10,15] —> X3(t) ∈ {NON}
X3(t) ∈ {OUI,SURS} ∧ X2(t) ∈ [100,SUP] —> X1(t) ∈ [20,30]

[def III-3]

La fonction EXT est la fonction qui, appliquée à un prédicat élémentaire, délivre son extension (c'est à dire l'ensemble des valeurs de l'attribut pour lesquelles le prédicat est vrai).

Ainsi EXT(Xi(tj) ∈ $\mathcal{S}$) = $\mathcal{S}$

[def III-4]

Une Dépendance Inter Attributs est une assertion sur les valeurs des attributs d'une p-occurrence. Nous la notons DIA.

[def III-5]

Une assertion représentant une dépendance entre valeurs d'attributs, notée DIA, est une clause de la logique du premier ordre ayant la forme suivante:

A ∧ B ... => M v N v ...

où

-A,B, ... M,N, ... sont des prédicats élémentaires
-A,B, ... prédicats de l'antécédent de l'implication, doivent porter sur des attributs différents.
-M,N, ... prédicats du conséquent de l'implication, doivent être des prédicats portant sur le même attribut.

Un cas particulier de ce type de clauses est celui où l'antécédent n'existe pas; c'est un des moyens de restreindre le codomaine d'une fonction attribut. Ainsi, la clause:

—> AGE(p) ∈]20,65]

restreint le codomaine de AGE aux entiers compris entre 21 et 65.

D'une façon stricte, les assertions représentant les DIAs ne sont pas des clauses de Horn [CHAN73], puisque le conséquent est une disjonction de littéraux. Mais ces littéraux portant sur un même attribut, il leur correspond naturellement un littéral unique dont l'extension est l'union des extensions des littéraux de départ. A la forme près, on se trouve donc toujours en présence de clauses de Horn.

Nous dénotons PP(Xi) l'ensemble de tous les prédicats d'un ensemble de DIAs portant sur une même fonction attribut Xi. Ainsi, dans l'exemple précédent:

PP(X1)={X1(t) ∈ [10,20], X1(t) ∈ [10,15], X1(t) ∈]20,30]}

PP(X2)={X2(t) ∈ [100,SUP]}

PP(X3)={X3(t) ∈ {OUI}, X3(t) ∈ {NON}, X3(t) ∈ {OUI,SURS}}

III-2 SOUS-DOMAINES STABLES

Formellement, un ensemble d'assertions écrites en logique du premier ordre est inconsistant (ou incohérent ou contradictoire) ssi il n'est satisfait pour aucune valeur de ses variables. Dans notre cas, la variable représente une occurrence du type pour lequel l'assertion a été spécifiée. Ainsi, l'inconsistance d'un ensemble de DIAs signifie qu'aucune occurrence du type ne peut exister dans la base de données. Un de nos objectifs est de tester si un ensemble de DIAs est inconsistant; cependant nous nous proposons de raffiner la notion d'inconsistance en introduisant la D-inconsistance (inconsistance sur les domaines).

Par exemple, les trois assertions du paragraphe précédent ne sont pas inconsistantes; cependant, la première et la deuxième interdisent l'existence de p-occurrences telles que $X1(t) \in [10,15[$, ce qui n'apparaît explicitement dans aucune des assertions.

Nous voulons que l'utilisateur soit conscient que l'interdiction du sous-domaine $[10,15[$ de $\Delta(X1)$ est une conséquence logique des DIAs, et qu'il prenne une décision explicite sur son interdiction. En effet, un ensemble d'assertions réalise une partition de toutes les valeurs possibles des occurrences d'un type, et, en conséquence il exclut certaines combinaisons de valeurs d'attributs. Avec la D-inconsistance, nous voulons caractériser les combinaisons dont l'exclusion est susceptible d'être la conséquence d'erreurs de compréhension et/ou de spécification des DIAs.

Pour mettre en évidence les sous-domaines qui interviennent dans les combinaisons ainsi exclues nous transformons les assertions originelles de telle façon qu'elles fassent uniquement référence à des prédicats ayant des extensions soit disjointes soit identiques.

Pour définir ce que nous appelons les sous-domaines stables d'une

fonction attribut Xi, nous définissons la partition de $\bigwedge(Xi)$ par les prédicats de PP(Xi),

rappels [STA77]

-Une partition d'un ensemble A est une collection de sous-ensembles non vides et disjoints de A tels que leur union est l'ensemble A lui-même.
-Un élément d'une partition est appelé bloc logique ou bloc.
-Le nombre de blocs logiques d'une partition constitue son rang.
-Soit PI et PI' deux partitions d'un ensemble non vide A. On dit que PI' "affine" PI ssi tout bloc de PI' est contenu dans un bloc de PI.
-Soit PI1 et PI2 deux partitions d'un ensemble non vide A. Le produit des partitions PI1 et PI2, noté PI1.PI2, est une partition PI de A telle que
1) PI affine à la fois PI1 et PI2
2) tout PI' qui affine simultanément PI1 et PI2 affine aussi PI.

Soit Xi une fonction attribut du type t et P un prédicat élémentaire sur Xi. Le prédicat p a donc la forme $Xi(t) \in \delta$. De plus il définit une partition PI(P) de $\bigwedge(Xi)$; PI(P) est de rang 2 et ses blocs sont respectivement $EXT(P)=\delta$ et $EXT(\tilde{}P)=\bigwedge(Xi)-\delta$. Ces deux blocs sont aussi les deux classes d'équivalence de la relation binaire R(P) définie par "ayant la même valeur de vérité pour P" :

X R(P) Y ssi P(X)=P(Y) c'est à dire:

$$(X \in \delta \wedge Y \in \delta) \vee (X \in (\bigwedge(Xi)-\delta) \wedge Y \in (\bigwedge(Xi)-\delta))$$

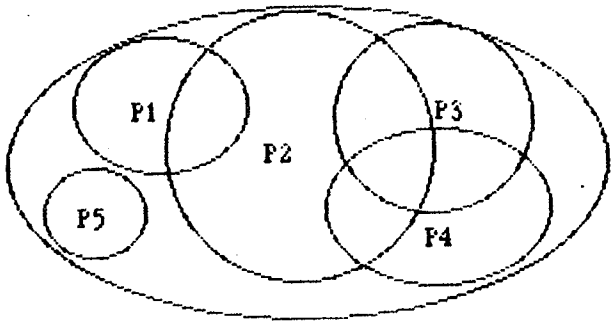
Nous allons définir de façon récursive la fonction SSD(P1,.. ,Pi,...,Pn) où tous les Pj, $j \in [1,n]$, sont des prédicats élémentaires sur une même fonction attribut Xi; le résultat de SSD est une partition de $\bigwedge(Xi)$.

[def III-6]

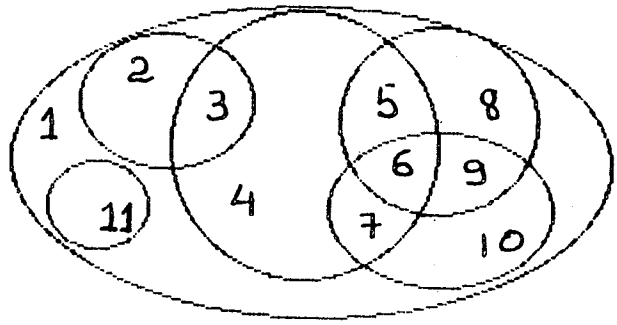
-Pour deux prédicats quelconques P1 et P2, SSd(P1,P2) est le produit des partitions PI(P1) et PI(P2); il est au plus de rang 4 et dans ce cas les blocs sont EXT(P1 ∧ P2), EXT(P1 ∧ ~P2), EXT(~P1 ∧ P2) et EXT(~P1 ∧ ~P2).

-SSD(P1, ... ,Pn,Pn+1) est le produit des partitions SSD(P1, ... ,Pn) et PI(Pn+1).

Par définition de l'opérateur "produit de partitions", l'ordre des opérandes est sans importance. Donc SSD(P1,P2)=SSD(P2,P1). De plus, SSD(P1,SSD(P2,P3)) = SSD(SSD(P1,P2),P3). Par exemple:



Δ(Xi)



SSD(P1,P2,P3,P4,P5)
est de rang 11

[def III-7]

Soit une fonction attribut Xi du type t; les blocs de la partition de Δ (Xi) par la fonction SSD(P1, ... ,Pn) où P1,P2,...Pn sont tous les prédicats de PP(Xi), sont appelés les SOUS-DOMAINES STABLES DE Xi, que nous notons SSD(Xi).

Par extension, nous appelons SSD(P1, ... ,Pn) la partition en sous-domaines stables de Xi relativement à un PP(Xi) courant et nous le notons SSD(Xi) si aucune ambiguité n'est possible. Le nombre de SSD de

l'attribut Xi, où PP(Xi) contient n prédicats est au plus égal à 2**n.

Par exemple les partitions de $\Delta(X1), \Delta(X2), \Delta(X3)$ relatives à PP(X1),PP(X2),PP(X3) vues auparavant sont:

SSD(X1)={[INF,10[,[10,15],]15,20],]20,30],]30,SUP]}
SSD(X2)={[INF,100[,[100,SUP]}
SSD(X3)={{OUI},{NON},{SURS},AUTRES}

[Proposition III-2]

Soit Xi une fonction du type t, tout prédicat élémentaire Pi sur Xi peut être réécrit sous forme d'une disjonction de prédicats élémentaires Pi1 v ... v Pij v ... Pin, telle que :

1) l'extension de tout prédicat Pij, $1 \leq j \leq n$ est un sous domaine stable

2) - $\bigcup_j$ EXT(Pij) est égal à EXT(Pi)

Nous nommons "prédicat élémentaire sur un sous-domaine stable" tout prédicat élémentaire dont l'extension est un sous-domaine stable.

[Lemme III.1]

Toute assertion représentant une DIA peut être remplacée par un ensemble d'assertions "normalisées" (cf. def. III-5), composées exclusivement par des prédicats élémentaires sur des sous-domaines stables.

Preuve : Par conjugaison de [Proposition III.1] et de la règle d'équivalence logique [KOW79]

"(U $\wedge$ (X v Y) —> Z) <—> (U $\wedge$ X —> Z) (U $\wedge$ Y —> Z)"

En effet, étant donné qu'un "et" logique relie les assertions

représentant un ensemble de DIAs, l'assertion $U \wedge (X \vee Y) \longrightarrow Z$ est équivalente à l'ensemble d'assertions :

$$U \wedge X \longrightarrow Z$$

$$U \wedge Y \longrightarrow Z$$

[déf. III-8]

Une SDIA est une Dépendance Inter-Attribut qui utilise uniquement des prédicats élémentaires sur les sous-domaines stables.

Ainsi, l'utilisation de SSD(X1), SSD(X2) et SSD(X3) permet de réécrire les trois DIAs :

A1) $X1(t) \in [10,15] \vee X1(t) \in]15,20] \longrightarrow X3(t) \in \{OUI\}$

A2) $X1(t) \in [10,15] \longrightarrow X3(t) \in \{NON\}$

A3) $(X3(t) \in \{OUI\} \vee X3(t) \in \{SURS\}) \wedge X2(t) \in [100,SUP] \longrightarrow X1(t) \in]20,30]$

Finalement des SDIAs sont obtenues en "normalisant" les assertions précédentes (cf. déf. III-5).

A'1) $X1(t) \in [10,15] \longrightarrow X3(t) \in \{OUI\}$

A"1) $X1(t) \in]15,20] \longrightarrow X3(t) \in \{OUI\}$

A'2) $X1(t) \in [10,15] \longrightarrow X3(t) \in \{NON\}$

A'3) $X3(t) \in \{OUI\} \wedge X2(t) \in [100,SUP] \longrightarrow X1(t) \in]20,30]$

A"3) $X3(t) \in \{SURS\} \wedge X2(t) \in [100,SUP] \longrightarrow X1(t) \in]20,30]$

La présentation des assertions sous forme de SDIAs permet de mettre

en évidence des D-inconsistances ; ainsi, il est évident que A'1 et A'2 ne peuvent être simultanément satisfaites sauf si [10,15], sous-domaine de X1, est interdit. De même, A'1 et A'3 interdisent l'existence de toute p-occurrence t telle que X1(t) ∈ [10,15] et X2(t) ∈ [100,SUP].

III.3 - SDIAs ET LOGIQUE PROPOSITIONNELLE

Les sous-domaines stables de la fonction attribut Xi du type t sont tous distincts, déterminables statiquement et en nombre fini. Nous pouvons donc associer un nom unique à chacun d'eux. Ainsi, nous les nommons δi1, ... δipi. D'autre part, il existe une correspondance bi-univoque entre les prédicats élémentaires sur les sous-domaines stables et les sous-domaines stables eux-mêmes. En conséquence, nous pouvons également leur associer un nom unique : par exemple, on nommera dij le prédicat Xi(t) ∈ δij. La réécriture des SDIAs en utilisant les noms de prédicats fournit des formules de la logique propositionnelle.

Ainsi, dans notre exemple nous avons :

δ11 = [INF,10[; δ12 = [10,15] ; δ13 =]15,20] ; δ14 =]20,30] ; δ15 =]30,SUP]

δ21 = [INF,100] ; δ22 =]100,SUP]

δ31 = {OUI} ; δ32 = {NON} ; δ33 = {SURS} ; δ34 = AUTRES

Et, en utilisant la convention δij/dij, c'est-à-dire le prédicat sur le sous-domaine stable δij est appelé dij, nous pouvons réécrire les SDIAs sous la forme suivante :

A'1 : d12 —> d31

A"1 : d13 —> d31

A'2 : d12 —> d32

A'3 : d31 ∧ d22 —> d14

A"3 : d32 ∧ d22 —> d14

On a là un ensemble de formules de la logique propositionnelle dont les littéraux sont des identificateurs de prédicats. De ce fait, les vérifications qui ne font pas intervenir explicitement les variables (par exemple la consistance ou la D-inconsistance, définie en III.6) peuvent être effectuées avec les méthodes de la logique propositionnelle, qui sont décidables. Les vérifications liées spécifiquement aux objets de la base (par exemple la vérification de validité d'une occurrence) restent classiques, mais on peut réduire (cf. III.7 et cf. III.17) le nombre des vérifications dynamiques et leur forme est également simplifiée.

L'inconsistance peut être traitée de façon classique en ajoutant l'ensemble d'assertions qui traduisent la propriété d'exclusion entre les sous-domaines stables d'une même fonction attribut (pour Xi {dij —> ~ dik} avec $j \neq k$). Nous obtenons ainsi les axiomes propres du type pour lequel les DIAs ont été définies. Nous ne nous occupons pas spécifiquement du problème de l'inconsistance, qui sera traité comme un cas particulier de D-inconsistance (III.6). Nous introduisons maintenant les notions de semi-domaines et D-classes avant de définir la D-inconsistance.

III.4 - SEMI-DOMAINES ET D-CLASSES

Dans ce paragraphe, nous présentons les définitions et propositions nécessaires à l'utilisation des classes d'équivalence qu'un ensemble de SDIAs définit sur les occurrences d'un type.

[déf. III-9]

Le domaine de définition d'un type t caractérisé par les fonctions attributs X1, X2, ... Xn est le produit cartésien de ses codomaines Δ (X1) x Δ (X2) x ... x Δ(Xn).

Chaque Δ(Xi) $1 \leq i \leq n$, est considéré comme décomposé en sous-domaines stables $\delta i1$, $\delta i2$, ..., δipi.

[déf. III-10]

Un semi-domaine du type t caractérisé par n fonctions attributs est un n-uplet ($\delta 1j$, ... δik ...,δnl) ou son i-ème élément est soit un sous-domaine stable de Xi soit Δ(Xi) lui-même, noté "*".

[déf. III-11]

Une composante explicite d'un semi-domaine ($\delta 1j$ ... δnl) est un élément différent de"*". Le nombre de composantes explicites d'un semi-domaine constitue son degré.

Exemples de semi-domaines du type t (3 attributs)

```
D1 = ([10,15], *, *)            /*degré 1 */
D2 = ([10,15], ]100,SUP], *)    /*degré 2 */
```

[déf. III-12]

Une D-classe est un semi-domaine dont tous les éléments sont des composantes explicites.

Exemples de D-classes :

```
D3 = ([10,15], ]100,SUP], {OUI})
D4 = (]15,20], [INF,100], {OUI})
```

[déf. III-13]

Un semi-domaine D = (δ1j, ..., δnl) contient un semi-domaine D' = (δ'1f, ..., δ'np) ssi pour tout i ∈ [1,n] δ'ir ⊂ δik, c'est-à-dire soit r = k soit ik = *. Nous notons D' ⊂ D.

On peut dire aussi que D' ⊂ D ssi toute composante explicite de D est aussi une composante explicite de D'.

[lemme III-2]

Si D ⊂ D' et D' ⊂ D'' alors D ⊂ D''

Preuve :par définition même de ⊂ .

[déf. III-14]

Un semi-domaine D = (δ1, ..., δn) contient une occurrence t = (x1, ..., xn) ssi pour tout i ∈ [1,n] xi ∈ i. Nous écrivons t ∈ D.

En sachant que les sous-domaines stables d'une fonction attribut sont disjoints entre eux, on peut conclure que :

[Proposition III-3]

Toute occurrence d'un type t est contenue dans une et une seule D-classe du type.

Exemples

D2 $\subseteq$ D1

D3 $\subseteq$ D1, D3 $\subseteq$ D2

t = (12, 102, OUI) $\in$ D3

/* D3 est l'unique D-classe qui contient t */

[Proposition III-4]

Toutes les occurrences d'un type t, appartenant à une même D-classe ont le même comportement vis-à-vis de l'ensemble des SDIAs.

En effet, les prédicats d'une SDIA sont de la forme Xi(t) $\in \delta_i$ et ces mêmes δi constituent les composantes d'une D-classe. Chaque prédicat a donc la même valeur de vérité pour toutes les occurrences d'une même D-classe.

[Proposition III-5]

Soit le type t caractérisé par n fonctions attributs X1 ... Xn; soit SSD(Xi) l'ensemble des sous-domaines stables de Xi. Tout élément du produit cartésien SSD(X1) x .... x SSD(Xn) est une D-classe de t.

[Lemme III.3]

Une D-classe du type t est une classe d'équivalence du domaine de définition de t en considérant la relation "avoir la même valeur de vérité par rapport aux SDIAs ".

Preuve par conjugaison des propositions III-4 et III-5.

Une D-classe (δ1k, ..., δnp) définit une interprétation des assertions, avec le prédicat dij vrai si le sous-domaine stable σij appartient à la D-classe, et faux autrement. Elle est un modèle des SDIAs ssi l'ensemble des assertions est vrai pour l'interprétation en cause.

[déf. III-15]

Une D-classe est valide ssi elle est un modèle des SDIAs.

En conséquence, toute occurrence qui appartient à une D-classe valide est elle aussi un modèle des SDIAs. Elle est dite occurrence valide.

[déf. III-16]

Un semi-domaine D est un semi-domaine valide ssi il existe une D-classe valide D' telle que D' $\subset$ D.

Toute occurrence, D-classe ou semi-domaine qui n'est pas valide est dit non-valide.

Il faut remarquer qu'une D-classe valide ne contient que des occurrences valides, tandis qu'un semi-domaine valide, s'il doit contenir par définition des occurrences valides, peut aussi contenir des occurrences non valides.

[Lemme III-4]

Si D est un semi-domaine valide, tout semi-domaine D', tel que D $\subset$ D' , est lui aussi valide .

Preuve par definition même de $\underline{C}$.

Soit D un semi-domaine non valide; a priori, on ne peut rien dire sur la validité de tout semi-domaine D', tel que D $\underline{C}$ D'. Cependant si D est non valide , tout D', tel que D' $\underline{C}$ D est un semi-domaine non valide.

Le nombre de D-classes d'un type t est fini car le nombre d'éléments de SSD(X1) x ... x SSD(Xn) est fini. En consequence, on peut déterminer de façon exhaustive, les D-classes valides et les D-classes non-valides du type t.

III-5 SEMI - DOMAINES D'UNE SDIA

Chaque SDIA détermine deux familles de semi-domaines : celle des semi-domaines valides et celle des semi-domaines non valides. Ainsi, pour l'assertion d12 —> d31, la famille de semi-domaines valides est composée de (δ12, *, δ31) ; la famille de semi-domaines non valides étant (δ12, *, δ32), (δ12, *, δ33), (δ12, *, δ34), où δ32, δ33, δ34 sont les autres semi-domaines stables de X3.

[déf. III-17]

Soit ij un sous-domaine stable de la fonction Xi; nous nommons sous- domaines stables complémentaires de δij les sous-domaines stables de Xi distincts de δij. Nous notons leur ensemble /δij.

De même, les sous-domaines stables complémentaires d'un ensemble de sous-domaines stables d'une fonction attribut Xi sont les sous-domaines stables de Xi, différents de ceux contenus dans le premier ensemble.

Ainsi

$$/\delta 13 = \{\delta 12, \delta 11, \delta 14, \delta 15\}$$

$$/(\delta 13, \delta 14) = \{\delta 12, \delta 11, \delta 15\}$$

Par analogie, on notera /dij la négation du prédicat dij. /dij peut toujours s'écrire comme une disjonction de dik, $k \neq j$.

[Proposition III-6]

A partir d'une SDIA dont l'antécédent est composé de n prédicats, et dont le conséquent est composé de m prédicats (sur une même fonction attribut) on déduit :

i - m semi-domaines valides de degré n + 1 ;

ii - p semi-domaines non valides de degré n + 1 : p est la cardinalité du sous-domaine stable complémentaire des m sous-domaines stables concernés par les prédicats du conséquent.

Par exemple, à partir de

$$d32 \wedge d22 \longrightarrow d14 \vee d11 \vee d13$$

on déduit :

- trois semi-domaines valides

$$(\delta 14, \delta 22, \delta 32)$$
$$(\delta 13, \delta 22, \delta 32)$$
$$(\delta 11, \delta 22, \delta 32)$$

- deux (p = 2) semi-domaines non valides

$$(\delta 15, \delta 22, \delta 32)$$
$$(\delta 12, \delta 22, \delta 32)$$

Cet exemple met en évidence que n des n + 1 composantes explicites sont les sous-domaines stables qui apparaissent en antécédents de la SDIA ; il montre aussi que la (n + 1)ième composante est un sous-domaine stable de la fonction attribut qui apparaît dans le conséquent de la SDIA : selon que le sous-domaine stable appartient ou non à la SDIA, le semi-domaine est valide ou non-valide. On peut écrire :

semi-domaines valides : $(\delta 1j, \delta 22, \delta 32)$ $\quad j \in \{1, 3, 4\}$

semi-domaines non valides : $(\delta 1i, \delta 22, \delta 32)$ $\quad i \in \{2, 5\}$

NOTE : les semi-domaines valides que la proposition III-6 permet de déduire pour une SDIA ne sont pas les seuls semi-domaines valides déductibles. En effet, toute assertion P—>Q est aussi satisfaite par ~P. Cependant, dans le contexte de la détermination de la D-inconsistance, des redondances et de l'inconsistance, ces semi-domaines ne nous intéressent pas. Dans une certaine mesure, ils ne sont pas "concernés" directement par l'assertion.

III.6 - D-INCONSISTANCE

Nous nous plaçons dans l'hypothèse où l'utilisateur considère qu'il existe des objets de la base vérifiant ses assertions, et plus précisément leurs antécédents. Ainsi un utilisateur qui spécifie l'assertion AGE(t) > 30 —> SAL(t) $\geq$ 4000, considère qu'il existe effectivement des objets tels que AGE(t) > 30. Si d'autres assertions du type interdisent ces objets, on peut penser que l'utilisateur n'a pas été conscient des conséquences de l'ensemble des assertions. Une telle situation peut aussi résulter de l'écriture non concertée d'assertions par plusieurs utilisateurs. Elle peut encore refléter une simple erreur d'écriture des assertions.

[déf. III-18]

Une SDIA est D-inconsistante à l'intérieur d'un ensemble S de SDIAs ssi tout semi-domaine modèle de son antécédent est un semi-domaine non valide (pour l'ensemble de SDIAs).

On peut aussi la formuler de la manière suivante :

[déf. III-19]

Une SDIA P —> Q est D-inconsistante dans un ensemble de SDIAs S ssi S ∧ P est inconsistante, où P est une conjonction de prédicats élémentaires sur les sous-domaines stables.

[déf. III-20]

On appelle degré de D-inconsistance d'une SDIA D-inconsistante, le nombre de prédicats de son antécédent.

Avec la D-inconsistance, nous voulons donc mettre en évidence l'existence simultanée d'assertions de la forme (P —> Q) et (~P), la première étant en général directement spécifiée par l'utilisateur et la deuxième pouvant être une conséquence logique de l'ensemble de SDIAs. Cette situation n'est pas l'inconsistance logique puisque P —> Q est aussi satisfait pour ~P.

La D-inconsistance de l'assertion P —> Q est une conséquence de l'un des quatre schémas suivants :

1 - P —> Q
    P —> ~Q /*Cette SDIA peut être éventuellement une
              conséquence logique des SDIAs*/

En effet, de (P —> Q) et (P —>~Q), on <u>déduit</u> (~P).

Dans notre exemple, il y avait explicitement A'1 : d12 —> d31 et A'2 : d12 —> d32. Puisque d31 —> ~d32, on peut déduire —> ~d12 : aucun semi-domaine valide des SDIAs n'est un modèle de A'1. De même pour A'2.

2 - P—> Q et Q —> L et L —> M
P —> ~M /*éventuellement une conséquence logique des SDIAs*/

Ce cas se ramène au précédent en sachant que P —> M est une conséquence logique de la famille de SDIAs (P —> Q, Q —> L, L —> M).

3 - P —> Q
—> ~P /*Cette SDIA peut être une conséquence logique*/

De ces deux SDIAs, on déduit (—> ~P) [KOW79]. Exemple :

Soient les trois SDIAs spécifiées par l'utilisateur :

A1 : dij —> dkl

A2 : dkl —> dfm

A3 : dkl —> dfl

Ainsi, de A2 et A3, on déduit A4 : —> ~dkl, et finalement de A1 et A4, on déduit A5 : —> ~dij.

La généralisation de ce schéma étant celui où P —> Q est une assertion déduite de l'ensemble de SDIAs.

4 - P —> Q
—> ~P /*éventuellement une conséquence logique*/

Exemple :

Soient les trois assertions :

A1 : dij —> dkl

A2 : dij —> dfi

A3 : dij —> dfl

De A2 et A3, on déduit A4 : —> ˜dij.

La généralisation de ce cas étant celui où A2 et/ou A3 ne sont pas directement spécifiées par l'utilisateur, mais sont déduites de l'ensemble de SDIAs.

[déf. III-21]

Un ensemble S de SDIAs est D-inconsistant ssi il existe au moins une SDIAs Ai ∈ S, telle que Ai est D-inconsistante.

Lorsqu'une D-inconsistance est détectée, l'utilisateur doit explicitement choisir entre supprimer une ou plusieurs assertions parmi celles qui induisent la D-inconsistance ou accepter d'exclure certains semi- domaines.

III.7 - UTILISATION DES NOTIONS DE SEMI-DOMAINE ET DE D-CLASSE

Les quatres propiétés sous-jacentes à toutes les conclusions de ce paragraphe sont:

A- le nombre de D-classes d´ un type t est fini et connu;

B- la validité d´une D-classe est décidable;

C- si D est un semi-domaine valide tout D´, tel que D $\subseteq$ D´, est valide;

D- si D est un semi-domaine non valide, tout D´, tel que D´ $\subseteq$ D est non valide.

III.7.1 - DETECTION DES D-INCONSISTANCES

Pour déterminer si une SDIA Ai: P —> Q est D-inconsistante à l´intérieur d´ un ensemble S des SDIAS, on peut:

i- soit, vérifier la cohérence de S $\wedge$ P : si S P est logiquement incohérent, on est en présence d´une assertion D-inconsistante;

ii- soit, vérifier si pour tout semi-domaine valide D´, déduit à partir de Ai, il existe une D-classe valide D´, telle que D $\subseteq$ D´ .

III.7.2 - ELIMINATION DES REDONDANCES

Les D-classes d´un p-type t dépendent uniquement et exclusement du nombre de prédicats élémentaires distincts que l´ensemble de DIAs contient (cf. determination de sous-domaines stables). La multi-spécification d´une règle est donc indifférente pour la determination des D-classes. En consequence, les redondances, ou multi-spécification, sont éliminées implicitement.

III.7.3 - VERIFICATION DE LA VALIDITE D'UNE OCCURRENCE

Pour vérifier si une occurrence ti d'un type t est valide il faut :

i- déterminer sa D-classe : cette partie est obligatoirement dynamique.

ii- déterminer si la D-classe est valide : ce travail peut être fait de façon statique.

III.7.4 - MODIFICATION D'UNE OCCURRENCE

Soit ti une occurrence appartenant à la D-classe valide D. Soit D' la D-classe de ti après modification. Deux cas sont possibles:

i- D=D' :dans ce cas il n'y a rien à faire.

ii- D≠D' :il faut detérminer si D' est valide.

III.7.5 - INSERTION D'UNE NOUVELLE ASSERTION

L'insertion d'une nouvelle assertion Ai detérmine, en général, une nouvelle partition en sous-domaines stables du codomaine d'une ou plusieurs fonctions attributs. Après l'insertion de Ai les D-classes peuvent donc être différentes. Cependant les seules D-classes dont la validité peut être changées sont celles qui appartiennent aux semi-domaines non valides déduits de la nouvelle SDIA (cf. III.5). Par exemple, l'introduction de l'assertion A4 : X1(t) $\geq$ 12 —> X3(t) ="SURS", dans l'ensemble de trois DIAs qui ont servi d'exemple au long de ce chapitre induit la décomposition du sous-domaine stable δ12 = [10,15] de X1 (cf. III.2) en δ'12=[10,12[et δ"12=[12,15]; en conséquence, toute D-classe appartenant au semi-domaine D=(δ12,*,*) engendre, après l'insertion de A4, deux D-classes. La validité de chacune des nouvelles D-classes est la même que la D-classe de départ sauf pour celles qui

appartiennent aux semi-domaines non valides déduits de la nouvelle assertion. Ainsi, D1=(δ12,δ22,δ31) engendre D'=(δ'12, δ'22,δ31) et D"=(δ"12,δ22,δ31); les semi-domaines non-valides déduits de A4 sont (δ'12,*,δ31), (δ'12,*,δ32) et (δ'12,*,δ34) : indépendamment de la validité de D1 , D" est non valide; par contre, D' a la même valeur de vérité que D1.

III.8 - SOUS-DOMAINES STABLES GENERALISES

La méthode proposée au paragraphe III.7 pour déterminer si l'assertion Ai est D-inconsistante est très générale ; et surtout, elle ne fournit à l'utilisateur aucun renseignement sur les assertions qui sont directement en cause dans la D-inconsistance de Ai. Parallèlement, nous considérons que le nombre de D-classes d'un p-type peut rendre inefficace la méthode proposée pour déterminer la validité d'une p-occurrence. Dans cette deuxième partie du travail, nous proposons une méthode plus efficace basée sur la représentation des SDIAs sous forme d'un graphe dont les noeuds sont des sous-domaines stables généralisés.

L'existence de SDIAs, dont l'antécédent est une conjonction de prédicats élémentaires sur les sous-domaines stables, ainsi que le besoin de traitement uniforme des différentes catégories de SDIAs, exige la généralisation du concept de sous-domaine stable [cf. III-2]. Ainsi,

[déf. III-22]

Soit le p-type t caractérisé par n fonctions attributs X1, ..., Xn. Une fonction attribut généralisée Y de t est une fonction construction [BAC78] de m $\leq$ n fonctions attributs de t.

Notations et rappels

- nous notons Yikl la fonction construction des fonctions attributs Xi, Xk, Xl. Ainsi,

$$Yikl = [Xi, Xk, Xl]$$

- par définition [BAC78] :

$$Yikl(t) = [Xi, Xk, Xl]\ (t)$$
$$[Xi, Xk, Xl]\ (t) = Xi(t) \times Xk(t) \times Xl(t)$$

où "x" est le symbole de produit cartésien.

- nous nommons degré de Y, le nombre de fonctions attributs à partir desquelles Y est construite. Ainsi, le degré de Y12 = [X1, X2] est deux.

- par définition, la valeur de Yi(t) est égale à celle de Xi(t) ; par conséquent, tous les résultats déductibles sur le codomaine de Xi sont intégralement applicables à celui de Yi ; et, nous employons indifféremment Xi ou Yi.

[déf. III-23]

Soit Yi ... k une fonction attribut généralisée du p-type t ; un sous-domaine généralisé de Yi ... k est un élément de SSD(Xi)x ... x SSD(Xk), où SSD(Xj) est l'ensemble des sous-domaines stables de Xj.

Par exemple, (δ12, δ31) est un sous-domaine stable généralisé de la fonction attribut généralisée Y13, où Y13 = [X1, X3].

L'ensemble de sous-domaines stables généralisés de la fonction Yi ... k est donc le produit cartésien des ensembles SSD(Xi), ..., SSD(Xk). Par analogie, nous notons SSDG(Yi ... k) l'ensemble de sous-domaines stables généralisés de Yi ...k :

$$SSDG(Yi \ldots k) = SSD(Xi) \times \ldots \times SSD(Xk)$$

Soit $x = (\delta ij, \ldots, \delta kl)$ un sous-domaine stable généralisé de Yi ... k de degré m,

- on appelle composantes élémentaires de x les sous-domaines stables $\delta ij, \ldots, \delta kl$.

- on appelle composantes de x les sous-domaines stables généralisés x', tels que toute composante élémentaire de x' est une composante élémentaire de x.

- nous notons /x les sous-domaines stables généralisés complémentaires [SAL84] de x. Ainsi,

$$/x = SSDG(Yi \ldots k) - \{x\}$$

A tout sous-domaine stable généralisé $x = (\delta ij, \ldots, \delta kl)$ de la fonction Yi ... k est associé un prédicat p, tel que p est vrai ssi Yi ... k (t) $\in$ x. Nous notons px le prédicat associé au sous-domaine stable généralisé x. La définition de sous-domaine stable généralisé conduit à:

[Proposition III-7]

Le prédicat px associé à $x = (\delta ij, \ldots, \delta kl)$ est la conjonction de prédicats élémentaires sur les sous-domaines stables [cf. III-2] qui sont en relation bi-univoque [cf. III.3] avec les composantes élémentaires de x :

$$px = dij \wedge \ldots \wedge dkl$$

Nous nommons prédicats élémentaires généralisés, les prédicats associés aux sous-domaines stables généralisés.

[Théorème III-1] - Soit x = (δij, ..., δkl) un sous-domaine stable généralisé de Yi ... k ; px est vrai ssi pour toute composante x´ de x, px´ est vrai.

PREUVE : par conjugaison des définitions de composante et prédicat élémentaire généralisé avec l´axiome A ∧ B ∧ C ∧ ... |- A ∧ B ∧ C ∧ ... -> A [KOW79].

Soit le p-type t caractérisé par n fonctions attributs X1, ..., Xn ; soit Y une fonction attribut généralisée de t, et soient xi et xj, i ≠ j , deux sous-domaines stables généralisés de Y. Les axiomes que les sous-domaines stables généralisés de t doivent satisfaire sont :

[AXIOME 1] : pxi —> ~pxj
[AXIOME 2] : pxi —> px´i où x´i est une composante de xi

Soient un sous-domaine stable généralisé x et son prédicat px; déterminer si la base de données est susceptible de contenir des occurrences ti, telles que px est vrai équivaut à déterminer si le semi-domaine D, dont les composantes explicites sont les composantes élémentaires de x, est valide. Ainsi,

[Proposition III-8]

Un sous-domaine stable généralisé peut être représenté par le semi-domaine D ayant comme composantes explicites les composantes élémentaires de x. Nous notons Dx le semi-domaine qui représente x. La relation entre x et Dx est bi-univoque.

Rappels

Soit Yi une fonction attribut généralisée de degré un :

- SSDG(Yi) = SSD(Xi)

- pour tout x ∈ SSDG(Yi) px est un prédicat élémentaire sur les sous-domaines stables [cf. III.2].

- tout élément x de SSDG(Yi) est représentable par un semi-domaine de degré un.

Nous notons x/Dx/px le sous-domaine stable généralisé x, le semi-domaine qui le représente et le prédicat qui est associé à x (et à Dx).

III.9 - RELATIONS DE COMPATIBILITE ET D'EXCLUSION ENTRE SEMI-DOMAINES

Soient les semi-domaines Di et Dj. Di et Dj sont unifiables ssi il existe un semi-domaine D, tel que :

i - D $\subseteq$ Di
ii - D $\subseteq$ Dj
iii - Pour tout D', tel que D' $\subseteq$ Di et D' $\subseteq$ Dj, D' $\subseteq$ D.

Nous notons Dij le semi-domaine qui unifie Di et Dj.

[déf. III-24]

Soient les semi-domaines Di et Dj. Une relation de compatibilité C est établie entre Di et Dj ssi il existe un semi-domaine Dij qui unifie Di et Dj , et Dij est un semi-domaine valide.

[déf. III-25]

Soient les semi-domaines Di et Dj; une relation d'exclusion E est établie entre Di et Dj si Di et Dj ne sont pas unifiables ou si Dij est un semi-domaine non valide.

En sachant qu'il existe une relation bi-univoque entre un semi-domaine et le sous-domaine stable généralisé qu'il représente, nous utilisons aussi l'expression "relation de compatibilité (ou d'exclusion) entre sous- domaines stables généralisés.

Lorsqu'on dit qu'il existe une relation R (compatibilité ou exclusion) entre Di et Dj, nous disons implicitement que l'antécédent de R est Di et que son conséquent est Dj. Les notions d'antécédent et de conséquent d'une relation sont une conséquence de la sémantique de l'implication (—>). Par exemple, soit l'assertion Ai: d12 —> d31 il existe une relation de compatibilité entre Di = (δ12,*,*) et Dj = (*,*,δ31), car (δ12,*,δ31) est valide (cf. III.5). Toute occurrence ti, telle que Xi(ti) $\in$ δ12 et X3(ti) $\in$ δ31, est donc valide. Cependant, si toute occurrence tj, telle que X1(tj) $\in$ δ12 ne peut être valide que si X3(tj) $\in$ δ31 une occurrence tf , telle que X3(tf) $\in$ δ31 peut l'être sans que X1(tf) $\in$ δ12.

III.10 - CATEGORIES DE RELATIONS DE COMPATIBILITE

La sémantique de l'implication, ainsi que les formes possibles des SDIAs, nous permet de distinguer trois catégories de relations de compatibilité.

C5 - Relation de compatibilité de type C5

[déf. III-26]

Soient Dx et Dy deux semi-domaines représentant respectivement les sous-domaines généralisés x et y. La relation entre Dx et Dy est de type C5 (noté 5) ssi :

i - elle est une relation de compatibilité
ii - pour tout sous-domaine stable généralisé z, tel que z $\mathcal{G}$ /y, la relation entre Dx et Dz est une relation d'exclusion.

Par exemple, une SDIA, ayant un seul prédicat en conséquent, engendre une relation de type C5 entre les semi-domaines qui représentent les sous-domaines stables généralisés associés respectivement au prédicat de l'antécédent et au prédicat du conséquent.

C4 - Relation de compatibilité de type C4

[déf. III-27]

Soient Dx et Dy deux semi-domaines représentant respectivement les sous-domaines stables généralisés x et y. Une relation entre Dx et Dy est de type C4 (noté 4) ssi

i - elle est une relation de compatibilité
ii - il existe zi $\mathcal{G}$ /y, tel que la relation entre Dx et Dzi est une relation de compatibilité

iii - pour tout zj ∈ /y, la relation entre Dx et Dzj est soit une relation d'exclusion, soit une relation de compatibilité.

Ces relations sont engendrées par des SDIAs dont le conséquent est une disjonction de prédicats. Par exemple de px —> δ12 v δ13 on déduit des relations de type C4 entre Dx et (δ2,*,*) et entre Dx et (δ13,*,*) ; on déduit encore des relations d'exclusion entre Dx et (w,*,*), où w ∈ / (δ12, δ13).

Note : Contrairement au type C5, si la relation entre Dx et Dy est de type C4, on sait qu'il existe au moins une autre relation de type C4 entre Dx et Dz où z ∈ /y.

C3 - Relation de compatibilité de type C3.

[déf. III-28]

Soient Dx et Dy deux semi-domaines, représentant respectivement les sous-domaines stables généralisés x et y ; une relation entre Dx et Dy est de type C3 (noté 3) ssi :

i - elle est une relation de compatibilité
ii - il existe au moins un z ∈ /y tel qu'on ne connaît pas le type de la relation entre Dx et Dz.

La relation C3 est celle qui contient le moins d'information : elle indique seulement qu'il y a une relation de compatibilité entre Dx et Dy. En général, elle est transitoire, c'est-à-dire des informations supplémentaires peuvent la transformer en C4 ou en C5.

[Proposition III-9]

L'existence entre Dx et Dy d'une relation de type C5 (ou C4 ou C3) garantit, a priori, que la relation entr Dy et Dx est, au moins, de type C3.

[Proposition III-10]

L'existence entre Dx et Dy d'une relation d'exclusion (noté 0) garantit que la relation entre Dy et Dx est, elle aussi, une relation d'exclusion.

Les types des relations qui peuvent exister entre semi-domaines, ainsi que les propositions III-9 et III-10, permettent de synthétiser les informations implicites et explicites déductibles de l'implication logiques .

III.11 - RELATIONS ENTRE SOUS-DOMAINES STABLES GENERALISES

Dans ce paragraphe, nous nous intéressons aux relations intrinsèques entre des sous-domaines stables généralisés, c'est-à-dire aux relations indépendantes de tout ensemble particulier de SDIAs. Ces relations de compatibilité et d'exclusion sont directement déductibles d'AXIOME 1 et d'AXIOME 2 [cf. III-8].

Soient xi et xj deux sous-domaines stables généralisés, tels que xi $\mathcal{G}$ / xj, et donc xj $\mathcal{G}$ /xi ; soit x'i une composante [cf. III-8] de xi; et soient Dxi, Dxj et Dx'i les semi-domaines qui représentent xi, xj et x'i. Ainsi

- il existe une relation d'exclusion entre Dxi et Dxj, et vice-versa [cf. AXIOME 1] et [Proposition III-10]

- il existe une relation C5 entre Dxi et Dx'i, et donc, il existe une relation d'exclusion entre Dxi et Dw, où w $\mathcal{G}$ /x'i [AXIOME 2]

- il existe une relation de type C4 entre Dx'i et Dxi [Proposition III-9] et [def III-27].

III.12 - RELATIONS DEDUITES D'UNE SDIA

Les relations de compatibilité et d'exclusion déductibles d'une SDIA Ai dépendent de sa forme. Nous distinguons deux grandes catégories de SDIAs :

CAT1 - Cette catégorie de SDIAs est composée par des assertions ayant un seul prédicat en conséquent. Par exemple :

$$px \longrightarrow dkl$$

où px est un prédicat élémentaire généralisé, [cf. III-9] ; px est donc soit dik, soit dik $\wedge$ dfm ...

Soient x, px et Dx respectivement un sous-domaine stable généralisé, son prédicat et le sous-domaine qui le représente. D'une assertion de CAT1 Ai : px —> dij, on déduit :

i - une relation de type C5 entre Dx et Di, telle que Di est de degré un et sa composante explicite est δij

ii - des relations d'exclusion entre Dx et Dj, où Dj représente un semi-domaine de degré un et sa composante explicite w $\in$ /δij ; et des relations d'exclusion entre Dj et Dx

iii - une relation de type C3 entre Di (cf. point i) et Dx.

En effet, le semi-domaine Dxi qui unifie Dx et Di est l'unique semi-domaine valide déduit de Ai (cf. III-5).

CAT2 - La forme des SDIAs qui appartiennent à cette catégorie est

$$px \longrightarrow dij \vee dik \vee \ldots$$

où px a la même signification qu'en CAT1. Les SDIAs de CAT2 ont donc une disjonction de prédicats dans le conséquent. En utilisant la notation x/px/Dx qui devient habituelle, nous pouvons dire que d'une assertion Ai : px —> dij v dik v ... ayant n prédicats dans le conséquent, on peut déduire :

i - n relations de type C4, respectivement entre Dx et chacun des semi-domaines Dk, Dj... qui représentent un des sous-domaines stables du conséquent

ii - des relations de type C3 entre chacun de ces mêmes n semi-domaines Dk, Dj ... et le semi-domaine Dx

iii - des relations d'exclusion entre Dx et tout semi-domaine D' représentant un w ∈ /(δij, δik,...) ; des relations d'exclusion entre tout D' et Dx.

III.13 - REPRESENTATION DES RELATIONS ENTRE SEMI-DOMAINES

La détection des D-inconsistances et des redondances de l'ensemble de SDIAs du p-type t, ainsi que la détermination de la validité des D-classes de t, est faite à l'aide d'un graphe dont les noeuds représentent les semi - domaines de t. En sachant qu'un semi-domaine Dx est en relation bi-univoque avec le sous-domaine stable généralisé x [cf. III-8], on peut aussi conclure qu'un noeud du graphe représente un sous-domaine stable généralisé. Nous identifions le noeud représentant Dx par les composantes explicites de Dx.

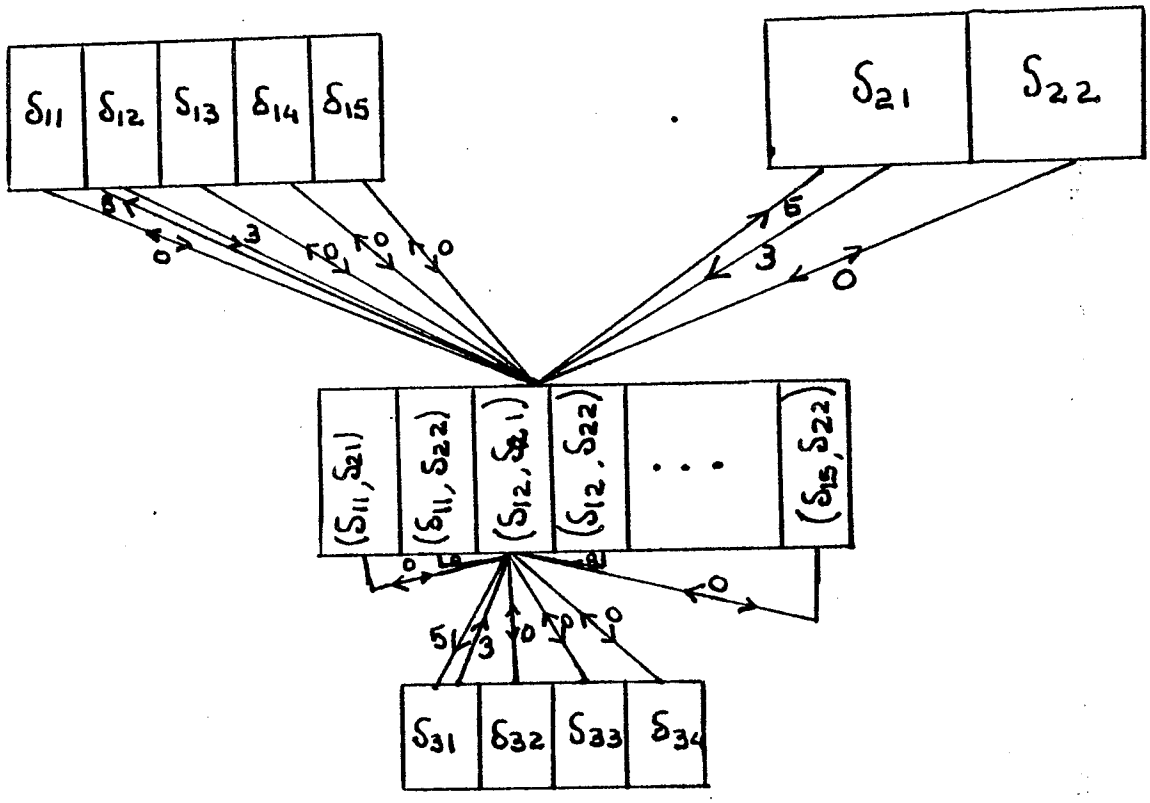
Les arcs du graphe représentent les relations de compatibilité et d'exclusion existantes entre les semi-domaines de t. Ainsi,

[Proposition III-11]

Soient Dx et Dy deux semi-domaines du p-type t ; soit R le type de la relation entre Dx et Dy ; la relation R est représentée par un arc dirigé et typé dont le père est le noeud qui représente Dx et le fils celui qui représente Dy.

Nous notons b = (Dx, Dy) l'arc entre Dx et Dy, et nous disons que b est de type R_x où R est soit E soit C3 ou C4 ou C5.

Ainsi, la représentation des relations intrinsèques au semi-domaine ($\delta 12$, $\delta 21$) et des relations déductibles de Ai : d12 ∧ d21 —> d31 permet de construire le graphe ci-dessous :



[Fig. III-1] - Représentation des relations intrinsèques à ($\delta 12$, $\delta 21$) et des relations déduites de d12 ∧ d21 —> d31

Par commodité, nous groupons les noeuds représentant les sous-domaines stables généralisés d'une même fonction Y.

Ce graphe permet de déduire [WAR62] un arc c entre les noeuds représentant respectivement ($\delta 12$,*,*) et (*,*,$\delta 31$). Les arcs du graphe représentant des relations, il faut donc déduire le type c.

III.14 - COMPOSITION SEQUENTIELLE D'ARCS

Dans ce paragraphe, nous nous plaçons dans le cas où il existe un arc a = (Dx, Dy) de type R1 et un arc b = (Dy, Dw) de type R2. A priori, on peut déduire un arc c = (Dx, Dw) [WAR62]. Cependant, les arcs représentant des relations entre semi-domaines, l'arc c ne doit être présent que s'il existe une relation de compatibilité ou d'exclusion entre Dx et Dw. Dans ce cas, nous disons que c est obtenu par composition séquentielle de a et de b, et nous notons c = a.b. Dans le cas contraire, on ne fait rien.

[Proposition III-12]

Soient a = (Dx, Dy) et b = (Dy, Dw). La composition séquentielle de a et b engendre un arc c = (Dx, Dw) ssi à partir de SDIAs qui sont à l'origine de a et b on peut déduire une SDIA Ai : px —> p'w où px est le prédicat associé à x (de Dx) et p'w est soit pw soit ~pw.

Pour déterminer si à partir des arcs a = (Dx, Dy) et b = (Dy, Dw) on peut déduire un arc c, nous utilisons surtout les résultats de III-4, III-10 et III-11. Ainsi,

[Proposition III-13]

Soient a = (Dx, Dy) et b = (Dy, Dw) deux arcs représentant deux relations de compatibilité (C3 ou C4 ou C5). A partir de a et de b on déduit un arc c = (Dx, Dw), représentant lui aussi une relation de compatibilité.

En effet, si a et si b représentent des relations de compatibilité, il existe deux SDIAs A1 : px —> (py v ...) et A2 : py —> (pw v ...) ; de A1 et de A2 on déduit A3 :px ∧ py —> py (pw v...); de A3 on déduit px ∧ py —> pw v ... et finalement, on déduit px —> pw v ... : il existe donc une relation de compatibilité entre Dx et Dw.

[Proposition III-14]

Soit a = (Dx, Dy) un arc représentant une relation d'exclusion ; soit b = (Dy, Dw) un arc représentant une relation R entre Dy et Dw ; quel que soit le type de R, on ne peut jamais déduire un arc c = (Dx, Dy).

En effet, si a représente une relation d'exclusion, il existe une SDIA Ai : px —> ˜py ; l'arc b indique soit l'existence d'une SDIA A'2 : py —> pw v ..., soit l'existence d'une SDIA A"2 : py —> ˜pw.

De A1 et A2, on déduit A'3 : px ∧ py —> ˜py ∧ (pw v ...) qui nous permet de conclure que tout objet t satisfaisant l'antécédent A'3 (px ∧ py) est un objet non valide.

De même de A1 et A"2, on déduit A"3 : px ∧ py —> ˜py ∧ ˜pw : il n'y a aucun objet valide qui soit modèle de son antécédent.

Cependant, de A'3 ou de A"3, on ne peut rien conclure sur la validité d'un objet qui satisfait px ∧ pw : il peut exister un py' tel qu'un objet satisfaisant px ∧ py' ∧ pw soit valide, où py' est le prédicat associé au sous-domaine stable généralisé y' ∈ /y.

[Proposition III-15]

Soit b = (Dy, Dw) un arc représentant une relation d'exclusion ; soit R une relation entre Dx et Dy représentée par b = (Dx, Dy). Si le type de R est exclusion ou C3 ou C4, les arcs a et b ne permettent pas de déduire un arc c entre Dx et Dy.

La raison de cette proposition est la même que celle de la [proposition III-14] : on peut déduire que tout objet satisfaisant py ∧ pw est non valide ; donc l'objet satisfaisant px ∧ pw ∧ py est lui

aussi non valide; on ne sait rien sur les objets satisfaisant px ∧ pw ∧ py´, où y´ ∈ /y.

[Proposition III-16]

Soit a = (Dx, Dy) un arc de type C5 ; soit R une relation entre Dy et Dw, représentée par l´arc b = (Dy, Dw) ; la composition séquentielle de a et de b engendre un arc c (Dx, Dw) de même type que l´arc b.

En effet, si a est de type C5, il existe une SDIA A1 : px —> py : tout semi-domaine D, tel que D ⊂ Dx n´est valide que si D ⊂ Dy (implicitement ou explicitement) ; d´autre part, l´arc b permet de "conclure à l´existence d´une SDIA A2 : py —> (pw v ...) ; de A1 et A2 on déduit A3 : px ∧ py —> py ∧ (pw v ...).

La validité d´un objet t qui satisfait l´antécédent de A3 dépend de la validité (ou non validité) de Dyw obtenue par unification de Dy et Dw ; si Dyw est non valide, l´occurrence est non valide ; sinon elle est valide.

Ci-dessous, nous présentons de façon exhaustive les résultats obtenus par composition séquentielle des arcs a et b,

a	b	c = a.b	commentaire
5	5	5	dij -> dkl, dkl -> dfm ¦- dij -> dfm
	4	4	dij->dkl, dkl->dfm v dfj ¦- dij->dfm v dfj
	3	3	dij->dkl, dkl->dfm v ... ¦- dij->dfm v ...
	0	0	dij->dkl, dkl->~dfm ¦- dij-> ~dfm
4	5	3	dij->dkl v dkm, dkl->dfm ¦- dij->dfm v ...
	4	3	
	3	3	
	0	rien	
3	5	3	
	4	3	
	3	3	

	0	rien
0	5	rien
	4	rien
	3	rien
	0	rien

Ce tableau peut être résumé par :

p1 : 5.x = x

p2 : (3 ou 4).(5 ou 4 ou 3) = 3

p3 : (3 ou 4).0 = rien

p4 : 0.x = rien

A ces quatre propriétés nous ajoutons deux autres :

p5 : a.b $\neq$ b.a

p6 : a.(b.c) = (a.b).c

Ces deux dernières propriétés peuvent être déduites des quatre premières.

NOTE : Par la suite "rien" est noté 2. 2 est donc le type inconnu.

III.15 SYNTHESE D'ARCS

La détection de D-inconsistances et de redondances est faite en utilisant la propriété suivante :

P1 : Une SDIA engendre, au plus, un arc entre Dx et Dy (c'est-à-dire, un arc dont le père est le noeud qui représente Dx et le fils est celui qui représente Dy).

Ainsi, l'existence entre Dx et Dy de plusieurs arcs met en évidence une multi-spécification d'une même règle. Selon les types des arcs en présence, la multi-spécification indique un raffinement, une redondance ou une incohérence.

Nous nous intéressons donc à l'existence simultanée de deux ou plusieurs arcs ayant exactement le même père et le même fils : a = (Dx, Dy), b = (Dx, Dy)... Nous appelons synthèse des arcs a et b, l'arc c ayant le même père et le même fils des arcs a et b, et nous notons c = a x b.

Le type de l'arc c est une fonction des types des arcs a et b. Le tableau ci-dessous détermine le type de c en fonction de celui de a et de b :

a	b	c = a x b	commentaire
5	5	5	REDONDANCE
5	4	ERR	D-INCONSISTANCE. Les [def III-26] [def III-27] permettent de prévoir la synthèse d'un arc de type 0 avec un arc de type C4
5	3	5	RAFFINEMENT - C5 ayant une information plus riche est le dominant
5	0	ERR	D-INCONSISTANCE
4	4	4	REDONDANCE
4	3	4	RAFFINEMENT - C4 domine C3
4	0	ERR	D-INCONSISTANCE

3	3	3	REDONDANCE
3	0	ERR	D-INCONSISTANCE
0	0	0	REDONDANCE

On complète le tableau par la propriété de commutativité sous-jacente :

p2 : a x b = b x a

De ce tableau, on peut encore déduire que :

p3 : a x (b x c) = (a x b) x c

avec : ERR x y = ERR

III.16 - CONSTRUCTION DU GRAPHE DE VERIFICATION DE COHERENCE : DETERMINATION DES D-INCONSISTANCES DE DEGRE N ET DES REDONDANCES

Les étapes de la méthode que nous proposons pour détecter les redondances et les D-inconsistances de degré n d'un ensemble de SDIAs sont décrites ci-dessous. A la fin de la dernière étape, un graphe, appelé graphe de vérification de cohérence (noté GVC) est construit pour le p-type auquel l'ensemble de SDIAs appartient. Ainsi, il faut:

a - déterminer les sous-domaines stables de chacune des fonctions attributs : ces sous-domaines stables sont les sous-domaines stables généralisés de fonctions attributs généralisées de degré 1 du p-type (cf. III-8);

b - déterminer les sous-domaines stables généralisés x des fonctions attributs de degré ≥ 2 et, tels qu'il existe une SDIA ayant px en antécédent. Les noeuds du graphe représentent les sous-domaines stables généralisés obtenus en a et en b.

c - représenter les relations entre les sous-domaines stables obtenus en a et en b [cf. III-11].

d - représenter chacune des SDIAs [cf. III-12] et effectuer la synthèse d'arcs [cf. III-15] si nécessaire.

e - effectuer la fermeture du graphe obtenu en d [WAR62] en appliquant la composition séquentielle [cf. III-14] pour déterminer les types des nouveaux arcs. Cette étape peut exiger des nouvelles synthèses d'arcs.

Techniquement, le nombre de noeuds du graphe d'un p-type t ayant n fonctions attributs X1, ..., Xn est égal à

$$SSD(Xi)+m$$

où m est le nombre de sous-domaines stables généralisés obtenus dans l'étape b de la méthode proposée dans ce paragraphe.

III.17 - UTILISATION DU GRAPHE DE VERIFICATION DE COHERENCE

Le graphe de vérification de cohérence d'un p-type t est nécessaire dans 4 situations différentes :

- Détermination de la validité d'une D-classe

- Insertion d'une p-occurrence ti de t

- Modification d'une p-occurrence ti

- Insertion d'une nouvelle SDIA.

III.17.1 - DETERMINATION DE LA VALIDITE D'UNE D-CLASSE

Soit un p-type t ayant n fonctions attributs X1, ..., Xn. Une D-classe de t est un semi-domaine ayant n composantes explicites. Ainsi, pour déterminer la validité d'une D-classe, nous considérons les propriétés suivantes:

P1 - D est valide ssi tout D', tel que $D \subseteq D'$ est valide. (En conséquence du [lemme III-4].

P2 - Si D' est valide, tout D", tel que $D' \subseteq D''$ est valide (par définition de $\subseteq$).

P3 - S'il existe une relation de compatibilité entre les sous-domaines stables généralisés x et y alors le semi-domaine Dxy obtenu par unification de Dx et Dy est valide.

P4 - S'il existe une relation d'exclusion entre x et y, alors Dxy est non valide ou n'existe pas.

Ainsi, pour déterminer si la D-classe D est valide, il faut :
1 - déterminer la famille FD de semi-domaines Dx tel que :

a) $D \subseteq D_x$

b) Il existe un noeud en GVC qui représente Dx

c) Il n'existe aucun autre noeud dans le graphe qui représente Dy, tel que $Dy \subseteq Dx$ et $D \subseteq Dy$.

2 - Vérifier qu'il n'existe pas d'arcs d'exclusion entre les sous-domaines stables généralisés représentés par les semi-domaines obtenus en 1.

III.17.2 - Validité d'un p-occurrence

Pour déterminer si une p-occurrence t est valide, il faut :

1 - déterminer sa D-classe D

2 - déterminer si D est valide .

III.17.3 - Modification d'une p-occurrence

Soit ti une p-occurrence de t appartenant à la D-classe valide D. Après la modification de la valeur de deux ou de plusieurs fonctions attributs, la p-occurrence ti appartient à la D-classe D'. Deux cas sont possibles :

1 - D' = D dans ce cas, il n'y a rien d'autre à vérifier

2 - sinon il faut vérifier si D' est valide .

La détermination de la validité de D' exige la détermination de FD'. Il ne faut vérifier le type de relation entre les composantes de FD' que pour les éléments de F'D qui n'appartiennent pas à FD.

III.17.4-Insertion d'une SDIA

Les propriétés qui sont à la base de la détection de redondances et de D-inconsistances lors de l'introduction de nouvelles assertions sont intrinsèquement liées à la notion même de sous-domaine stable. Ainsi,

P1 - S'il existe une relation $\mathcal{E}$ entre δij et δkl, alors il existe une relation $\mathcal{E}$ entre $\delta' ij \subseteq \delta ij$ et δkl.

P2 - S'il existe une relation C5 entre δij et δkl, alors il existe une relation C4 entre δij et $\delta' kl$ $\underline{C} \delta kl$.

P3 - S'il existe une relation r, différente de C5 entre δij et δkl, alors il existe une relation r entre δij et $\delta' kl$ $\underline{C}$ δkl.

Lors de l'insertion d'une nouvelle assertion Ai : dij $\wedge$ dkl —> dfm v dfn, il faut :

1 - Déterminer les sous-domaines stables des fonctions nommées en Ai, c'est à dire, de Xi, Xk et Xf.

2 - En présence de la nouvelle décomposition en sous-domaines stables de Xi, Xk et Xf, déterminer les nouveaux sous-domaines stables généralisés.

3 - Rétablir les relations pour les nouveaux sous-domaines obtenus en 1 et en 2 (utilisation des propriétés P1 à P3).

4 - Créer, si nécessaire, un nouveau noeud pour représenter le sous-domaine stable généralisé associé à l'antécédent de Ai ; et représenter les relations intrinsèques à ce sous-domaine stable généralisé.

5 - Représenter les relations déduites de Ai, et exécuter les synthèses d'arcs si nécessaire.

6 - Fermer le graphe obtenu en 5 : pour la fermeture, on ne considère que noeuds qui sont père et/ou fils d'un nouvel arc [WAR62].

IV - DEPENDANCES INTER-OBJETS

IV.0 - INTRODUCTION

Les dépendances inter-attributs étudiées au chapitre précédent sont des règles qui déterminent la validité des occurrences d'un p-type t isolé : elles définissent la validité d'une p-occurrence indépendamment des autres p-occurrences de l'univers.

Or, en général, une p-occurrence établit des relations avec d'autres p-occurrences (cf. parag. I.4). Selon le modèle proposé, la spécification d'une relation entre p-occurrences exige la spécification d'un p-type t, dont $n \geq 1$ de ses fonctions attributs ATTR1, ATTR2, ..., ATTRn sont à résultat p-type t1, ..., tn : l'algèbre de t contient les algèbres de t1, ..., tn. En conséquence, la validité d'une p-occurrence x de t dépend de la validité des p-occurrences y1, ..., yn, telles que $ATTR^1(x) = y1$, ..., $ATTRn(x) = yn$. De même, la validité d'une p-occurrence yi du type ti, $1 \leq i \leq n$, dépend non seulement de sa validité individuelle (c'est-à-dire, du fait que yi satisfait les dépendances inter-attributs de ti), mais aussi de la satisfaisabilité de tous les axiomes de toute algèbre qui la contient. En particulier, la validité de yi dépend donc des axiomes de t. Dans ce chapitre, nous nous intéressons aux règles à partir desquelles on peut déterminer les axiomes de t qui font intervenir des p-occurrences de tout ti contenu en t.

Nous considérons les règles qui déterminent, pour toute relation R entre les éléments d'un ensemble A et les éléments d'un ensemble B, le nombre minimum et le nombre maximum d'éléments de B qui doivent être associés à un élément de A. Nous nommons dépendance inter-objets toute règle appartenant à cette catégorie. Dans un premier temps, nous étudions la sémantique qui leur est associée ; puis, nous définissons les propriétés qu'un état de la base de donnée doit avoir pour qu'il puisse être considéré valide, par rapport aux dépendances inter-objets ; finalement, nous proposons un mécanisme qui permet de garantir la validité d'un état de la base.

Le mécanisme proposé prend en compte le fait qu'une transition

d'états est déterminée par l'application d'une fonction primaire (insertion, modification ou suppression) ou par l'application d'une suite de fonctions primaires, ainsi que le fait que des dépendances inter-objets sont explicitement ou implicitement définies lors de la spécification d'un p-type t qui contient n p-types t1, ..., tn. Il consiste dans la détermination, à partir des dépendances inter-objets, des pré-conditions que les fonctions primaires des p-types t, t1, ...,tn doivent satisfaire pour qu'un état, obtenu par l'application d'une de ces fonctions, soit valide.

IV.1 - DEPENDANCES INTER-OBJETS

Soient A et B, deux ensembles de p-occurrences, éventuellement du même p-type ; soit R une relation binaire dont le domaine est A et le codomaine est B. Par définition de relation, à chaque élément ai de A est associé un sous-ensemble de B.

[déf IV.1]

Le sous-ensemble de B associé à ai ∈ A est l'ensemble composé par les n éléments de B bk, bj, ... tels que le prédicat "ai est en correspondance avec bk à travers R", noté (ai R bk), est vrai. Nous notons ai R B le sous-ensemble de B associé à ai.

En général, on ne sait pas déterminer a priori la cardinalité de ai R B. Par contre, on peut évaluer la limite inférieure (cmin) et la limite supérieure (cmax) entre lesquelles |ai R B| doit être comprise.

[déf IV.2]

Soit R une relation binaire entre l'ensemble A et l'ensemble B ; une dépendance inter-objets est une assertion qui limite inférieurement et supérieurement la cardinalité de tout ensemble ai R B, où ai est un élément quelconque de A.

[Proposition IV-1]

La dépendance inter-objets de la relation R est exprimée par une paire de valeurs (cmin, cmax), telles que, pour tout ai ∈ A, cmin ≤ |ai R B| ≤ cmax est un axiome.

L'utilisation de la paire (cmin, cmax) est classique dans le domaine des bases de données, surtout comme outil d'aide à la conception d'un schéma [ABR74] [MES81]. Par exemple, soit la relation ELEVES-DE dont le domaine est l'ensemble de professeurs et le codomaine est l'ensemble d'élèves, et dont la dépendance inter-objets est (1, *), "*" vaut "plusieurs". Habituellement, cette relation est représentée par :

$$\text{ELEVES-DE: prof} \xrightarrow{(1,*)} \text{élève}$$

Cette représentation suppose que dans la base de données, il n'existe qu'un ensemble dont les éléments sont de type "prof". De même, "élèves" est supposé être le type des éléments d'un seul ensemble.

A partir de la valeur (1, *) d'ELEVES-DE, on peut déduire un certain nombre de caractéristiques qu'une bonne structure doit avoir pour implanter de façon satisfaisante ELEVES-DE. Ainsi, si le SGBD concerné est de type réseau, une "bonne" structure serait un "set" ayant prof comme père et élève comme fils ; si le SGBD utilisé est de type relationnel, la valeur (1, *) permet de conclure que la clé de la "relation" implantant ELEVES-DE ne peut être prof ; cependant, selon la valeur (cmin, cmax) de la relation inverse d'ELEVES-DE, prof peut appartenir à la clé.

Dans ce chapitre, nous nous intéressons aussi aux informations déductibles d'une dépendance inter-objet. Cependant, une des lignes directrices de notre travail étant la spécification des propriétés d'un univers, nous utilisons ces règles non pas dans le but du choix d'une structure, mais plutôt dans le but de détermination de la validité d'une

p-occurrence dans un contexte où la spécification des relations entre p-occurrences a été faite.

IV.2 - ETATS D'UNE BASE DE DONNEES

Un certain nombre de travaux sur les bases de données et la logique ont mis en évidence que spécifier un schéma équivaut à spécifier les axiomes d'un univers[CAS77] [JAC82][GAI82]. Ils ont aussi montré qu'un état d'une base de données est une interprétation du schéma. Ainsi :

[déf IV.3.1]

Un état Ei d'une base de données est dit valide ssi il est une interprétation valide du schéma.

[déf IV.3.2]

Soit un schéma S composé par les p-types t1, ..., tn. Un état valide de S est composé par n algèbres t'1, ..., t'n, telles que t'i, $1 \leq i \leq n$ est une sous-algèbre de Wti.

Pour modifier l'état d'une base de données, l'utilisateur dispose de trois fonctions primaires de manipulation : l'insertion, la modification et la suppression d'une occurrence.

[Proposition IV-2]

Une transition Tij entre l'état Ei et l'état Ej est soit le résultat de l'application d'une fonction primaire opi, soit le résultat d'une séquence d'opérations primaires O : opi.opj. ...

[Proposition IV-3]

Une transition Tij, entre l'état valide Ei et l'état Ej, est valide ssi Ej est un état valide.

D'un point de vue algébrique, une transition d'états Tij entre l'état Ei composé par les algèbres t'1, ...,t'n, sous-algèbres de Wt1,...Wtn, et l'état Ej composée par les algèbres t"1, ..., t"n est valide ssi

i - toute t"i est une sous-algèbre de Wti

ii - soit t"i $\subseteq$ t'i soit t'i $\subseteq$ t"i

où Wti est l'algèbre des termes du p-type ti.

Soit Tij une transition entre l'état valide Ei et l'état Ej. Si Tij met en jeu des insertions ou des modifications de p-occurrences, une condition pour que Ej soit valide est que toute occurrence insérée ou modifiée appartienne à une D-classe valide (cf. chap. III). Cette condition est nécessaire mais elle n'est pas suffisante. En effet, l'appartenance d'une occurrence à une D-classe garantit sa validité individuelle et indépendante de l'existence d'autres occurrences. Cependant, elle ne garantit pas que l'état Ej soit valide. Par exemple, dans un contexte où la relation ELEVES-DE et sa dépendance inter-objet (1, *) ont été spécifiées, la validité de l'état Ej, obtenue par insertion d'un professeur pf, dépend de la validité de la D-classe de pf et de l'existence, dans l'état Ei, d'un élève ek, tel que le prédicat (pf ELEVES-DE ek) est vrai.

[Proposition IV-4]

Soit Tij la transition entre l'état valide Ei et l'état Ej ; soit Oij la séquence d'opérations primaires qui sont à l'origine de Tij, la transition Tij est valide ssi :

i - toute occurrence insérée ou modifiée appartient à une D-classe valide

ii - toutes les dépendances entre objets sont satisfaites.

Nous considérons que les seules règles spécifiées sont les dépendances inter attributs et les dépendances inter-objets. Dans un cas plus général, d'autres catégories de règles peuvent et doivent être introduites.

Au paragraphe suivant, nous proposons une sémantique pour les dépendances inter objets, puis nous déduisons les caractéristiques qui permettent de décider à priori de la validité d'une transition d'états, par rapport aux dépendances inter objets exprimées par l'utilisateur.

IV.3 - SEMANTIQUE D'UNE DEPENDANCE INTER-OBJET

[Proposition IV-5]

Soit R une relation définie entre les éléments des ensembles A et B ; spécifier la dépendance inter-objets (cmin, cmax) de R équivaut à spécifier que tout ai ∈ A doit être en correspondance

i - au minimum avec cmin occurrences de B

ii - au plus avec cmax occurrences de B

Cette proposition est une conséquence directe de la proposition IV-1.

Elle met en évidence les caractéristiques d'un état valide par rapport aux dépendances inter-objets. De plus, elle nous permet de déduire les axiomes qui conditionnent la validité d'une transition d'états. Ainsi,

[Proposition IV-6]

Soit R une relation entre A et B, et soit (cmin, cmax) sa dépendance entre objets ; soit Ei un état valide ; soit Tij : E —> Ej une transition d'états déterminée par l'insertion d'une occurrence ai ∈ A. Dans ce contexte, Tij est valide ssi dans l'état Ei, l'ensemble B contient c occurrences b1, b2, ... telles que ai est en correspondance avec ces c occurrences et cmin ≤ c ≤ cmax.

[Proposition IV-7]

Soit R une relation entre A et B ; soit Ei un état valide ; soit Tij : Ei —> Ej une transition d'états déterminée par la suppression d'un élément bk de B. Tij est valide ssi pour toute occurrence ai ∈ A, telle que (ai R bk), |ai R B| ≥ cmin + 1.

[Proposition IV-8]

Soit R une relation entre A et B ; soit Ei un état valide ; soit Tij : Ei —> Ej une transition d'états déterminée par la modification d'un élément bk ∈ B. Tij est valide ssi pour toute ai telle que, avant la modification de tk, elle est en correspondance avec bk, soit (ai R bk) après la modification, soit |ai R B| ≥ cmin + 1.

[Proposition IV-9]

La condition suffisante pour qu'une transition d'état Tij : Ei —> Ej, déterminée par une séquence Oij d'opérations primaires o1, o2, ... soit valide, est que les transitions déterminées par chacune des composantes de Oij soient elles-mêmes valides.

Dans ce cas, Tij peut être vue comme une séquence de transitions "primaires" valides.

En bases de données, on se restreint en général à deux cas de cmin (cmin = 0 et cmin = 1) et à deux cas de cmax (cmax = 1 et cmax = *). Ces quatre cas permettent de déterminer si la représentation fonctionnelle de la relation est respectivement une fonction partielle (cmin = 0) ou totale (cmin = 1) et si son type est un type simple (ou composé) (cmax = 1) ou le type ensemble (cmax = *). Dans ce chapitre, nous nous intéressons spécialement à ces quatre cas.

IV.4 - DEPENDANCES INTER-OBJETS ET MODELE PROPOSE

Selon le modèle proposé, une relation entre p-occurrences est implantée par un p-type ayant une, ou plusieurs, fonctions attributs à résultat p-type (cf. proposition II-3). Par exemple,

```
p-type personne

  attributs
     ...

     ENFS : personne —> ens[personne]
     ...

fin
```

ou encore

p-type enseignant

attributs

PROFESSEUR : enseignant —> prof /*prof est une vue de personne*/

ELEVE : enseignant —> élève /*élève est une vue de personne*/

...

fin

De plus, le modèle proposé permet la définition de plusieurs ensembles paramétrés par la même vue d'un p-type.

[Proposition IV-10]

Soit le p-type t ayant n fonctions attributs à résultat p-type t1,...,tn. Spécifier les dépendances inter-objets des relations t $\rightleftarrows$ ti, pour tout $1 \leq ti \leq n$, équivaut à spécifier des axiomes de l'algèbre hétérogène de t qui contient les algèbres de t1, ..., tn.

Pour chacune des fonctions à résultat p-type ti d'un p-type t, l'utilisateur peut spécifier deux dépendances inter-objets : l'une pour la relation ti $\rightrightarrows$ t et l'autre pour la relation inverse, c'est à dire t $\rightrightarrows$ ti. Ainsi,

[Proposition IV-11]

Soit le p-type t et soit ATTR une de ses fonctions attributs à résultat p-type ; soit ti, éventuellement égal à t, le type de ATTR ; soit R la relation t $\rightrightarrows$ ti :

i - si cmin de R est 0, la valeur indéfinie appartient au codomaine de ATTR (cf. chap. II)

ii - si cmax de R est plusieurs ("*"), la fonction attribut ATTR est une fonction de type ens[ti].

Lorsque la valeur indéfinie ne doit pas appartenir au codomaine de ATTR, cmin de R doit être 1. De même, lorsque le résultat de ATTR doit être un élément de type ti, la valeur de cmax de R est 1.

[Proposition IV-12]

Soit le p-type t et soit ATTR une de ses fonctions attributs à résultat p-type ; soit ti le type d'ATTR ; soit R' la relation ti $\rightleftharpoons$ t :

i - spécifier que cmin de R' est zéro équivaut à spécifier que les p-occurrences de type ti sont indépendantes de t

ii - spécifier que cmin = 1 équivaut à spécifier que pour tout t' de type ti il existe un t" de type t, tel que ATTR(t") = t'.

[Proposition IV-13]

Soit le p-type t et soit ATTR une de ses fonctions attributs à résultat p-type ti ; par défaut :

i - la dépendance inter-objet de la relation t $\rightleftharpoons$ ti a la valeur (1, 1)

ii - la dépendance inter-objet de la relation ti $\rightleftharpoons$ t a la valeur (0, *)

IV.5 - PRE-CONDITIONS D'UNE FONCTION PRIMAIRE

Au paragraphe IV.2, nous avons vu quelles caractéristiques un état Ei doit avoir pour être valide par rapport à une dépendance inter-objets. Dans ce paragraphe, nous déterminons les actions qui doivent obligatoirement accompagner l'exécution d'une fonction primaire pour que l'état final soit valide. Ces actions dépendent de l'opération et des dépendances exprimées. Ainsi, les seules fonctions primaires dont l'exécution peut mettre en cause la validité de la dépendance inter-objets de la relation R :ti $\rightleftharpoons$ tk sont l'insertion ou la modification d'un ti ou encore la suppression ou modification d'un tk. Le tableau ci-dessous résume les actions qui doivent accompagner chacune de ces quatre fonctions

pour que l'état final soit valide par rapport à R. Dans ce tableau, nous considérons que x est de type ti et y de type tk.

DIO	OPERATION PRIMAIRE QUI ENGENDRE Tij: Ei—>Ej	CONDITIONS QUE L'ETAT FINAL Ej DOIT SATISFAIRE	ACTIONS
$ti \xrightarrow{0,1} tk$ ou $ti \xrightarrow{0,*} tk$	1-INSERTION(x)	il peut n'exister aucun y en relation avec x	
	2-MODIFIER [ATTRl] (x,v)	il peut n'exister aucun y en relation avec x	
	3-MODIFIER [ATTRf] (y,w)	il peut n'exister aucun y en relation avec x	AUCUNE
	4-SUPPRIMER(y)	il peut n'exister aucun y en relation avec x	
$ti \xrightarrow{(1,1)} tk$	1-INSERTION (x)	il existe un y tel que (xRy):	
		a- si $tk \xrightarrow{*} ti$ ou $tk \xrightarrow{0,*} ti$, l'etat Ei peut avoir l'element y	a)Ei contient y: -ETABLIR la relation (xRy); b)Ei ne contient pas y: -CREER y; -ETABLIR la relation (xRy)
		b- si $tk \xrightarrow{1,1} ti$, y ne peut pas appartenir a Ei	-CREER y; -ETABLIR la relation (xRy);
	2-MODIFIER [ATTRl] (x,v)/* avant modification (xRy)	il existe un z tel que, apres la modification de x, (xRz):	
		a- y=z	aucune
		b- $y \neq z$ /*(xRy)*/	memes actions que pour l'insertion
	3-MODIFIER [ATTRj] (y,o) /*avant la modification de y il existe la relation (xRy)*/	il existe z tel que (xRz):	
		a-apres modification (xRy)/*z=y*/	aucune action

		b-apres modification (xRy)		
				a-soit les actions du cas INSERTION(x)
				b-soit supprimer x
	4-SUPPRIMER(y) tel que (xRy)	il existe z tel que (xRz)		
				a-soit les actions de l'insertion de x
				b- soit supprimer x
ti →* tk	1-INSERTION(x)	memes conditions que pour "INSERTION(x)" avec ti →* tk		
	2-MODIFIER [ATTRe] (x,w) /*avant modification de x |x R B| ≥ 1 */	∃v telle que (x R v):		
		a-si v ∈ (x R B)	→	aucune action
		b-sinon	→	memes actions que en "INSERTION (x)"
	3-MODIFIER [ATTRj] (y,o) /* la modification de y telle que il existe x1, x2, ..., xn, telles que (xi R y) pour 1 ≤ i ≤ n	pour tout xi telle que (xi R y) il faut:		
		a-soit |xi R y| >1	→	aucune action
		b-soit (xi R y)	→	aucune action
		c-soit ∃v telle que (x R y)	→	a-soit les memes actions que pour "INSERTION (x)"
				b-soit "SUPPRIMER (x)
	4-SUPPRIMER (y)	pour tout xi telle que (x R y) il faut:		
		a-soit |xi R B| >1	→	aucune action
		b-∃v telle que (x R v)		a-soit les memes actions que pour "INSERTION (x)
				b-soit celles de "SUPPRIMER (x)"

Pour garantir la validité d'une transition d'états conséquence de l'application d'une fonction primaire opj, nous exigeons, comme pré-condition de opj, que les conditions qui lui sont associées soient valides; une condition est satisfaite ssi une des actions qui lui sont associées, est exécutée. Par exemple,

p-type p-e

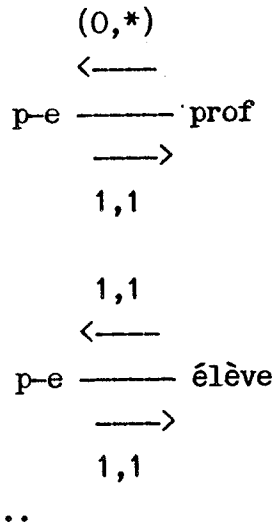
attributs

ENSEIGNANT : p-e —> prof

ELEVE : p-e —> élève

...

assertions



...

fin

Les pré-conditions associées aux opérations d'insertion sont (nous considérons que w est de type p-e ; x de type prof ; et y de type élève) :

1 - INSERER (w) ==> PRE-COND : ∄ x, telle que

ENSEIGNANT (w) = x ;

∄ y, telle que

ELEVE (w) = y

- La condition "∄ x telle que ENSEIGNANT (w) = x" est vraie si x appartient déjà à la base ou sinon x doit être créé par l'utilisateur (cf. tableau précédent).

- La condition "∄ y telle que ELEVE (w) =y" exige obligatoirement la création d'un nouvel élève (élève $\overset{1,1}{\underset{1,1}{\rightarrow}}$ p-e), car pour tout élève existant, il existe une relation w', et un élève rentre au plus dans une relation

2 - INSERER (y) ==> PRE-COND ∄ w telle que ELEVE (w) = y

- Cette condition est satisfaite ssi on insère l'élément w

3 - INSERER (x) ==> PRE-COND Rien

De la même façon, on peut déduire les pré-conditions des opérations de modification et de suppression.

Dans la plupart des cas, lorsque le système trouve "∄ x ... ", il doit demander à l'utilisateur la valeur de x ; selon la valeur de x, il doit choisir l'action à exécuter. Cette action est une fonction de la pré- existence de x dans la base et/ou de la possibilité de l'insérer. Au paragraphe suivant, nous proposons deux catégories d'assertions qui permettent de restreindre les opérations qui doivent être exécutées lorsque la clause "∄ x ... " est trouvée.

V.6 - ASSERTIONS SUR LES FONCTIONS ATTRIBUT A RESULTAT P-TYPE

Comme le nombre d'occurrences d'une base de données est fini, la

validité de la clause "∄ x ... " est toujours décidable : on doit parcourir tous les ensembles manipulés par l'utilisateur et dont les éléments sont du même type que x. S'il n'existe pas, on doit l'insérer dans tous ces mêmes ensembles.

Les assertions que nous proposons dans ce chapitre servent à restreindre les ensembles d'intérêt dans une recherche ou une insertion.

Par exemple, si l'on dispose de prof-grenoble : ens[prof] et de prof-agrégé : ens[prof], il est normal que l'utilisateur puisse indiquer dans lequel des deux ensembles il veut insérer x lorsqu'il insère w de type p-e (cf. paragraphe IV.5). Ainsi, nous proposons que l' utilisateur spécifie deux categories d' assertions:

Catégorie 1 - Spécification du codomaine des fonctions attribut à résultat p-type

Exemple

ENSEIGNANT (w) ∈ prof-agrégé

ou encore

ENSEIGNANT (w) ∈ prof-agrégé v ...

ENSEIGNANT (w) ∈ prof-grenoble

ou dans la terminologie du chapitre III

--> ENSEIGNANT (w) ∈ prof-agrégé v ...v

ENSEIGNANT (w) ∈ prof-grenoble

Avec cette catégorie d'assertion, le système peut déterminer les ensembles possibles où x doit être recherché et éventuellement inséré s'il rencontre la pré-condition "∄ x ... "

Catégorie 2 - Spécification des relations entre les valeurs des attributs d'un p-type ayant deux ou plusieurs fonctions attribut à résultat p-type. Ainsi, et pour le même exemple, l'utilisateur aurait pu spécifier

ELEVE(w) ∈ élève-grenoble —> ENSEIGNANT(w) ∈ prof-grenoble

Avec cette assertion, l'utilisateur dit que élève-grenoble appartient au codomaine d'ELEVE et que prof-grenoble appartient au codomaine de ENSEIGNANT ; il indique aussi quels sont les professeurs possibles des élèves de grenoble.

Dans cette catégorie d'assertion, on peut spécifier des conditions aussi précises que l'on souhaite. Par exemple, on aurait pu écrire :

VILLE-HAB (ELEVE(w)) = VILLE-TRAV (ENSEIGNANT(w))

Cette deuxième catégorie de règles est facultative ; cependant, la première catégorie est fondamentale. Nous considérons que si la spécification explicite du codomaine d'une fonction attribut à résultat p-type pi n'a pas été fait, tout ensemble paramétré par pi appartient au codomaine. La spécification explicite ou implicite des codomaines permet aussi de choisir les types ou insérer les pré-conditions déduites d'une dépendance inter-attributs. Ainsi, soit encore l'exemple du p-type p-e ; si lors de spécification de p-e l'utilisateur a spécifié:

ENSEIGNANT (w) ∈ prof-grenoble

ELEVES (w) ∈ élève-grenoble

On peut conclure que (cf. paragraphe IV.5) "INSERER (y) —> PRE-COND ... " doit appartenir à l'ensemble élève-grenoble. De même "INSERER (x) —> ..." est un axiome de l'ensemble prof-grenoble. La non-spécification de la première assertion a comme conséquence que "INSERER (y) —> ... " doit appartenir à tout ensemble paramétré par le type prof, et en particulier à prof-grenoble et à

prof-agrégé.

De même, en présence des deux assertions précédentes, l'axiome "INSERER (w) —> ... " doit appartenir à tout type ensemble dont le paramètre est de type p-e.

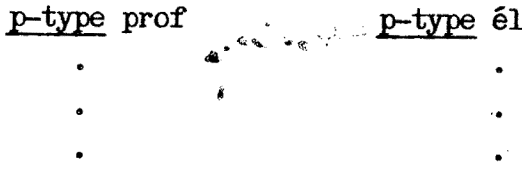
Comme dans le cas des dépendances inter-objet, la spécification de ces deux catégories d'assertion ne peut être faite qu'au niveau d'un p-type t qui "contient" d'autres p-types.

Les axiomes de t, significatifs pour les p-types ti qu'il contient, sont déduits à partir des dépendances inter-objet et de ces deux catégories d'assertions : les assertions restreignent la portée de " ∃... " de la pré-condition générale (cf. tableau du paragraphe IV.5)

Le besoin de traitement centralisé est nécessaire puisque un même p-type ti peut appartenir au support de plusieurs autres p-types tj,...,tf. Au paragraphe suivant nous presentons un cas où le p-type "prof" appartient aux p-types "p-e" et "etabl-prof".

IV.7 - DEPENDANCES INTER-OBJETS ET ALGEBRES HETEROGENES

Pour conclure ce chapitre, nous présentons un exemple où une même algèbre (prof) appartient à deux autres algèbres et en conséquence pour rétablir un état valide lors de l'exécution d'une fonction primaire sur professeur, il peut être nécessaire d'exécuter des opérations sur des types que l'on pourrait considérer a priori comme indépendants. Ainsi, supposons l'exemple



fin fin

p-type établissement

.
.
.

fin

p-type p-e

ENSEIGNANT : p-e —> prof

ELEVE : p-e —> élève

.
.
.

(1,1)
——>
p-e —— prof
<——
(0,*)

(1,1)
——>
p-e —— élève
<——
(1,1)

fin

p-type établ-prof

ETABL : établ-prof - établissement

PERSONNEL : établ-prof -> prof

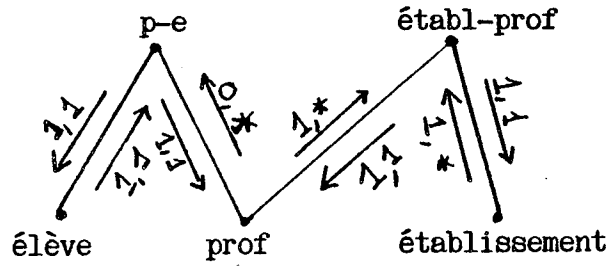
.
.
.

```
                1,1
               ---->
établ-prof ------ établissement
               <----
                1,*

                1,1
               ---->
établ-prof ------ prof
               <----
                1,*
```

.
.
.

fin

Les relations entre ces p-types peuvent être repésentées graphiquement par :



A partir de ce graphe, nous pouvons prévoir que l'insertion d'un élève engendre, dans un cas extrême, l'insertion d'une occurrence de chacun des autres p-types ((1,)) ; ainsi, il peut exiger l'insertion d'un p-e ; l'insertion d'un prof ; l'insertion d'un etabl-prof ; et finalement l'insertion d'un établissement.

De ce même graphe, on peut déduire que dans un cas extrême la suppression d'un établissement engendre la suppression d'occurrences de tous les p-types ; par contre, la suppression d'un élève y n'exige que la suppression de l'occurrence w de p-e, telle que ELEVE(w) est y. En effet, les professeurs peuvent "exister indépendamment" d'éléments de p-e. En conséquence, la suppression d'une occurrence w telle que ELEVE(w) = y n'exige pas la suppression du professeur x tel que ENSEIGNANT(w) = x.

J - ENVIRONNEMENT DE PROGRAMMATION

V.0 - INTRODUCTION

Le but de ce chapitre est de proposer des solutions techiques aux resultats formels du chapitre II. Nous commençons par présenter un environnement de programmation capable d'accepter la spécification d'un SCHEMA, puis nous montrons comment le concept de multifacettage peut être intégré dans la représentation du type abstrait d'un p-type. Finalement, nous proposons la spécification algébrique "externe" d'un schéma d'une base de données chargée de la gestion de l'"interface" entre les spécifications des utilisateurs et la représentation que nous proposons.

V.1 - ENVIRONNEMENT DE PROGRAMMATION

Dans ce paragraphe, nous spécifions les propriétés, les types abstraits et les fonctions minimaux qui sont nécessaires à la spécification d'un schéma. Nous rappelons que la spécification d'un schéma est faite en spécifiant des types, des p-types et des ensembles paramétrés par des p-types (cf. chap.II). En conséquence, nous nous intéressons aux propriétés, types et fonctions nécessaires à la spécification d'un p-type ou d'un ensemble paramétré par un p-type. Nous utilisons un langage proche de LPG [BER82] [BER83a] [BER83b].

Rappel

LPG est un Langage de Programmation Générique défini au sein de l'équipe Langage du Laboratoire IMAG. Son prototype est entré en service en 1983. Pour notre travail, les caractéristiques de LPG les plus intéressantes sont la possibilité de spécification algébrique de propriétés, de type et fonctions ; la possiblité de spécification modulaire, et, finalement, la possibilité de constituer des bibliothèques de propriétés, types et fonctions.

Ces bibliothèques, composées de spécifications pré-définies et pré-compilées constituent des environnements de programmation. Dans ce paragraphe, nous spécifions les propriétés, les types et les fonctions, qui doivent appartenir à une bibliothèque de bases de données : toute spécification d'un schéma exploite cette bibliothèque (c'est-à-dire environnement de la spécification d'un schéma) [BER83a]. C'est pourquoi ses propriétés, types et fonctions doivent permettre de caractériser un p-type. Or, (cf. chap. II) spécifier le p-type t équivaut à spécifier le type t, tel que :

i - un certain nombre de ses fonctions ont comme domaine t et codomaine ti (banalisé)

ATTR : t —> ti

Toute fonction de cette forme est appelée fonction attribut.

ii - un sous-ensemble des fonctions attributs de t sont des fonctions dont la valeur associée à une occurrence peut changer pendant la vie de l'occurrence : nous les avons nommées fonctions attributs modifiables.

iii - un sous-ensemble de fonctions attributs de t, disjoint de celui de ii), permet d'identifier de façon unique toute occurrence de t : la fonction, obtenue par construction de ces fonctions attributs, est appelée la CLE de t.

V.1.1 - Spécification des fonctions attributs

Le rôle qu'une fonction attribut peut être amené à jouer à l'intérieur même d'un type, nous oblige à distinguer les fonctions attributs des autres fonctions. En conséquence, nous spécifions la propriété "attribut" :

propriété attribut sur t, ti

opns

ATTR : t —> ti

fin

Par exemple, dans la spécification de "personne", on peut expliciter les fonctions attributs, en précisant quelles fonctionssatisfont la propriété "attribut".

type personne

opns

NOM : personne —> chaîne

ENFS : personne —> ens[personne]

. . .

eqns

satisfait attribut[personne, chaîne, opns

attribut[personne, ens[personne] opns ENFS]

fin

V.1.2 - Fonctions modifiables

Par définition même d'un type abstrait, on ne peut exiger qu'une fonction attribut soit une fonction modifiable, sauf s'il existe une fonction qui permet de modifier la valeur de la fonction attribut. Nous spécifions cette dernière fonction d'une façon générique.

a - Caractérisation d'une fonction modifiable

propriété attribut-modifiable sur t, ti

opns

MOD : t —> ti

satisfait attribut [t, ti opns MOD]

fin

Tout type t, qui à l'aide d'une fonction X satisfait "attribut-modifiable", satisfait aussi "attribut" puisque "attribut-modifiable" la satisfait.

b - Spécification de la fonction générique "MODIFIER"

enrich modifier-attr exige attribut-modifiable [t,tiopnsATTR]

opns

MODIFIER : (t, ti) —> t

vars x : t, y : ti

eqns

ATTR(MODIFIER (x,y)) == y

fin

Dans l'équation de "modifier-attribut" on fait appel implicite à l'occurrence de MODIFIER avec ATTR de la clause "exige..". En conséquence, la valeur de ATTR après l'application de la fonction MODIFIER est la valeur du deuxième paramètre de MODIFIER.

L'enrichissement "modifier-attr" exige la propriété attribut-modifiable. Dans cette condition, seuls les types abstraits qui satisfont "attribut-modifiable" peuvent être paramètres effectifs de la nouvelle fonction spécifiée, c'est-à-dire MODIFIER. Par exemple, la spécification ci-dessous :

type personne

opns

NOM : personne —> chaîne
ENFS : personne —> ens[personne]
SAL : personne —> entier
. . .

eqns

. . .

satisfait attribut [personne, chaîne opns NOM],
attribut [personne, ens[personne] opns ENFS],
attribut-modifiable [personne, entier opns SAL]

fin

garantit que le type personne satisfait "attribut-modifiable" pour l'attribut SAL et donc que SAL peut être paramètre effectif de "MODIFIER". Ainsi, une instanciation valide de cette fonction générique est :

MODIFIER [SAL] : (personne, entier) —> personne

Et, un appel de la fonction est :

MODIFIER [SAL] (pi, ej)

où pi et ej sont respectivement de type personne et entier.

Nous rappelons que du fait qu'"attribut-modifiable" satisfait "attribut", le type personne satisfait implicitement "attribut [personne, entier opns SAL].

Cependant, MODIFIER [NOM] : (personne, chaîne —> personne) est une instance non valide. En effet, la spécification de personne ne contient pas l'information que sa fonction NOM lui permet de satisfaire la propriété d'"attribut-modifiable".

V.1.3 - Spécification de la clé

Pour spécifier la CLE d'un p-type, nous devons expliciter sa composition, ainsi que le fait que pour chaque p-occurrence sa valeur soit unique. Nous faisons la spécification de CLE d'une façon modulaire. Ainsi :

a - Spécification de la propriété générale d'"égalité"

```
propriété égalité sur t

opns
  =: (t,t) —> bool
```

vars

x, y, z : t

eqns

x = x == vrai

x = y == y = x

x = y et non x = z == faux

fin

b - Explicitation que chacun des types, résultat d'une composante de la clé, satisfait la propriété d'"égalité" et est une fonction attribut:

propriété n-égalité sur t1,...tn

opns

EQ1 : (t1,t1) —> bool

. . .

EQn : (tn, tn) —> bool

hérite égalité [t1 opns EQ1],

. . .,

égalité [tn opns EQn]

fin

L'utilisation de "..." est abusive (une spécification algébrique contient un nombre fixe de paramètres). Cependant, nous les conservons par commodité, en sachant que pour chaque p-type le nombre n est bien déterminé.

propriété n-attributs sur t,t1,...,tn

exige n-égalité [t1...tn opns EQ1...EQn]

opns

C1 : t —> t1

. . .

Cn : t —> tn

hérite attribut [t,t1 opns C1]
. . .,
attribut [t,tn opns Cn]

fin

c - Spécification d'un enrichissement "Composition" qui spécifie la composition de la clé (G) et la fonction qui définit l'égalité pour le type de la clé (E) :

enrich composition exige n-attributs [t,t1,...,tn opns C1,...,Cn]
avec n-égalité [t1,...,tn opns EQ1,...,EQn]

opns
G : t —> (t1, ..., tn)
E : ((t1, ..., tn), (t1, ..., tn)) —> bool

vars
x : t
y, z : (t1, ..., tn)

eqns
G (x) == (C1(x),..., Cn (x))
E (y,z) == EQ1 (C1(y), C1(z)) et...et EQn (Cn(y), Cn(z))
satisfait égalité [(t1,...,tn) opns E]

fin

d - Finalement, on définit la propriété d'être "clé" et la propriété "égalité-par-clé". Celle-ci permet de décider si deux occurrences d'un p-type représentent ou non la même p-occurrence.

propriété clé sur t,ti exige égalité [ti opns EQ]

opns

CLE : t —> ti

fin

propriété égalité-par-clé sur t exige clé [t,ti opns CLE]
avec égalité [ti opns EQ]

opns

MEME : (t,t) —> bool

vars

x, y : t

eqns

MEME (x,y) == EQ(CLE(x), CLE(y))

fin

Exemple : soit le p-type salle dont la clé est composée par ses fonctions attributs ETAGE et NUM. En utilisant la fonction générique et les propriétés spécifiées dans ce paragraphe, nous spécifions le p-type salle par :

```
p-type salle
opns
    NUM    : salle —> entier

    ETAGE  : salle —> entier
           .  .  .

    MEME   : (salle, salle) —> bool

  satisfait clé [salle, (entier, entier) opns G [ETAGE, NUM]]
  avec égalité [(entier, entier) opns E [ETAGE, NUM]]
  hérite égalité-par-clé [salle opns MEME]
  .  .  .
fin
```

Ainsi, soit se et sj, deux variable de type salle, G(si) a pour résultat la clé de si et MEME (si, sj) est vrai si E (si, sj) est lui aussi vrai. Finalement, E(si, sj) est vrai si ETAGE(si) = ETAGE(sj) et NUM(si) = NUM(sj) ("=" est la fonction d'égalité entre entiers).

V.1.4 - Spécification d'un ensemble

Au deuxième chapitre, nous avons dit qu'un ensemble de p-occurrences est une instanciation du type abstrait générique "ens". Dans ce paragraphe, nous proposons une spécification du type ens de telle façon que chacune de ses instances soit un ensemble de p-occurrences.

```
type ens exige égalité-par-clé [t opns MEME]
             avec clé [t,ti opns CLE]
             avec égalité [ti opns EQ]
opns
     privé  EVIDE :  —> ens[t]
     EST-VIDE  : ens[t]  —> bool
     INSERER   : (ens[t],t) —> ens[t]
     SUPPRIMER : (ens[t],t) —> ens[t]
     REINSERER : (ens[t],t) —> ens[t]
     APP       : (ens[t],t) —> bool
     SELECTER  : (ens[t],ti) —> t
     privé APP-CLE :  (ens[t], ti) —> t
     ERREUR  :—>  exception

 vars       y:  ens[t]
     x, z:  t

 eqns

     EST-VIDE (EVIDE) ==vrai
     EST-VIDE (INSERER (y,z)) == faux
     INSERER (y,x) ==> ERREUR si APP(y,x)fsi
     INSERER (INSERER (y,z),x) == INSERER ((y,x),z)
     SUPPRIMER (y,x) ==> ERREUR si non APP (y,x) fsi
     SUPPRIMER (INSERER(y,x),z) == si EQ (CLE(x),CLE(z)) alors y
                                   sinon INSERER (SUPPRIMER (y,z),x) fsi
     APP (EVIDE,x) == faux
     APP (INSERER (y,z), x) == si EQ (CLE(z), CLE (x)) alors vrai
                               sinon APP (y,x) fsi
     REINSERER (y,x) ==> ERREUR si non APP (y,x) fsi
     REINSERER (INSERER (y,x),z) == si EQ (CLE(x), CLE(z))
                                    alors INSERER (y,z)
                                    sinon INSERER (REINSERER (y,z),x) fsi
     SELECTER (y,x) ==> ERREUR si non APP-CLE (y,x)
     SELECTER (INSERER (y,z),x) == si EQ (CLE(z), x) alors z
                                   sinon SELECTER (y,x) fsi
     APP-CLE (EVIDE,x) == faux
     APP-CLE (INSERER (y,z),x) == si EQ (CLE(z), x) alors vrai
                                  sinon APP-CLE (y,x) fsi

 fin
```

Le type proposé contient le nombre minimum de fonctions nécessaires à la manipulation d'un ensemble de p-occurrences. D'autres fonctions peuvent être spécifiées par enrichissement de ce type. Parmi ces fonctions, il y aura certainement la fonction qui permet de décider si deux ensembles sont égaux [BER79] et aussi celle qui permet de décider si un ensemble est vide. On pourrait aussi définir d'une manière générique des itérateurs sur les ensembles comme "pour-tout x tel que p(x) faire F(x)".

V.2 - Représentation logique d'un p-type

Dans ce paragraphe, nous proposons un modèle de mémoire qui garantit que toute p-occurrence satisfait le principe d'UNICITE [cf (I.4)]. Avec cette mémoire, le deuxième niveau de la spécification d'une donnée [CHE76] est réalisé. En conséquence, la mémoire permet d'associer une structure logique de données à une spécification de p-type (qui est le premier niveau de la spécification d'une donnée). Il ne faut pas oublier qu'une structure logique ne correspond pas encore à une structure de données directement implémentable. D'autre part, la spécification de la mémoire pré-suppose la validité de deux hypothèses couramment acceptées dans le domaine des bases de données.

[Hypothèse V.1]

Les occurrences d'un type de base (*) du langage (entier, bool, etc.) sont directement représentables sur l'ordinateur.

(*) Nous utilisons indistinctement les termes "type de base" et "type simple".

[Hypothèse V.2]

Une p-occurrence est directement représentable par sa CLE.

V.2.1 - Ensemble privé d'un p-type

Le modèle de mémoire est fondé sur l'existence d'un ensemble privé c'est-à-dire un ensemble non directement manipulable par l'utilisateur, associé à chaque p-type du schéma.

[Proposition V.1]

A tout p-type t est associé un ensemble privé, privé-t, composé, à chaque instant, par toutes les p-occurrences de type t qui ont une signification pour au moins un utilisateur de la base de données.

En sachant qu'une algèbre est associée à chacune des vues d'un p-type, ainsi qu'à son type nucléaire, on doit voir l'ensemble privé du p-type t comme le "support" commun aux k algèbres de t.

Nous utilisons la structure relationnelle pour représenter la structure logique des ensembles privés. Dans ce contexte, l'ensemble privé du p-type t, caractérisé par n fonctions attributs, est une relation sur n domaines : chaque domaine est, en effet, un co-domaine d'une fonction attribut. Cependant, comme le type d'une fonction attribut a été banalisé [cf.(II.1), la relation précédente n'est pas normalisée [BEE80] : les éléments d'un certain nombre de ses domaines ne sont pas directement représentables (cf. hypothèse V.1). Or, en ne considérant que les trois types de fonctions attributs les plus souvent rencontrées, les seules fonctions dont la représentation pose un problème sont les fonctions à résultat ensemble. En effet :

i - Si une fonction attribut est de type simple, les éléments de son codomaine sont directement représentables (cf. hypothèse V.1).

ii - Si le type d'une fonction attribut est un type composé, son codomaine est une partie du produit cartésien de m types. Les éléments de chacun

des m composantes sont soit de type simple, soit de type ensemble. Ce résultat est obtenu par la conjugaison de :

a - la prise en compte de seulement trois types de fonctions attributs ;

b - la "pré-existence" de n-1 des n types d'une présentation de type abstrait.

Exemple : soit la fonction attribut DIPLOME du p-type personne :

```
p-type personne

attributs

  DIPLOME :  personne —> diplôme

          . . .

fin


type diplôme

opns
   DATE    : diplôme —> date
   TITRE   : diplôme —> chaîne
   MENTION : diplôme —> chaîne
   ECOLE   : diplôme —> chaîne
   *CONST   : (date, chaîne, chaîne, chaîne) —> diplôme

fin

type date
opns
  MOIS  : date —> chaîne
   ANNEE : date —> entier
fin
```

le codomaine de DIPLOME est une relation 5-aire dont les domaines sont les codomaines des fonctions :

MOIS, ANNEE, TITRE, MENTION, ECOLE

Ainsi, la fonction DIPLOME est représentée par :

DATE		TITRE	MENTION	ECOLE
MOIS	ANNÉE			

Fig V.1 - fonction attribut de type composé

Cependant, lorsqu'une fonction attribut, ou une fonction composante d'une fonction attribut, est de type ensemble, le maintien de la souplesse de la représentation logique exige la spécification d'un nouveau p-type, dont chaque p-occurrence permet d'acceder à un élément de type du paramètre de la fonction que l'on représente. Par exemple, la représentation relationnelle de la fonction Xi dont le type est ens [xi] exige :

i - la spécification du p-type un-xi caractérisé par trois fonctions attributs, l'une desquelles (UN-Xi) est de type xi. Les autres deux fonctions sont d'un même type. Pour chaque p-occurrence, leurs valeurs sont respectivement la clé (CLE-UN-Xi) de la p-occurrence et la clé d'une autre p-occurrence.

p-type un-xi

attributs
CLE-UN-Xi :un-xi —> ξi
UN-Xi : un-xi —> xi
SUIVANT-UN-Xi : un-xi —> ξi
fin

Chaque p-occurrence pi de un-xi permet l'accès à un unique élément de type xi ; de plus, pi est liée, par l'intermédiaire de la valeur de SUIVANT-UN-Xi, à une autre p-occurrence de xi.

ii - Le "remplacement" du codomaine de Xi par le type de la fonction CLE-UN-Xi, c'est-à-dire ξi. Ainsi, soit t le p-type où Xi est définie et, tj une p-occurrence de t pour laquelle Xi(tj) n'est pas un ensemble vide ; à tj est associée la clé d'une unique p-occurrence pi de un-xi. Toutes les p-occurrences pj, pk... de un-xi, tels que UN-Xi(pj) ∈ Xi(tj), UN-Xi(pk) ∈ Xi(tj)..., sont attachées à pi. Parmi pj, pk... il y a une p-occurrence pe qui est attachée directement à pi (c'est-à-dire SUIVANT-UN-Xi(pi) = CLE-UN-Xi(pe)). Toutes les autres le sont de façon indirecte.

Dans la mesure où chaque p-occurrence de un-xi n,est liée qu'un unique élément de type xi, le nombre de p-occurrences de un-xi nécessaires à l'implémentation de l'ensemble Xi(ti) est égale à ¦ Xi(ti)¦.

Ci-dessous, nous proposons la représentation d'une fonction attribut Xi, dont une des fonctions composantes (Z) est de type ens[z].

p-type t
opns
Xi : t —> ti
. . .
fin

où ti est défini par :

type ti

opns

Y : ti —> chaîne
Z : ti —> ens[z]

fin

type z

opns

W : z —> chaîne
K : z —> entier

fin

la représentation relationnelle de Xi demande la spécification du p-type un-z :

p-type un-z

attributs

CLE-UN-Z : UN-Z —> ξ
UN-Z : UN-Z —> z
SUIVANT-UN-Z : UN-Z —> ξ

clé

CLE-UN-Z

fin

représenté par :

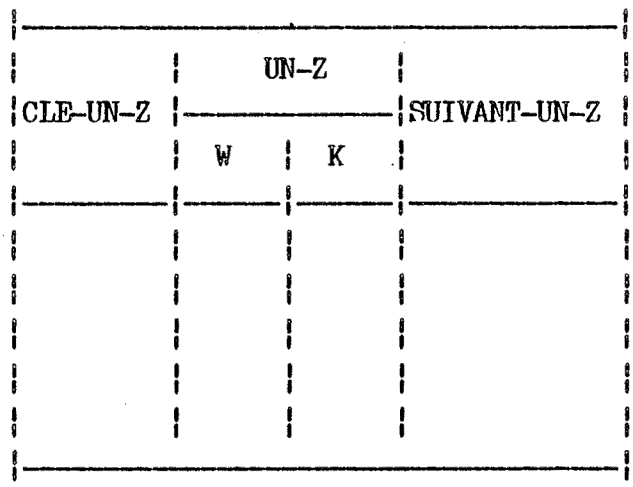


Fig V.2 - représentation relationnelle de privé-un-z

En conséquence, la représentation de Xi devient :

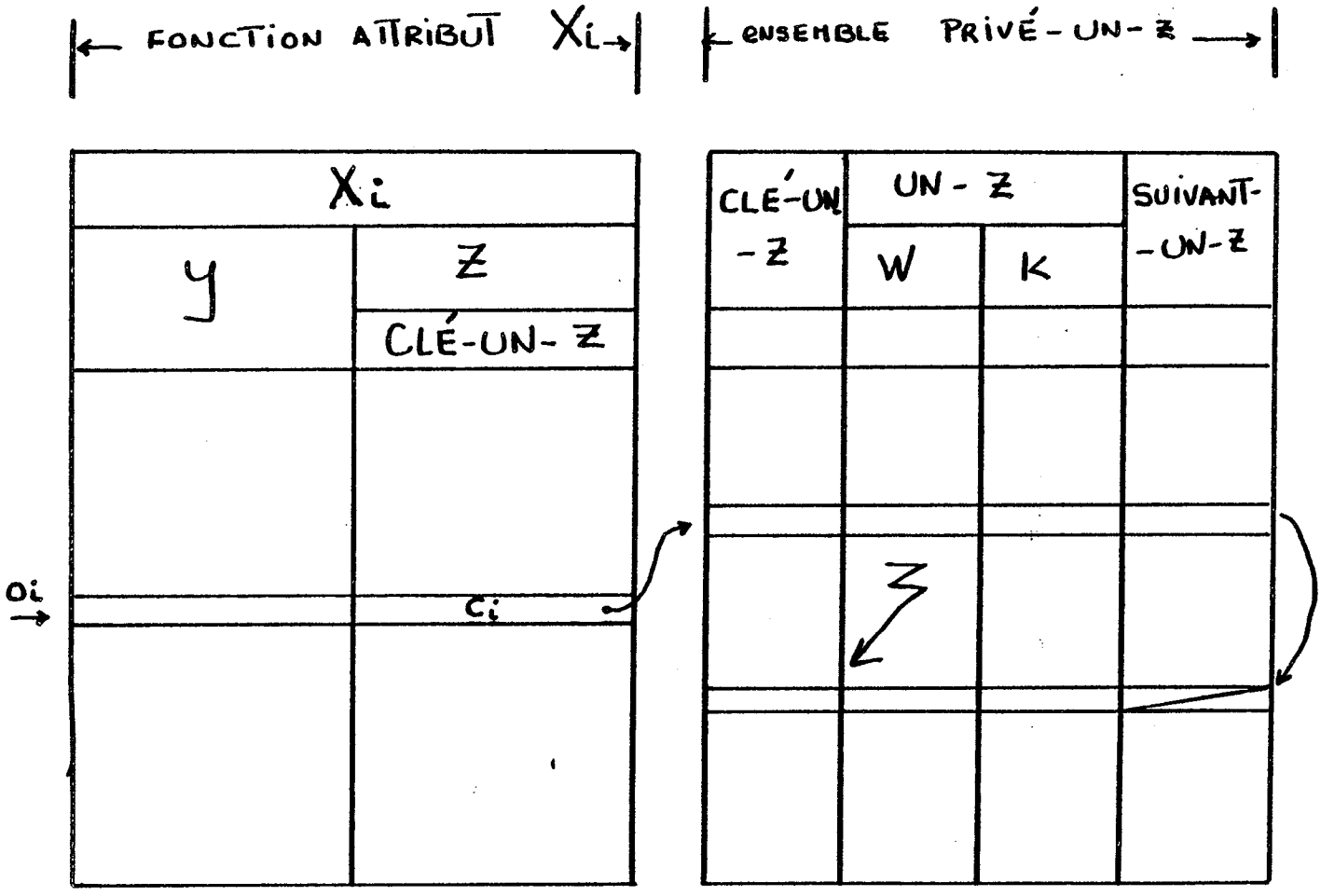


Fig V.3 - représentation d'une fonction de type ensemble

On a CLE-UN-Z(Oi) = ξi ssi il existe une occurrence pi-z telle que CLE-UN-Z(pi-z) = ξi et UN-Z(pi-z) ∈ Z(Oi) ; tous les autres éléments de Z(Oi) sont liés à l'élément pi-z.

La représentation logique proposée est une représentation très générale ; dans certains cas particuliers, d'autres solutions sont possibles [PAI79].

En résumé

L'ensemble privé du p-type t caractérisé par n fonctions attribut est représenté, sous une forme normalisée, par une relation de m ≥ n domaines; et fait appel à p ≥ o autres ensembles (p est le nombre de fonctions

de type ensemble du p-type t).

Si, à un instant donné, le nombre de p-occurrences du p-type t est K, le nombre de m-úples de la relation normalisée qui représente le p-type est lui aussi égal à K.

V.2.2 - Ensembles des utilisateurs

Etant donné l'hypothèse V.2, nous considérons qu'un ensemble de p-occurrences directement manipulables par des utilisateurs (c'est-à-dire crées par les utilisateurs) est logiquement représenté par un ensemble de clés (des p-occurrences). Une clé est composée par $n \geq 1$ fonctions attribut ; en conséquence, on représente un ensemble de p-occurrences, sous forme relationnelle normalisée, par une relation m-aire, où, $m \geq n$ est le nombre de codomaines de type simples obtenus directement, ou par des décompositions, des fonctions attributs qui appartiennent à la clé.

Par exemple, l'ensemble profs : ens[prof] est représenté par une relation 1-aire, puisque la clé de prof est NSS : personne —> chaîne.

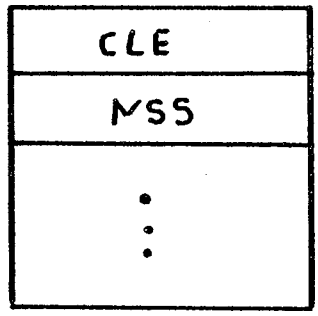


Fig. V.4 - Vision relationnelle de l'ensemble profs

Le nombre d'éléments d'un ensemble crée par des utilisateurs a :ens[t'] est inférieur ou égal au nombre d'éléments de l'ensemble privé qui est le "support" de t'.

V.2.3 - Fonctions attribut à résultat p-type

En utilisant à nouveau l'hypothèse V.2, nous considérons que la représentation d'une fonction attribut dont le résultat est un p-type t, est faite en représentant uniquement les fonctions attribut du type t qui appartiennent à sa CLE.

Ainsi, soit le p-type prof-élève spécifié par :

```
p-type prof-élève

opns
  ENSEIGNANT : prof-élève —> prof
  ETUDIANT   : prof-élève —> élève
  MATIERE    : prof-élève —> matière
      . . .

fin
```

La représentation des fonctions ENSEIGNANT et ETUDIANT est simplement :

ENSEIGNANT (p-type)	ETUDIANT (p-type)	MATIERE ...
CLE p-type	CLE p-type	...
NSS	NSS	...

Fig. V.5 - Représentation de fonctions attribut à résultat p-type

De même, l'attribut ENFS de personne :

p-type personne

attributs

ENFS : ens [personne]

.
.
.

fin

est représenté par :

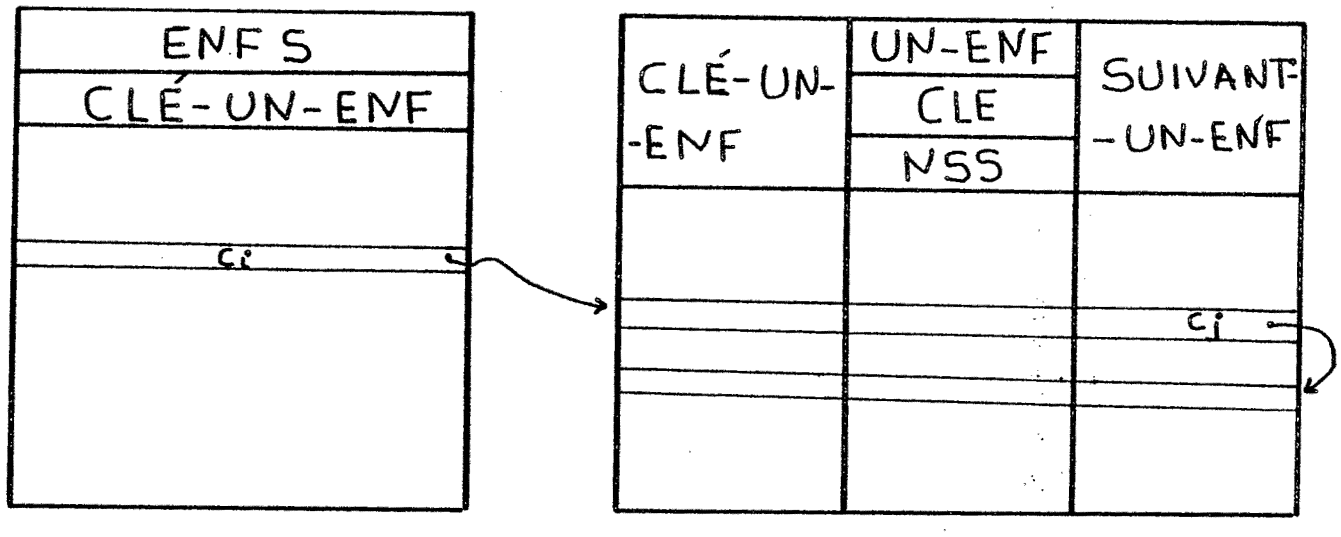


Fig. V.6 - Représentation de la fonction ENFS

V.2.4 - Modèle de mémoire

Notre modèle de mémoire est obtenu en mettant en commun les données des trois paragraphes précédents. Ainsi, soit un p-type t tel que :

- ses p-occurrences peuvent être vues de n façons différentes,

- il existe m ensembles créés par des utilisateurs paramétrés par des facettes de t,

- il existe q autres p-types ayant au moins une fonction attribut de type t.

La representation de t engendre un ensemble privé "privé-t" et m ensembles utilisateurs. De plus, sa clé est représentée dans q autres ensembles privés ; et, finalement, pour tout p-type p, appartenant aux q p-types préalables, si la fonction attribut de type t rentre dans la composition de la clé de p, alors la clé de t appartiendra aussi aux ensembles utilisateurs paramétrés par p. Une représentation de ce que l'on vient de décrire est, par exemple :

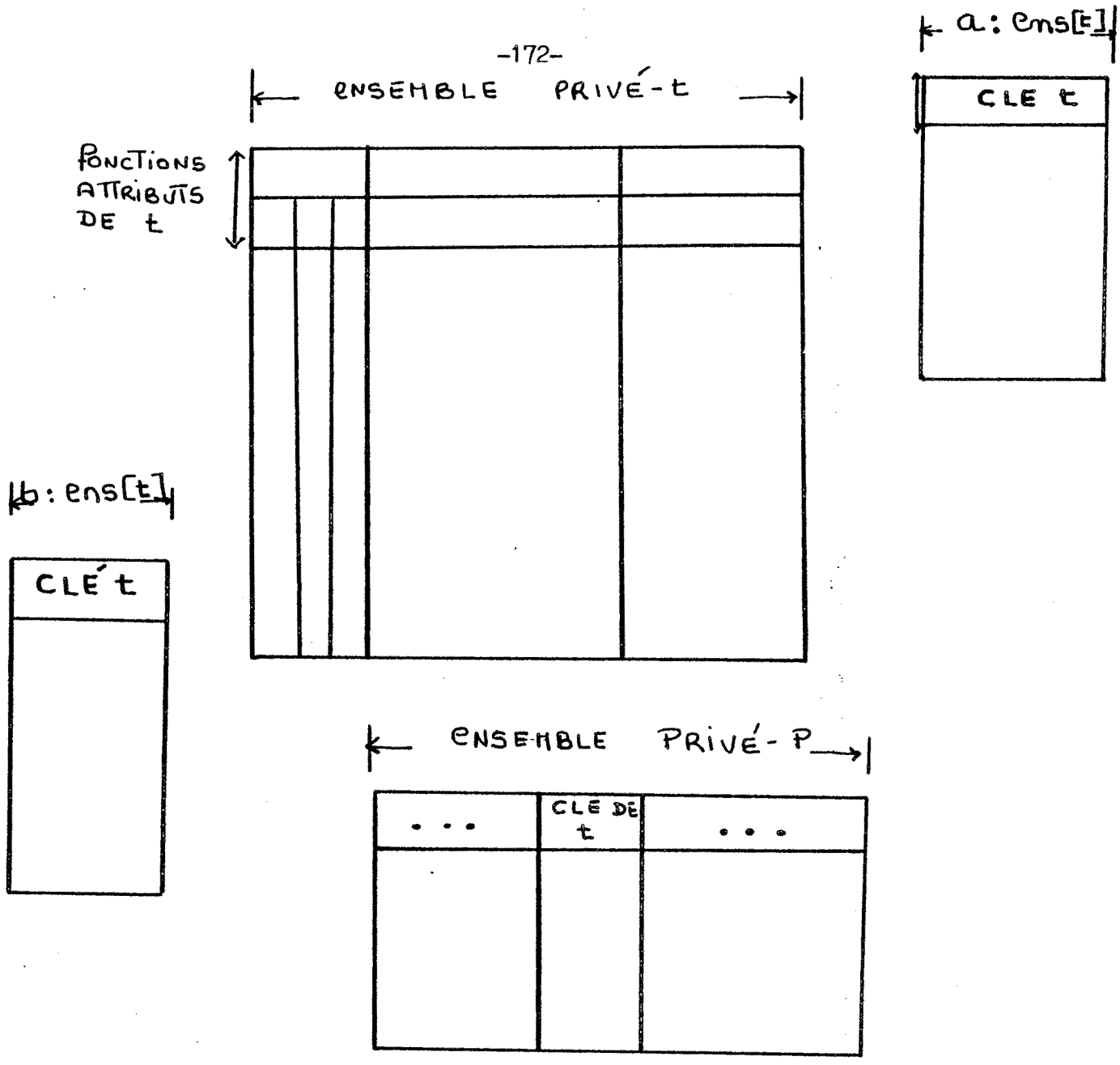


Fig. V.7 - Modèle de mémoire d'un p-type t

L'ensemble "privé-t" centralise toutes les informations sur toutes les p-occurrences de t. Partout ailleurs, il existe seulement les clés des p-occurrences. La centralisation de l'information associée à une p-occurrence, garantit, à tout instant, qu'une information unique soit associée à une p-occurrence : le principe d'UNICITE est, en conséquence, lui aussi garanti.

Nous considérons que la manipulation de la mémoire est soumise aux

principes suivants :

i - toute p-occurrence privé-ti du p-type privé-t, "support" du p-type t doit être significative pour au moins un utilisateur, c'est-à-dire
privé-ti appartient à privé-t, ssi il existe une p-occurrence ti de t et un ensemble créé par des utilisateurs, b :ens[t'] (t' est une vue de t), tels que ti ∈ b et CLE(ti) = CLE(privé-ti).

ii - la modification de la valeur de la fonction attribut ATTR pour la p-occurrence ti ∈ b, où b :ens[t'], est valide ssi après modification ti continue à appartenir à b.

iii - soit ti une p-occurrence de t qui satisfait les vues t', t"... ; après une modification valide de ti on doit le supprimer de tout ensemble c :ens[t"] tel que ti ne satisfait plus t".

Les conséquences de la centralisation de l'information, et des trois principes précédents, sur les opérations de manipulation des ensembles créés par des utilisateurs, sont synthétisés dans le tableau ci-dessous. Nous considérons que t' (t", etc.) est soit le type nucléaire, soit une vue du p-type t :

OPERATION	ACTION SUR L'ENSEMBLE UTILISATEUR a:ens[t']	ACTION SUR L'ENSEMBLE PRIVE prive-t
SELECTION de la p-occurrence ti dans l'ensemble a	1-ERREUR si CLE(ti) $\notin$ a 2-si CLE(ti) $\in$ a ⟶	selection en prive-t des informations de ti <—> seules les informations significatives a t', qui est le parametre de "a", sont selectionnees
INSERTION de la p-occurrence ti dans l'ensemble a	1-ERREUR si ti n'est pas de type t' 2-si ti est de type t' ⟶	ti est deja present en prive-t: i- sous forme de t' et avec les memes valeurs fournies par l'utilisateur
	INSERTION en a de CLE(ti). ⟵	
		ii- sous forme de t' mais avec au moins une valeur differente pour une fonction attribut
	ERREUR <—> ti ne satisfait pas l'UNICITE ⟵	
		iii-sous forme de t'', tel que t'' $\xrightarrow{m}$ t', et pour les fonctions attributs communes a t' et a t'' il y a concordance entre les valeurs stockees en prive-t et fournies par l'utilisateur: a-REECRITURE en prive-t les valeurs de ti qui ne sont pas encore stockees b- ⟵
	INSERTION de CLE(ti) en a ⟵	

		iiii-sous forme de t´´ tel que t´´ t´, mais il y a au moins une fonction de t´´ pour laquel le il n´y a pas de concordance entre la valeur stockee en prive-t et la valeur fournie par l´utilisateur
	ERREUR ←	
	3- si CLE(ti) a →	et il y a concordan- entre les valeurs stockees en prive-t et celles fournies par l´utilisateur
	RIEN ←	
MODIFICATION de la valeur de la fonction attribut Xi de la p-occurrence ti "appartenant" a l´ensemble a	1-ERREUR si CLE(ti) n´ appartient pas a a 2-ERREUR si apres modification ti n´est plus de type t´ 3-si apres modification ti est de type t´ →	MODIFIER la valeur associee a Xi pour la p-occurrence ti et
	SUPPRIMER CLE(ti) de tout ensemble b tel que b:ens[t´´´] et t´—>t´´´ ←	
SUPPRIMER ti de l´ ensemble a	1-ERREUR si CLE(ti) $\notin$ a 2-si CLE(ti) $\in$ a il faut la supprimer de a et i-si ti n´appartenait qu´ a l´ensemble a →	SUPPRIMER ti de l´ensemble prive-t
	ii-si ti n´appartient plus a aucun ensemble b:ens[t´´´], $t´\xrightarrow{m}t´´´$ →	EFFACER les valeurs associees aux fonctions attributs propres a t´

V.3 - GESTION DES NOMS SIGNIFICATIFS D'UN SCHEMA

En général, une ou plusieurs représentations, ou instances de représentations génériques, sont associées de façon exclusive à un type abstrait. Au deuxième chapitre, nous avons conclu que spécifier un p-type t, ayant n vues significatives, équivaut à spécifier n + 1 algèbres. Parmi ces algèbres, il existe celle associée au type nucléaire de t. Elle constitue l'algèbre minimale de t, c'est-à-dire l'algèbre qui inclut toutes les autres n algèbres.

Or, le modèle de mémoire proposé exige qu'un même ensemble soit le "support" des n + 1 algèbres. De plus, en général, chaque algèbre n'est concernée que par une partie de la représentation globale. Ainsi, puisque la vue t' du p-type t est spécifiée, uniquement par une partie des fonctions et des règles de t, la représentation de t (privé-t) qui est significative pour t' est délimité d'une part par les fonctions attributs de t', et d'autre part par les p-occurrences de t qui sont modèles de t'. Pour résoudre le problème du partage de l'ensemble support par les n + 1 algèbres, nous proposons qu'une base de données soit chargée de gérer les noms significatifs du SCHEMA spécifié par l' utilisateur. La gestion des noms significatifs permet en outre d'établir une interface entre les noms significatifs d'une vue et la représentation globale de son p-type.

De plus, pour vérifier aisément et efficacement les trois principes auxquels la gestion de la mémoire est soumise, nous proposons que l'ensemble des fonctions attributs de tout type nucléaire du p-type t soit automatique "étendu" par deux fonctions attribut APP-VUE et APP-ENS. Pour chaque p-occurrence ti de t, ces fonctions délivrent, respectivement, les vues selon lesquelles ti peut être perçue et les ensembles auxquels elle appartient.

V.3.1 - Bases de données de noms significatifs

Le SCHEMA de la base de données chargée de la gestion des noms significatifs d'un schéma d'utilisateurs, est composée par :

A - un p-type dont chaque p-occurrence est associée a une fonction attribut :

```
p-type nom-attribut

attributs

    NOM-ATTRIBUT :  nom-attribut --> chaîne
    REPRESENTATION :  nom-attribut --> rep

clé

    NOM-ATTRIBUT

modifiable

    REPRESENTATION

fin
```

"rep" est le type qui modélise la représentation des fonctions attributs sur les ensembles support :

```
type rep

opns

    NOM-ENS-SUPPORT :  rep --> chaîne/*nom de l'ensemble privé */
    COLONES :  rep --> ens[τi]
    *CONST :  (chaîne, ens[τi]) --> rep

fin
```

et, finalement le type τi est :

type τ i

opns DEBUT : τi —> entier
LONGUEUR : τi —> entier
*CONST : (entier, entier) —> τi

fin

B - un p-type dont chaque p-occurrence est associée à un nom de vue ou de type nucléaire :

p-type nom-de-vue

attributs

NOM-VUE : nom-de-vue —> chaîne /*nom d'une vue ou d'un type nucléaire*/
P-TYPE : nom-de-vue —> chaîne/*nom du p-type*/
ATTRIBUTS : nom-de-vue —> ens[nom-attribut]
CLE-DE : nom-de-vue —> ens[nom-attribut]
SONT-INCLUS : nom-de-vue —> ens[nom-de-vue]
EST-VUE : nom-de-vue —> bool
EST-T-NUCLEAIRE : nom-de-vue —> bool

clé

NOM-VUE

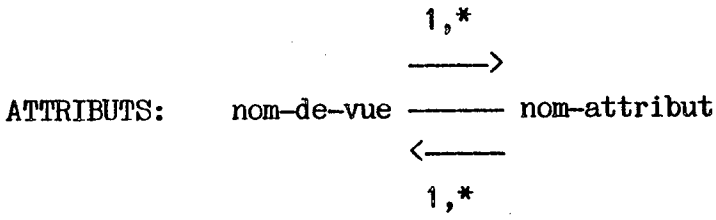
modifiables

ATTRIBUTS

assertions

EST-VUE(x) == NON EST-T-NUCLEAIRE(x)
NOM-VUE(x) = P-TYPE(x) == EST-T-NUCLEAIRE(x)
CLE(x) ∈ ATTRIBUTS(x)

× ∈ SONT-INCLUS(x)



fin

Les p-occurrences de nom-de-vue sont (automatiquement) engendrées par la spécification d'un type nucléaire ou d'une vue. Par contre, l'extension d'une vue t' a comme unique conséquence la modification de la valeur de ATTRIBUTS(t'). Ainsi, si l'extension de t' est accomplie à l'aide de n > 0 fonctions attributs, on sait que :

$$|\text{ATTRIBUTS}(t')|_{\text{après}} = |\text{ATTRIBUTS}(t')|_{\text{avant}} + n$$

C'est la fonction ATTRIBUTS de nom-de-vue qui, pour chaque vue t' du p-type t restreint les fonctions de t à celles propres à t'.

C - un p-type dont chaque p-occurrence est associée à un ensemble créé par des utilisateurs :

p-type nom-ensemble

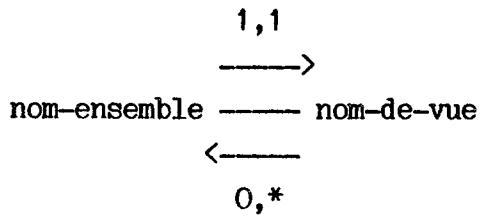
attributs

NOM-ENSEMBLE : nom-ensemble —> chaîne
T-PARAMETRE : nom-ensemble —> nom-de-vue

clé

NOM-ENSEMBLE

assertions



fin

Le schéma de la base de données chargée de la gestion des noms significatifs est donc composé de trois uniques p-types. La représentation relationnelle de ce SCHEMA est :

privé-nom-attribut :

	REPRESENTATION	
NOM-ATTRIBUT	NOM-ENS-SUPPORT	COLONES
		CLE- i

← PRIVÉ - NOM - DE - VUE →

NOM-VUE	P-TYPE	ATTRIBUTS	CLE-DE	SONT-INCLUS	EST-VUE	EST-T-NUCLEAIRE
		CLE-UN-NOM-ATTRIBUT	CLÉ-NOM-ATTRIBUT	CLÉ-NOM-VUE		

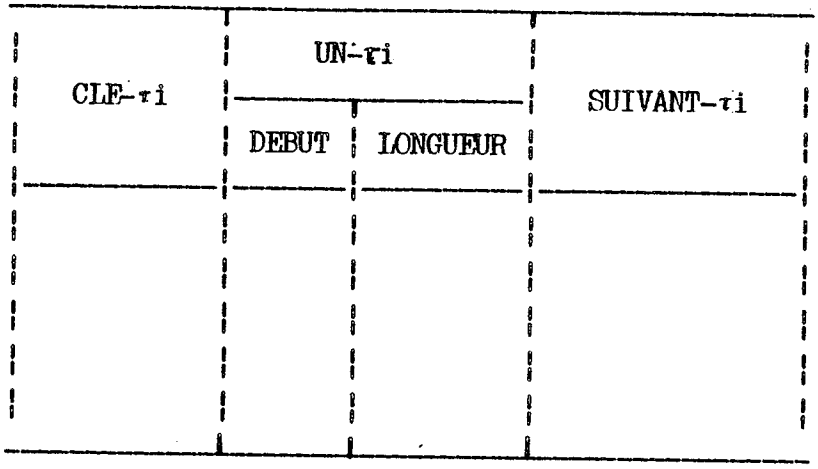
← PRIVÉ - UN - NOM - ATTRIBUT →

CLE-UN-NOM-ATTRIBUT	UN-NOM-ATTRIBUT	SUIVANT-UN-NOM-ATTRIBUT
	CLE	
	NOM-ATTRIBUT	

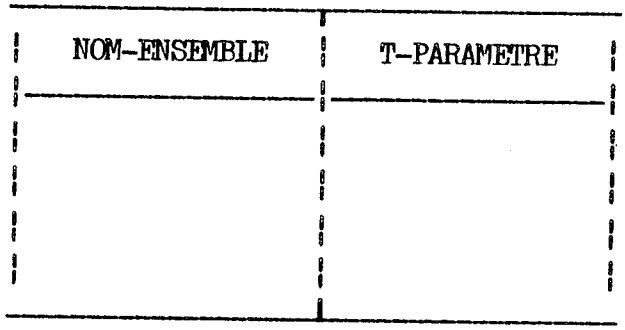
← PRIVÉ - UN - NOM - VUE →

CLE-UN-NOM-VUE	UN-NOM-VUE	SUIVANT-UN-NOM-VUE
	CLE	
	NOM-VUE	

privé-UN-τi :



privé-nom-ensemble :



Lors d'une spécification d'un SCHEMA des p-occurrences de nom-attribut, de nom-de-vue et de nom-ensemble sont engendrées ; elles constituent les valeurs de la base de données de noms significatifs. Ainsi :

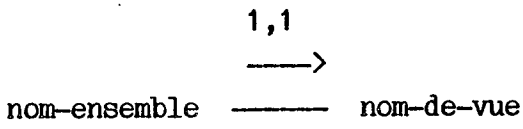
- La spécification d'une vue, ou d'un type nucléaire, engendre une p-occurrence de nom-de-vue et n p-occurrences de nom-attribut, n étant le nombre de fonctions attributs de la vue.

- La spécification d'un nouveau ensemble engendre la création d'une p-occurrence du type nom-ensemble.

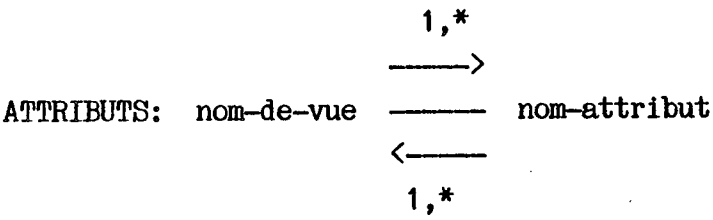
- L'extension d'une vue engendre la modification d'une p-occurrence de nom-de-vue (ATTRIBUTS) et la création de n p-occurrences de nom-attribut, où n est le nombre des nouvelles fonctions spécifiées dans l'extension.

De plus, les "dépendances entre objets" que nous avons spécifiées exigent que :

i - le type paramètre d'un ensemble créé par des utilisateurs soit connu au moment de la spécification de l'ensemble :



ii - une vue ou un type nucléaire ait, au minimum, une fonction attribut ; et une fonction attribut appartient obligatoirement à, au moins, une vue :



V.3.2 - Fonctions supplémentaires des types nucléaires

Les deux fonctions que nous proposons d'ajouter automatiquement à tout type nucléaire t sont :

p-type t

attributs

APP-ENS : t —> ens[nom-ensemble]
APP-VUE : t —> ens[nom-de-vue]

assertions

x ∈ APP-ENS(z) —> ∃y : y ∈ APP-VUE(z) et T-PARAMETRE(x) = y

fin

Le résultat de l'application d'APP-VUE et d'APP-ENS à une p-occurrence ti du p-type t est, respectivement, l'ensemble de vues selon lesquelles ti peut être perçue, et, l'ensemble des ensembles auxquels elle appartient. Notre premier principe sur la manipulation de la mémoire [cf. V.2.4] exige à toute p-occurrence ti de t, appartenant à privé-t, que APP-ENS(ti) et APP-VUE(ti) ne soient pas des ensembles vides.

La suppression de ti d'un ensemble b : ens[t'] exige la suppression de l'élément b de APP-ENS(ti) et, s'il n'y a aucun autre élément de APP-ENS(ti) tels que son paramètre est de type t', alors la suppression de ti de b est accompagnée par la suppression de t' de APP-VUE(ti).

Dans la mesure où ti peut appartenir à plusieurs ensembles paramétrés par la vue t' de t, la vérification que ti est un modèle de t', c'est-à-dire que satisfait les assertions de t', est faite une seule fois; en effet, si ti est déjà vue selon t' (t' C APP-VUE(ti)), l'insertion de ti en c : ens[t'] n'a pas besoin d'autres vérifications.

VI - CONCLUSION

VI - CONCLUSION

TRAVAIL EFFECTUE

La définition d'un langage de très haut niveau pour la spécification des bases de données est un thème vaste et complexe. En effet, une base de données est, en général, au service de plusieurs catégories d'utilisateurs ayant chacune sa propre vision de l'univers. De plus, les besoins des utilisateurs, et l'univers lui-même, peuvent évoluer. Par conséquent, un langage de spécification orienté vers les bases de données, doit contenir des concepts assez puissants pour permettre de prendre en considération la vision individuelle de chaque catégorie d'utilisateurs, la vision multiple de l'univers qui en résulte, ainsi que l'évolution de l'univers et des besoins des utilisateurs. Mais il faut également pouvoir garantir la cohérence de la spécification globale de l'univers, ainsi que la validité de tout état de la base.

Ainsi, face à la prolifération des langages, il nous a semblé plus important de caractériser les notions, les concepts et les techniques de base d'un langage de spécification de bases de données que de faire une définition complète d'un nouveau langage.

Dans cette perspective, notre démarche a été d'abord de cerner et de synthétiser différentes manières de définir un univers. Nous sommes arrivés à la conclusion que, indépendamment de leur type, tous les SGBDs permettent de définir, sous une forme plus ou moins commode, des objets (appelés "record", "relation", "entity", etc) par leurs "attributs", et des associations entre objets (appelées "set", "relation", "associationship", etc) ; ils offrent également la possibilité de définir plusieurs visions de l'univers (appelées "sous-schéma", "schéma externe", etc) pour les différentes catégories d'utilisateurs.

Le choix des concepts de base du langage nous a demandé de bien comprendre l'essence même des notions d'objet et d'association entre

objets et de bien distinguer la partie spécification de la partie représentation dans une définition de données. Nos conclusions ont été :

1 - Il n'y a pas de différence formelle entre le concept d'objet et celui d'association entre objets ;
2 - Un objet est défini par un certain nombre de propriétés (attributs et règles) ;
3 - Un objet peut entrer dans la constitution d'autres objets ;
4 - Un objet peut être vu selon plusieurs facettes.

En conséquence, le concept de base d'un langage pour les bases de données est celui d'objet satisfaisant le principe d'unicité (cf. I.4), noté p-type dans ce travail : quelle que soit l'occurrence d'un p-type, sa valeur est unique et indépendante du nombre d'ensembles auxquels cette occurrence appartient, du nombre de facettes selon lesquelles on peut l'apercevoir, et du nombre de relations qu'elle établit.

Le choix d'une structure mathématique qui puisse être un support valide à la spécification des p-types a constitué la deuxième phase du travail. Ainsi, nous nous sommes intéressés aux solutions formelles adoptées dans les langages du spécification, et, en particulier, aux types abstraits algébriques. L'étude de la théorie algébrique qui leur est sous-jacente, ainsi que des techniques développées pour l'adapter aux besoins de modularité, de généricité et de maximisation des vérifications statiques, nous a permis de conclure que la structure algébrique est a priori une structure valide. En effet :

1 - Un type abstrait est défini par ses fonctions. L'importance de cette caratéristique peut être mesurée en sachant que le type d'une fonction est quelconque, et que sa valeur dépend exclusivement de ses paramètres. Dans ces conditions, l'unicité, nécessaire aux objets des bases de données, est formellement obtenue si une définition fonctionnelle du concept d'"attribut" est possible. De plus, la généralisation du type d'un "attribut" devient immédiate, ce qui permet à l'utilisateur de spécifier sa vision d'un objet.

2 - La spécification d'un type abstrait t exige la donnée de sa sémantique. Or, la sémantique d'un type est le reflet des lois qui

permettent de caractériser les éléments valides et distincts de Wt. Cette centralisation des lois rend la validité des occurrences de t indépendante de ses utilisateurs éventuels, ce qui est fondamental dans les bases de données.

3 - On peut enrichir un environnement et, en particulier, un type déterminé. Cette caractéristique est importante par la modularité qu'elle permet d'introduire dans la spécification d'un objet.

4 - L'indépendance entre la spécification d'un type et sa représentation. Ceci est déjà reconnu comme important pour implanter le concept d'indépendance entre la spécification et la représentation des données ("data-independance").

Notre deuxième conclusion a été que, pour profiter au maximum de cette structure, il faut la restreindre à nos besoins concrets, sans quoi son utilisation risque d'être très peu efficace et donc non viable.

La troisième phase du travail a donc consisté à redéfinir le concept de p-type en termes algébriques, puis à proposer des techniques qui rendent l'utilisation du p-type facile, sûre et efficace. Ainsi, nous avons proposé la redéfinition fonctionnelle du concept d'attribut et la définition algébrique des concepts de type nucléaire et de vues d'un p-type. En conséquence, nous montrons que le concept d'association entre objets est une extension de celui d'objet par l'introduction des fonctions à résultat p-type. Finalement, nous proposons que la définition d'un schéma soit vue comme la spécification des types abstraits. Plus concrètement, ces types peuvent être des p-types et des ensembles paramétrés par des p-types.

Les techniques développées sur les p-types appartiennent essentiellement à deux grandes catégories. La première catégorie a pour but de permettre la spécification des lois qui expriment sa sémantique. Nous avons considéré que la donnée d'un ensemble S de lois n'est valide, dans le contexte des bases de données, que si l'on peut mettre en évidence des éventuelles incohérences, et si l'on peut minimiser les vérifications

dynamiques lors des modifications de la base. Nous nous sommes limités aux lois qui expriment des dépendances inter-objets.

La deuxième catégorie de techniques fournit un support de représentation aux différentes algèbres associées à un p-type. Cette représentation garantit l'unicité de l'information associée à toute occurrence du p-type. Finalement, pour gérer les noms significatifs externes (connus par l'utilisateur) et internes (propres à la représentation), nous proposons le schéma d'une base de données ad-hoc à toute base de données particulière.

PERSPECTIVES FUTURES

Le travail que nous avons commencé n'est pas encore arrivé à son terme. L'expérience acquise au long du projet, nous amène à envisager trois phases pour sa continuation :

1 - Formalisation des concepts qui dépendent directement des résultats déjà obtenus. Parmi eux, nous soulignons :

- La formalisation de la suppression d'une vue. A notre avis, les mécanismes de la suppression d'une vue sont a priori assez semblables à certains des mécanismes qui sont en jeu lorsqu'une vue v est enrichie. En effet, après enrichissement la vue v "est remplacée" par une vue v1, ce qui peut poser des problèmes au niveau des vues v', telles que, avant l'enrichissement, $Wv1 \subseteq Wv'$.

- La formalisation du concept de "vue dérivée". L'introduction de ce concept demande l'adaptation, au problème concret des bases de données, de l'opérateur "DERIVE" [BUR79b] [GOG77]. Il permettrait de spécifier des vues dont la valeur de certaines fonctions attributs sont déduites à partir des valeurs d'autres fonctions attributs du p-type. Les "vues dérivées" sont un bon support au besoin de schémas externes qui ne sont pas des sous-ensembles directs du schéma conceptuel [TSI78] [DEL78].

2 - Le but du travail à réaliser dans cette phase est d'augmenter les possibilités de spécification du langage. Ainsi, nous considérons que :

- Il faut permettre à l'utilisateur de spécifier d'autres catégories de règles. Cela signifie en particulier qu'il est nécessaire de cerner ces autres catégories de règles et de proposer les mécanismes appropriés d'aide à la spécification et de maximisation des vérifications statiques. Dans cette partie du travail, il est intéressant d'étudier comment adapter des travaux d'autres chercheurs et en particulier ceux de [NIC82] [NIC83].

- On doit permettre de spécifier les catégories d'utilisateurs par leurs droits d'accès (lecture, écriture, modification, transmission, spécification, etc...). Il est nécessaire que, en présence des spécifications des vues et de la spécification des droits d'une catégorie d'utilisateurs, on puisse décider si la catégorie est bien définie. Par exemple, en présence de la règle prof $\xrightarrow{*,*}$ p-e, on ne peut pas donner à une catégorie d'utilisateurs le droit d'insérer un professeur sans lui donner celui d'insérer un élément de p-e.

- Il nous semble intéressant que le langage de spécification offre la possibilité de définir des transactions au niveau du p-type. A priori, l'introduction des transactions ne pose pas de problèmes théoriques, sauf si l'on considère le problème essentiel de maximisation des vérifications statiques. En effet, on peut toujours considérer une transaction comme une suite d'opérations primaires et donc, on peut penser qu'à la fin de la transaction , le système doit effectuer la série de vérifications nécessaires à chacune de ses opérations.

  Cependant, il serait intéressant de minimiser ces vérifications dynamiques, ce qui exige l'étude des vérifications à effectuer lors de l'exécution d'une séquence d'opérations primaires.

3 - Dans cette phase, nous considérons l'établissement de liens entre ce qui a été réalisé jusqu'à la phase précédente et d'autres domaines directement concernés. Ainsi, on doit envisager une étude systématique du langage de manipulation . puis une étude de l'adaptation du langage de spécification aux bases de données réparties : les dépendances

inter-objets peuvent être vues comme une première approche à la spécification du "site" dans une base de données répartie.

En paraphrasant [GOG76], nous concluons en disant que ce travail a été très intéressant pour le domaine des bases de données et pour la spécification algébrique. En effet, l'exemple concret que sont les bases de données nous a permis d'introduire le concept d'unicité au niveau de la spécification algébriques. Parallèlement, la structure algébrique a été un bon support théorique pour les bases de données : elle nous a permis de mettre en évidence des concepts fondamentaux d'un langage de spécification des bases de données.

BIBLIOGRAPHIE

[ABR74] ABRIAL J.R. "data semantics"

IFIP conference carsege april 1974

[ANS77] "ansi/x3/sparc dbms frame work study group on data base management systems"

A.C.M. SIGMOD bulletin 7, no.2, 1975

[AST76] ASTRAHAN M.M. and ALL "system R: relational approach to database management"

A.C.M. trans. on database systems, vol.1, no.2, june 76

[BAB82] BABERYE,G., JOUBERT,T., MARTIN,M. "OASIS:un outil d'aide à la spécification interactive et structurée"

actes des journées BIGRE-GRENOBLE 1982

[BAC78] BACKUS J. can programming be liberated from de von newmann style ? A functional style and its algebra of programs"

C.A.C.M., vol.21, no.8, 1978

[BEE80] BEERI C., BERNSTEIN P., GOODMAN N. "a sophisticate introduction to database normalisation theorie"

Proc. VLDB 80, IEEE 80

[BEN81] BENNEWORTH R., and all "the implementation of GERM, an entity-relationship database management systems"

Proc. VLDB 81, IEEE 81

[BRN80] BERNSTEIN PH., BLAUSTEIN B., CLARK E. "fast maintenance of integrity assertions using redundant aggregate data" Proc VLDB 80, IEEE 80

[BER77] BERT D., JACQUET P. "generic abstract data types"

5th annual conf., wc 2.1, IFIP, mai 1979

[BER79] BERT D. la programmation generique-construction de logiciel, spécification algébrique et vérification"

thèse DOCTEUR ES-SCIENCE, juin 1979, usmg GRENOBLE

[BER81] BERT D. "LPG langage pour la programmation générique"

r.r. no. 262, sept 81

[BER81b] BERT D.,SOLER R. "about data type genericity"

lectures notes in computer science 1981

[BER82] BERT D. "generic programming: a tool for designing universal operators. Application to program algebra"

r.r. no. 336, nov 1982

[BER82b] BERT D. "software components constructions"

in "notion for programming constructions", camb. univ. press., page 347-376, 1982

[BER83] BERT D. "refinements of generic specification with algebric tools"

in "information processing 83, ed. mason, IFIP 83

[BER83b] BERT D. "manuel de reference de LPG, version 1.2"

r.r. no. 408, dec 83

[BRO78] BRODIE M. "a specification and verification of database semantics integrity"

Ph. D. univ. TORONTO, 1978

[BRO78b] BRODIE M., SCHMIDT J. "what is the use of abstract data type in data bases?"

Proc VLDB 1978,IEEE 1978

[BRO80] BRODIE M. data abstractions, databases and conceptual modelling"

Proc VLDB 1980, IEEE 1980

[BRO80b] BRODIE M. "standardization and relational approach to database: an ansi task group status report"

IEEE 1980

[BUR77] BURSTALL R.M., GOGUEN J.A. "putting theories together to make specification"

Proc. of 5th inter. join conf. on art. int., cambridge 1977

[BUR79] BURSTALL R.M., GOGUEN J. "an informal introduction to specifications using CLEAR"

Proc of 1979 compegnagen winter school on abst. soft. specifications, feb 1980

[BUR79b] BURSTALL R., GOGUEN J. "the semantics of clear, a specification language"

Proc. of 1979 compenhagen winter school on abst. soft. specification, feb 1980

[BUR80] BURSTALL R. "electronic category theorie"

m.f.c.s., 1980, poland

[BUR81] BURSTALL R., GOGUEN J. "algebras and theories: an introduction for computer scientists"

expos. pas., july 1981

[CAD76] CADIOU J.M. "on semantic issues in the relational model of data"

IBM res.lab., 1976

[CAS77] CASANOVA M., BERNSTEIN P. "the logic of a relational data manipulation language"

1977

[CAS80] CASANOVA M., BERNSTEIN P. "a formal system for reasoning about programs accessing a relational database"

A.C.M. trans. on prog. language, vol.2, no.3, july 1980

[CAT83] CATELL R.G.G. "design and implementation of a relationship-entity-datum data model"

C.S.L.-83-4, may 83

[CHA81] CHAMBERLIN D., GILBERT A., YOST R. "a history of system R and SQL/data system"

Proc. VLDB, IEEE 1981

[CHAN73] CHANG C.L., LEE R.T.C. "symbolic logic and mechanical theorem proving"

new york, academic press 1973

[CHE76] CHEN P. "the entity-relationship model: toward a unified view of data"

A.C.M. trans. on database system, vol.1, no.1, march 1976

[CLE78] CLEMONS E.C. "the external schema and codasyl"

Proc VLDB 1978, IEEE 1978

[CODA78] CODASYL DDL COMMITTEE "report of the codasyl data description language in information systems"

vol.3, no.4, 1978, perg. press

[COD70] CODD E.F. "a relational model of data for shared data banks"

C.A.C.M., vol.13, no.6, 1970

[COD71] CODD E.F. "a database sublanguage founded on the relational calculus"

Proc 1971 sigfidet workshop acm, new york 1971

[COD72] CODD E.F. "a relational completeness of database sublanguage"

courant computer sciences symposia, vol.6: data base systems, prentice-hall, englewood cliffs, 1972

[COD79] CODD E.F. "extending the data relational model to capture more meaning"

A.C.M. trans. on database systems, vol.4, no.4, dec 1979

[DAT81] DATE C. "an introduction to database systems"

3th ed. add.-wes.-publ., 1981

[DAT81b] DATE C. "referential integrity"

Proc VLDB 1981, IEEE 1981

[DEL77] DELOBEL C. "normalisation and hierarchical dependencies in the relational data model"

A.C.M. trans. on database systems, vol.3, no.3, sept77

[DEL78] DELOBEL C. "les bases de données"

publ. IMAG 1978

[ENG76] ENGMANN R. "set-theoritic concepts in data structures and data bases"

T.H.T., memorandum no.111, 1976

[FAG77] FAGIN R. "multivalued dapendencies and a new normal form"

A.C.M. trans. on database systems, vol.2, no.3, sept. 1977

[FAG81] FAGIN R. "normal forms and relational database based on domains and keys"

A.C.M. trans. on database systems, vol.6, no.4, dec 1981

[KOW79] KOWVALSKI R. "logic for problem solving"

the computer science library; int. art. series, 1979

[GAI82] GALLAIRE H., MINKER J., NICOLAS J.M. "logic and database: an overview and survey"

joint report CERT/CGE/UNIV. MARYLAND, 1982

[GOG76] GOGUEN J.A., THATCHER J.W., WAGNER E.G., WRIGHT J.B. "a junction between computer science and category theory, I: basics concepts and exemples (part I)"

R.C. 4526, sept 1976, IBM research report

[GOG77] GOGUEN J., THATCHER J., WAGNER E. "initial algebra semantics and continuous algebras"

J.A.C.M., vol.24, no.1, jan 1977

[GOG80] GOGUEN J., THATCHER J., WAGNER E. "an initial algebra approach to the specification, correctness and implementationof abstract data types"

in support du cours "les types de données, concepts et applications", IRIA, mars1980

[GOI80] GOLDMAN N., WILE D. "a database foundation for process specifications"

I.S.I./r.r.-80-84, octobre 1980

[GUT77] GUTTAG J. "abstractdata types and the developmentof data structures"

C.A.C.M., vol.20, no.6, june 1977

[GUT78] GUTTAG J., HORNING J. "the algebric specification of abstract data types"

acta informatica, no.10, 1978

[JAC82] JACOBS B. "on database logics"

J.A.C.M., vol.29, no.2, april 1982

[JCQ78] JACQUET P. "les types abstraits génériques, propositions pour un mecanisme d'abstraction dans les langagesde programmation"

thèse de 3ième cycle, GRENOBLE 1978

[JOR77] JORRAND PH. "very high level languages: some aspectsof the evolutionof language design"

int. comp. symp. 1977

[LEA80] LEAVENWORTH B. "an integrated database programming language"

r.c.8462 (36846), computer science, 1980

[LIS74] LISKOV B., ZILLES B. "programming with abstract data types"

Proc. of a symp. on very high level language, SIGPLAN notices, vol.9, no.4, april 1974

[MES81] MESGUICH A., NORMIER B. "comprendre les bases de données, théorie et pratique"

manuels inf. masson, 1981

[MOI82] MOITRA A. "direct implementationof algebric specification of abstract data types"

Trans. on soft. eng., vol.SE-8, no.1, 1982

[MUS80] MUSSER D. "abstract data type specificationin the affirm system"

Trans. on soft. eng., vol.SE-6, no.1, jan. 1980

[MYL80] MYLOPOULOS J., WONG H. "some features of TAXIS data model"

Proc. of VLDB 1980, IEEE 1980

[MYL80b] MYLOPOULOS J., BERNSTEIN P., WONG H. "a language facility for designing database-intensive applications"

A.C.M. trans. on database systems, vol.5, no.2, june 1980

[NIC79] NICOLAS J.M. "contribution a l'étude théorique des bases de donneées—apports de la logique mathématique"

THESE docteur es-science, toulouse 1979

[NIC82] NICOLAS J.M. logic for proving integrity checking in relational data bases"

acta informatique, 1982

[NIC83] NICOLAS J.M., YAZDANIAN K. "un apperçu de BDGEN: un sgbd deductif"

Proc. IFIP 1983

[PAI79] PAIR C., GAUDEL M.C. "les structures de données et leur representation en mémoire"

INRIA, 2ième ed., 1979

[RIS77] RISSANEN J. "independent components of relations"

A.C.M. trans. on database systems, vol.2, no.4, 1977

[SAI84] SALES A., SIMONET M. "analyse et verifications statiques d'une famille de contraints d'intégrité"

r.r. (à paraitre)

[SAV81] SAGIV Y, DELOBEL C, FAGIN R. "an equivalence between relational databases dependencies and a fragment of propositional logic"

J.A.C.M., vol.28, no.3, july 1981

[STA77] STANAT D., McALLISTER D. "discret mathematics in computer science"

prentice-hall inc., 1977

[SCH77] SCHMIDT J "some high level language constructs for data of type relation"

A.C.M. trans. on database systems, vol.2, no.3, sept 1977

[TAY76] TAYLOR R., FRANK R. "codasyl data-base menagement systems"

computing surveys, vol.8, no.1, march 1976

[TSI76] TSICHRITZIS D., LOCHOVSKY F. "hierarchical data base management systems"

A.C.M. comp. surv. vol.8, no. 1, march 1976

[TSI78] TSICHRITZIS D., KLUG A "the ansi/x3/sparc dbms framework: report of study group on database management systems"

inf. systems no.3, 1978

[WAR62] WARSHALL S. "a theoremon boolean matrices"

J.A.C.M., vol.9, 11-12, jan. 1962

[WII80] WILSON G.A. "a conceptual model for semantic integrity checking"

Proc VLDB 1980, IEEE 1980

[ZAN79] ZANIOLO C. "multimodel external schemas for codasyl data base management systems"

IFIP TC-2 working conf. on database

DERNIERE PAGE D'UNE THESE

3È CYCLE, DOCTEUR INGÉNIEUR OU UNIVERSITÉ

Vu les dispositions de l'arrêté du 16 avril 1974,

Vu les rapports de MM. Philippe JORRAND et Didier BERT....

M. .....

Madame Ana Maria SALES..... est autorisée

à présenter une thèse en vue de l'obtention du grade de DOCTEUR Ingénieur.....

.....

Grenoble, le - 9 AVR. 1984

Le Président de l'Université Scientifique
et Médicale

M. TANCHE