Institut National Polytechnique de Lorraine

Centre de Recherche en Informatique de Nancy

SINDBAD

UN SYSTÈME EXPÉRIMENTAL D'AIDE À LA SPÉCIFICATION ET À L'UTILISATION DE BASES DE DONNÉES DÉDUCTIVES

(Concepts et techniques basés sur les types abstraits de données)

THESE

soutenue publiquement le 9 Mai 1984

À L'INSTITUT NATIONAL POLYTECHNIQUE DE LORRAINE

pour l'obtention du grade de
DOCTEUR INGÉNIEUR EN INFORMATIQUE
par

Nacer BOUDJLIDA

devant la Commission d'Examen

J.C. DERNIAME
VICQ
JJ. CHABRIER
D. COULON
S. SPACCAPIETRA
J. DUCLOY

Institut National Polytechnique de Lorraine

Centre de Recherche en Informatique de Nancy

SINDBAD

UN SYSTÈME EXPÉRIMENTAL D'AIDE À LA SPÉCIFICATION ET À L'UTILISATION DE BASES DE DONNÉES DÉDUCTIVES

(Concepts et techniques basés sur les types abstraits de données)

THESE

soutenue publiquement le 9 Mai 1984

À L'INSTITUT NATIONAL POLYTECHNIQUE DE LORRAINE

pour l'obtention du grade de
DOCTEUR INGÉNIEUR EN INFORMATIQUE
par

Nacer BOUDJLIDA

devant la Commission d'Examen

Président:	M.	J.C. DERNIAME
Examinateurs:	Mme	VICQ
	MM.	JJ. CHABRIER
		D. COULON
		S. SPACCAPIETRA
		J. DUCLOY

A mes parents,

Je tiens à remercier expressément,

M. le professeur J.C. DERNIAME, Directeur du centre de recherche en informatique de NANCY, d'avoir accepté de présider ce jury et aussi d'avoir été un lecteur très critique d'une version intermédiaire de cette thèse.

Mme VICQ, chargée de mission auprès de l'Agence pour le Développement de l'Informatique d'avoir montré beaucoup d'intérêt à ce travail et d'avoir accepté de prendre part à ce jury.

M. S. SPACCAPIETRA, professeur à l'Université de DIJON, d'avoir dans un premier temps prêté une oreille attentive à ce travail puis de siéger dans ce jury.

M. D. COULON, professeur responsable de la formation à l'Institut Polytechnique de LORRAINE d'avoir accepté de faire partie de ce jury.

M. J. DUCLOY (ANL-CRIN) de s'être intéressé à ce travail et d'avoir accepté de le juger.

M. JJ. CHABRIER, professeur à l'Université de DIJON et membre du CRIN. C'est sous sa très avisée direction que ce travail a été mené. Qu'il trouve ici l'expression de toute ma gratitude pour toutes ses qualités techniques et humaines dont il m'a fait bénéficier.

Mme M. CREHANGE et M. J.P. FINANCE d'avoir tous deux contribué à ma venue et à mon insertion au CRIN.

Je remercierai également Monsieur le Directeur du Centre d'Etudes et de Recherches en informatique d'ALGER de m'avoir dégagé de mes fonctions le temps nécessaire pour accomplir cette thèse.

./...

Que Mme J. CHABRIER, Mlles A. BENCHERIFA, M. N. TERRASSE et MM. K. PROCH, J. DURAND, T. GRISON acceptent également mes remerciements pour l'ambiance amicale et joyeuse mais néanmoins studieuse qu'ils ont su faire régner au sein de l'équipe.

Mme M.T. DRIQUERT a réalisé la frappe de ce document avec diligence, gentillesse et... un ineffaçable sourire. Mme D. MARCHAND m'a apporté son aide pour la réalisation de son tirage.

Qu'elles acceptent toutes deux tous mes remerciements.

SOMMAIRE

CHAPITRE II : LES SYSTEMES EXPERTS

CHAPITRE III : LES BASES DE DONNEES DEDUCTIVES

- PARTIE B : ABSTRACTION ET SYSTEMES DEDUCTIFS -

CHAPITRE I : TYPES ABSTRAITS DE DONNEES : CONCEPTS ET MECANISMES

INTRODUCTION

Avoir des machines qui "raisonnent", les rendre disponibles à des utilisateurs non spécialistes ... sont des préoccupations actuelles d'un grand nombre de chercheurs. Notre travail s'inscrit dans cette ligne de recherche. Il a trait, de façon plus précise, aux bases de données dites déductives ([GALL 78], [NICO 82 a]). Dans ce domaine, nous nous sommes particulièrement intéressés à étudier (et à expérimenter) des moyens qui élargiraient les possibilités d'interrogation offertes par les systèmes de gestion de bases de données (SGBD) conventionnels, tout en réduisant le degré de technicité exigé de leurs utilisateurs.

Par ce biais, nous voulons parvenir à la réalisation de systèmes dont le rôle ne se limite plus à "détenir la mémoire" d'un domaine (la base de données du domaine), y ranger des informations, les conserver et les restituer à la demande, mais qu'il soit élargi par des aptitudes à :

- utiliser des informations de la mémoire pour en extraire de nouvelles qui n'y figurent pas explicitement. (Dès à présent, nous appellerons de tels systèmes : systèmes de données déductifs).
- et interpréter des demandes incomplètes d'information, réduisant ainsi le degré de spécialisation de leurs utilisateurs et étendant leurs possibilités "d'utilisation naïve".

Ces systèmes doivent donc assumer les fonctions classiques (mémorisation, restitution, contrôle, préservation de la qualité des données ...) et en offrir de nouvelles (déduction et compréhension). Ils s'apparentent, de ce fait, à des systèmes experts ([LAUR 82], [STEF 82]...).

L'examen des expériences menées dans le domaine ([CHAN 76], [KELL 76], [NICO 82 b]...) et dans celui des systèmes experts ([LESS 77], [CLAM 81], [BART 81], [STEF 82], [LAUR 82],...) montre que les points déterminants pour la puissance et l'efficacité de

ces systèmes sont ceux qui concernent :

- l'appréhension et la représentation du "savoir" que doit posséder le système (c'est-à-dire, comment acquérir et mémoriser les connaissances que l'on a sur un domaine d'application donné).
- la mise en oeuvre d'un "savoir-faire" par le système (c'est-à-dire, comment utiliser ces connaissances pour résoudre les problèmes rencontrés dans le domaine d'application).

L'appréhension et la représentation du savoir peuvent être menées en s'appuyant sur des méthodes et des outils plus ou moins formels,statiques (logique [DAHL 82], [ONER 82], [GALL 78], [KOWA 79b], réseaux sémantiques [KOWA 79a], [BRAC 77]...) ou dynamiques (programmes ...).

Pour notre part, compte tenu de notre objectif en matière d'informatique pour non spécialistes, nous userons des unes et des autres. Nous utiliserons ainsi, deux formes de représentation :

- l'une formelle, exprimée en termes de spécifications de types abstraits ([ADJ 78], [DER & FI 79], [LISK 75]...). Elle permet de caractériser, de façon rigoureuse, les objets du système et leurs transformations.
- l'autre graphique, réseau sémantique et réseau d'entités et d'associations ([TARD 75], [CHEN 75],...) , à l'intention des utilisateurs non spécialistes. Elle permet de "visualiser" simplement une grande partie de la connaissance du système, tout en en "cachant" la description formelle.

Par ailleurs, la diversité du savoir nous amènera à distinguer deux formes de "représentation informatique". Nous aurons :

- d'une part, une "méta-base" qui décrit, à l'intention du système, le savoir recueilli sur un ou plusieurs domaines .

Elle est le résultat de la "compilation" des spécifications que nous venons d'évoquer.

- d'autre part, une base concrète (ou base de données physique) qui contient les occurrences de faits apparus ou d'objets présents dans le domaine d'application.

Le savoir-faire du système résidera, alors, dans ses capacités à utiliser l'une et l'autre pour répondre aux demandes de ses utilisateurs. Il délivrera des réponses

- extraites "directement" de la base concrète,
- ou en mettant en oeuvre un processus logico-mathématique qui utilise des informations (règles) de la méta-base et des données de la base.

Nous avons concrétisé ce premier aspect du savoir-faire à l'aide de VEGA ([CHAB 82]). C'est un système expérimental qui

- prend en compte des spécifications abstraites de données et de programmes,
- rend disponible un langage de manipulation de données,
- permet de bénéficier des capacités de déduction de PROLOG ([MELO 76], [COLM 82], [CHABJ 82]).

Un second aspect du savoir-faire concerne les aptitudes du système à s'accomoder d'expressions incomplètes de demandes. (Une requête exprimée dans un contexte fait abstraction d'un certain nombre d'informations qui devront être explicitées avant que le système n'essaie d'y répondre).

Dans ce cadre, nous avons conçu et réalisé un système qui admet des expressions incomplètes de requêtes. Il en assure la complétion (ou compréhension) avec la participation de l'utilisateur.

Ainsi, notre travail comporte :

- des aspects outils de représentation du savoir

Dans la première partie de cette thèse, nous passerons en revue ceux qui sont le plus souvent utilisés dans le domaine des bases de données (chapitre I/A) et des systèmes experts (chapitre II/A). Nous présenterons un chapitre I de la troisième partie (chapitre I/C), les concepts que nous utilisons pour représenter le "savoir".

- des aspects utilisation du savoir par le système

Ceci concerne à la fois les mécanismes qu'il doit mettre en oeuvre pour satisfaire les demandes en information et les outils (langages) qu'il offre à ses utilisateurs pour les exprimer.

On trouvera, dans le chapitre I/A, une présentation des langages de manipulation de données associés à quelques modèles de représentation.

Les chapitres II/A et III/A décrivent :

- l'architecture et les fonctionnalités des systèmes experts et des systèmes de données déductifs. Quelques uns, parmi ces derniers, sont présentés dans le chapitre III/A.
- les mécanismes mis en oeuvre durant le processus de déduction.

Le système VEGA est décrit au paragraphe III du chapitre I/B.

Le chapitre III de la partie C détaille nos propositions, en mettant "face à face" la présentation de nos expérimentations en VEGA et la présentation d'une solution dans un SGBD relationnel "papier". Cette façon de procéder nous permet de mettre en relief ce qui sépare les systèmes existants de celui projeté.

Le chapitre IV/C est consacré à la présentation de l'utilisation du système à l'aide d'expressions incomplètes de requêtes. On y présente l'outil d'expression des requêtes et le système de compréhension. Ce chapitre contient également des réflexions sur des extensions envisageables du système de compréhension.

- enfin des aspects méthode, tant pour élaborer une représentation du savoir, que pour la mettre sous une forme "intelligible" par des outils informatiques. Nous montrons (chapitre I/C) comment, parvenir progressivement

  - à une représentation abstraite (statique, indépendante des contraintes technologiques) du savoir en mettant en oeuvre des techniques de spécification de types abstraits,
  - puis à une représentation en termes d'objets informatiques (structures de données et programmes).

Les techniques et concepts utilisés, pour les spécifications abstraites, sont présentés au chapitre I de la partie B. Le deuxième chapitre de cette même partie expose leurs apports attendus dans le domaine des systèmes déductifs.

Les expérimentations que nous allons décrire pourront sembler hybrides, du fait qu'elles ont été effectuées à l'aide de logiciels hétérogènes (en l'occurrence VEGA et son interface avec PROLOG qui répond à certains de nos besoins (validation de spécification, déduction ...) et SINDBAD que nous avons réalisé, en PASCAL sur MICRAL 9050, pour en satisfaire d'autres (création de méta-bases, compréhension de requêtes et génération de "programmes PROLOG" rudimentaires). Cette hétérogénéité nous permettra de situer les limites de ces logiciels et de présenter une architecture fonctionnelle et organique d'un système logiciel "idéal" qui concrétiserait totalement nos propositions.

Le bilan de nos expérimentations et la présentation de quelques prolongements possibles de ce travail feront l'objet de la conclusion de la thèse.

Note de lecture : Les références à des paragraphes sont exprimées dans cet écrit, sous la forme :
p/c/s (paragraphe p du chapitre c de la partie s).

PARTIE A

CHAPITRE I : LES SYSTÈMES DE GESTION DE BASES DE DONNÉES "TRADITIONNELS"

I - INTRODUCTION

Les bases de données et leurs logiciels de gestion (systèmes de Gestion de Bases de Données ou SGBD) sont apparus en début des années 60. Leurs objectifs étaient :

- de réduire ou d'éliminer la redondance des données par leur regroupement au sein d'un fonds de données commun à plusieurs applications : La BASE de DONNEES.
- de mettre à la disposition de l'utilisateur des outils logiciels pour la description des données et de leurs relations (Langage de définition de données ou LDD).
- de réduire la dépendance entre données et programmes en
  - dissociant l'aspect description de l'aspect utilisation ,
  - dégageant le programmeur des préoccupations relatives à l'organisation des données sur leur support et ce en mettant à sa disposition des primitives de haut niveau pour lui permettre l'accès et/ou la modification de ses données (rechercher, ajouter, modifier, supprimer).
- de permettre à des informaticiens et à des non-informaticiens d'accéder et d'utiliser les données de la base, en mettant à leur disposition des langages ad hoc.

Dans les paragraphes qui suivent, nous nous attachons à détailler les aspects description et manipulation de données.

II - LES NIVEAUX DE REPRESENTATION DES DONNEES

Le but d'une base de données est de recevoir, conserver et restituer de l'information. Les informations de la base décrivent la connaissance que l'on a sur une partie d'un monde réel.

En ce sens, elle est un modèle de ce monde, évoluant avec lui pour en être toujours une image fidèle. Les états du modèle reflètent ainsi la connaissance acquise sur ce monde et son évolution.

Cette connaissance peut être de diverses natures. Elle concerne :

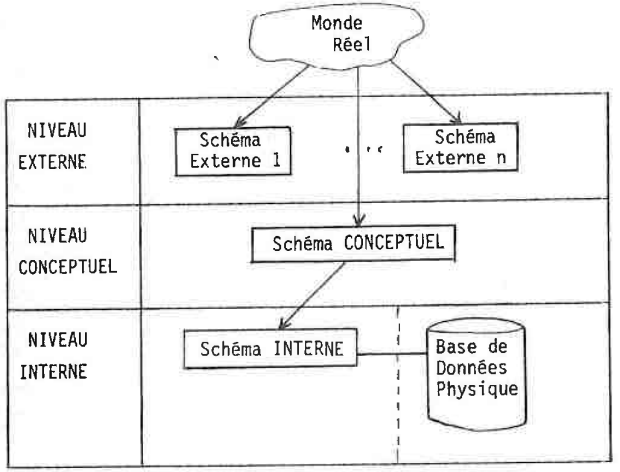
- des faits et des objets élémentaires
  ex : 12 ans, 32 ans, SOCRATE est un homme, 125 FF...
- des règles simples exprimant des propriétés des objets élémentaires
  ex : A toute personne est associée une situation matrimoniale.
- d'autres plus élaborées
  ex : Si une personne a déjà été 'MARIEE', sa situation matrimoniale ne peut plus être "CELIBATAIRE".
- ou encore des règles de calcul de faits à partir d'autres
  ex : • Montant Bénéfice = Prix de vente - Prix de revient
  • L'adresse des enfants mineurs est la même que celle de leurs parents.

La nécessité de représenter cette connaissance a donné naissance à divers formalismes ou Modèles de représentation des données.

Nous allons présenter les plus connus.

II.1.- Les niveaux de représentation

Le rapport intérimaire d'ANSI/SPARC, [ANSI 75] issu d'une réflexion pour la proposition de normes en matière de Bases de données et de SGBD, préconise la distinction de trois niveaux de représentation (schématisés et définis ci-dessous), communément admis depuis la publication du dit rapport.



(A chacun des niveaux est associé un schéma (plusieurs pour le niveau externe) du même nom).

a) Le niveau conceptuel

C'est un niveau de représentation abstraite d'un monde réel observé. Son intérêt est de fournir une modélisation de ce monde qui soit indépendante de toute contrainte technologique et qui pourrait constituer un outil de dialogue entre les utilisateurs potentiels de la future base de données et le(s) concepteur(s).

b) Le niveau externe

Si le schéma conceptuel est global, au sens où il définit une

représentation du monde observé, les schémas externes, quant à eux, représentent des sous-mondes. Chacun définit le sous-monde qui concerne une application ou une classe d'applications.

c) Le niveau interne

C'est celui de la représentation physique de la base. On y tient compte des contraintes matérielles et logicielles pour fixer le mode d'implémentation des objets de la base, des structures d'accès particulières (index , fichiers inverses,...)...

Le schéma interne élaboré est issu de la transformation du schéma conceptuel (par des règles que définissent généralement les méthodes de conception). Donc, seul le schéma conceptuel donne lieu, sous sa forme interne, à implémentation physique. Les schémas externes constituent, pour leur part, une vue logique qu'a une (ou une classe d') application sur la base. Ils permettent de ne définir que les seules informations nécessaires à l'application. (Ils sont, de ce fait, déjà un moyen d'assurer un certain degré de confidentialité).

L'expression de chacun des niveaux est faite dans un formalisme qui,

- pour le niveau interne est celui du SGBD utilisé (C'est son langage de définition de données ou LDD),
- pour les niveaux conceptuel et externe est celui du modèle de représentation des données utilisé.

d) Un niveau intermédiaire : le niveau logique

Pour des raisons essentiellement méthodologiques, il est souvent nécessaire d'introduire un quatrième niveau se situant entre le niveau conceptuel et le niveau interne. Il s'agit d'un niveau dit logique [TARD 76 - FLOR 82], plus adapté à la prise en compte

de paramètres quantitatifs (volume de données, fréquence des opérations, coûts de traitement et de stockage...). Ces paramètres et les estimations qu'ils permettent de faire pourront aider à la détermination de l'organisation physique de la base ainsi que celle de chemins d'accès privilégiés.

II.2.- Les modèles de représentation et la sémantique des données

La modélisation d'une base de données est un processus d'expression des phénomènes d'un monde réel observé, qui utilise les concepts d'un modèle de représentation. Il s'agit concrètement de passer d'une perception de faits, d'objets... à leur représentation par des données.

Ce "monde" est observé à un instant donné, ou en un certain nombre d'instants. Cependant il évolue, communique avec d'autres mondes , la base qui le représentera devra être apte à refléter cette vie, cette évolution et la communication avec d'autres mondes.

Un des problèmes les plus cruciaux est de faire ressortir le "sens", la signification, des phénomènes observés à travers leur représentation. Chacun des modèles conceptuels prétend le permettre. Pour essayer d'expliciter cette notion de sémantique, nous ne nous hasarderons pas dans une tentative de formalisation, mais nous nous appuyerons plutôt, d'après l'idée de [KERS 76], sur une analogie avec une méthode d'analyse en linguistique : l'analyse dite prédicative. Elle vise à identifier, dans une phrase, les prédicats et leurs arguments.

Dans une phrase, on reconnaît généralement une structure prédicative principale et des structures secondaires qui modifient la première. Ainsi certaines de ces structures secondaires peuvent être vues comme équivalentes aux fonctions adjectives et adverbiales de la syntaxe qui, respectivement, qualifient les arguments et modifient les prédicats.

Exemple

(1) - "Des foyers d'activité et de loisirs organisent des stages" : on peut reconnaître dans cette phrase uns structure principale ORGANISATION-DE-STAGE.

(2) - "Des foyers d'activité et de loisirs organisent des stages de natures diverses (ski de fond, ski alpin, canoë-kayak...) sur des lieux d'activité pendant des périodes déterminées" : nature des stages, lieu et période modifient la structure principale.

Nous verrons que certains modèles s'appliqueront à appréhender une sémantique dite de surface alors que d'autres iront plus en profondeur.

Dans le premier cas, (2) serait représenté par une "structure" unique du type ORGANISATION-STAGE (FOYER, STAGE, LIEU, PERIODE) alors que dans le second cas, on détaillerait davantage la représentation en considérant, par exemple :

- "le qui" : Foyer d'activité
- "le quoi" : Stage
- "le où" : Lieux d'activité
- "le quand" : Périodes déterminées.

En d'autres termes, dans le second cas, on explicite les structures dites secondaires.

III - LES MODELES DE REPRESENTATION DES DONNEES

III.1.- Les modèles de type entité/association (E/A)

III.1.1.- Notion d'entité

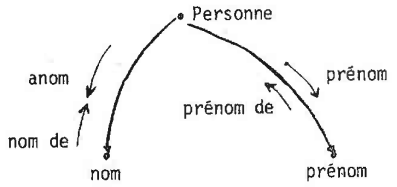
On peut définir une entité comme un objet abstrait ou concret ayant une existence dans le monde observé. Certains auteurs (TARDIEU, CHEN) la caractérisent par ses propriétés (exprimées en termes de

données élémentaires), d'autres (ABRIAL) par les relations (ou associations) qu'elle peut avoir avec d'autres entités.

Exemple

(1) - Dans le modèle individuel [TARDIEU & al 75], une entité PERSONNE se décrit par ses attributs NOM, PRENOM, NO-IMMATRICULATION...

(2) - Dans Data semantics [ABRIAL 74], PERSONNE, NOM, PRENOM... sont des entités (catégories) caractérisées par leurs associations.



Donner une définition de cette notion n'est en définitive pas chose aisée. Nous citerons néanmoins quelques caractéristiques communément admises :

- Une entité a une existence propre
- Elle est abstraite ou concrète
- Elle appartient à une famille d'objets de sa nature (Nous appellerons cette famille ENTITE-TYPE)
- Au sein de cette famille, tout membre (que nous appellerons OCCURRENCE de l'ENTITE) est identifiable, c'est-à-dire désignable sans ambiguïté.

III.1.2.- Notion d'association

Une association est un moyen d'expression de liaisons "séman-

tiques" entre des entités-type. Elle se caractérise par sa dimension (ou degré ou n-arité) et ses cardinalités.

- La dimension d'une association est le nombre d'entités-type qu'elle met en liaison.

  ex : • Une personne POSSEDE des voitures : l'association POSSEDE (entre personne et voiture) est de dimension 2.

  • Un fournisseur fournit des produits à des clients : FOURNIT est de dimension 3 (FOURNISSEUR, PRODUIT, CLIENT).

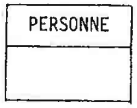
- La cardinalité est un couple de valeurs (n,m) désignant respectivement le nombre minimum et maximum de fois qu'une même occurrence d'entité peut être "impliquée" dans une association.

  ex : Dans l'association POSSEDE, les cardinalités peuvent être
  - (0,n) pour PERSONNE (une personne peut ne pas posséder de voiture (0) comme elle peut en avoir plusieurs (n)).
  - (0,1) pour voiture (une voiture peut ne pas avoir de propriétaire ou, si elle en a, il est unique).

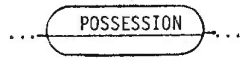
Notes :

1.- Nous utiliserons le graphisme suivant pour symboliser entité-type et association (graphisme emprunté à TARD 75) :

• Un rectangle avec une fenêtre comportant un nom représentera une entité-type



Une ellipse comportant un nom symbolisera une association



2.- Les cardinalités seront inscrites du côté de l'entité-type à laquelle elles correspondent.

La notion de cardinalité mérite que l'on s'y attarde pour expliciter davantage la sémantique des associations.

a) Les associations binaires

Une association A entre deux entités-type E1 et E2 peut se définir formellement par deux fonctions (éventuellement multivaluées) inverses l'une de l'autre (que nous noterons f_A et f_A^{-1}) et par un prédicat P_A (qui peut être défini comme une fonction booléenne).

$$f_A : E1 \rightarrow E2$$
$$f_A^{-1} : E2 \rightarrow E1$$
$$P_A : E1 * E2 \rightarrow Bool$$

(En fait les profils de f_A et f_A^{-1} peuvent être différents de ceux ci-dessus, comme nous allons le voir tout de suite).

Les cardinalités associées aux entités-type sont :

α) d'une part un moyen de caractériser la nature des fonctions :

- Une cardinalité minimale de 0 ou de 1 exprime le fait que la

fonction est partielle ou totale, respectivement.

- Une cardinalité maximale de un (1) exprime le fait que la fonction est monovaluée ou, au contraire, multivaluée si la cardinalité est supérieure à 1.

Les profils de f_A et f_A^{-1} deviennent alors

f_A : E1 $\rightarrow$ E2^n

f_A^{-1} : E2 $\rightarrow$ E1$^{n'}$, où n et n' sont les cardinalités maximales respectives de E1 et E2.

Notations :

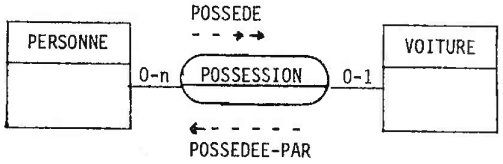
1.- Nous noterons $\longrightarrow$ une fonction totale monovaluée

$- - \rightarrow$ une fonction partielle monovaluée

$\longrightarrow\!\!\!\!\rightarrow$ une fonction totale multivaluée

$- - \rightarrow\!\!\!\!\rightarrow$ une fonction partielle multivaluée

2.- Dans la représentation graphique utilisée, quand nécessaire, nous représenterons f_A et f_A^{-1} en faisant apparaître leur nom et leur "sens" d'application.



β) et d'autre part, un moyen de définir des contraintes d'intégrité

En effet, pour préserver la cohérence de la base, on se devra (utilisateur ou SGBD) de veiller à ce que la nature des fonctions ne soit pas altérée par les opérations de mise à jour.

Ainsi par exemple, dans le cas d'une association de cardinalités ((1-n),(n-m)) l'ajout d'une occurrence de l'entité-type E1 (de cardinalité (1-n)) devra s'accompagner de la mise en liaison avec au moins une occurrence de l'entité-type E2 (ayant la cardinalité (n-m)). On devine alors aisément la succession des événements. S'il n'existe pas d'occurrence appropriée de E2 (c'est-à-dire telle que P_A(e1,e2) avec e1 $\in$ E1 et e2 $\in$ E2), il faudrait en créer ; si E2 participe à d'autres associations pour lesquelles sa cardinalité minimale est de un, il faudrait exécuter le processus pour les occurrences de E2 ainsi créées....

Nous aurons l'occasion d'évoquer une nouvelle fois ce problème dans les chapitres à venir. Contentons-nous, pour le moment, de faire deux remarques à propos de l'utilisation des modèles binaires pour l'élaboration du schéma conceptuel d'une base de données.

Remarque 1

La première remarque concerne le fait que l'observation du monde réel permet souvent de déceler des associations de dimension supérieure à deux. Comment alors, les représenter ?

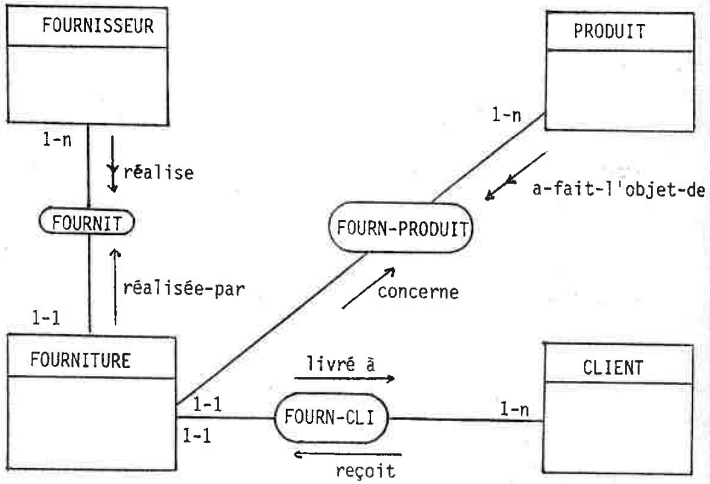
Exemple : Dans "Un fournisseur fournit des produits à des clients", nous pouvons voir trois entités-type : FOURNISSEUR, PRODUIT et CLIENT et une association FOURNITURE qui les relie.

Face à ce type d'association, la solution généralement préconisée par les auteurs des modèles binaires est la création d'une entité-type représentant l'association en question et, la mise en relation de cette entité-type avec les autres.

Ainsi l'exemple ci-dessus serait modélisé à l'aide de

- quatre entités-type (FOURNISSEUR, PRODUIT, CLIENT, FOURNITURE)
- et trois associations dénommées :

- FOURNIT entre FOURNISSEUR et FOURNITURE exprimant le fait que l'acteur d'une fourniture est le FOURNISSEUR.
- FOURN-CLI indiquant le receveur d'une fourniture.
- FOURN-PRODUIT désignant l'objet de la fourniture.



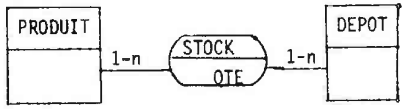
Remarque 2

Parfois les associations comportent des informations qui les qualifient. Comment peut-on les incorporer au sein du schéma ?

Exemple : • Des produits sont stockés dans des dépôts.
• A chaque produit est associé son niveau de stock dans les dépôts où il se trouve.

Certains modèles permettent d'inclure ces informations au sein de l'association [CHEN 75, TARD 75].

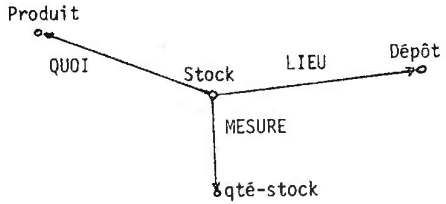
(1)



D'autres utilisent le même artifice que ci-dessus.

On introduit une entité-type que l'on associe aux informations qui la caractérisent.

(2)



Ces formes de représentation explicitent davantage la sémantique des associations.

Enfin, on notera que la schématisation (1), dans la remarque (2), exprime la structure principale alors que la seconde (2) explicite les structures secondaires. (cf. § II.2/I/A).

b) Les modèles n-aires (n ≥ 2)

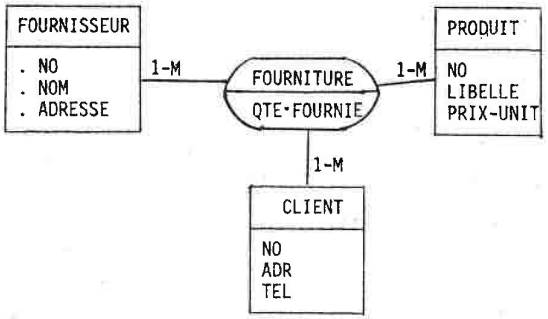
Des modèles tels ceux définis par CHEN [CHEN 75] et TARDIEU & al [TARD 75] autorisent l'expression d'associations de dimension supérieure à deux.

Pour CHEN, une entité est une "chose" identifiable (ce peut être, par exemple, une personne, une entreprise, un événement...). Elle appartient à un ensemble (désigné par ENTITE-TYPE dans notre terminologie) de "choses" de même nature.

Pour TARDIEU & al, une entité-type (appelée INDIVIDU-TYPE) est une famille d'objets abstraits ou concrets, ayant une existence propre ; chaque objet est identifiable au sein de sa famille et les objets d'une même famille sont dotés des mêmes propriétés sur toute "l'étendue du réel" (c'est-à-dire caractérisés par le même ensemble de données élémentaires).

Les associations expriment ici aussi des relations entre entités-type. Elles peuvent être de dimension quelconque (> 2) et peuvent "contenir" des informations qui leur sont propres.

Exemple : "Un fournisseur fournit des produits à des clients dans des quantités déterminées" s'exprimerait par le schéma :



Si on se réfère encore une fois, aux notions de sémantique profonde ou de surface définies en II.2, une telle représentation exprime davantage la seconde notion. Mais CHEN utilise un concept de ROLE d'une entité-type dans une association, qui permet d'en expliciter davantage le sens et d'atteindre, selon nous, le niveau de sémantique dit profond.

Le rôle est l'expression de la fonction tenue par une entité-type dans une association. Ainsi dans l'exemple qui précède, ACTEUR, OBJET, RECEVEUR pourraient être une désignation des rôles respectifs de FOURNISSEUR, PRODUIT et CLIENT dans l'association FOURNITURE.

Notes :

1.- La représentation des associations par des données est généralement faite par un produit cartésien.

FOURNIT : CLIENT×PRODUIT×FOURNISSEUR×QTE-FOURNIE → FOURNITURE

Les rôles des entités sont alors les projections sur leurs composants respectifs :

ACTEUR : FOURNITURE → FOURNISSEUR
OBJET : FOURNITURE → PRODUIT
RECEVEUR : FOURNITURE → CLIENT
QTE-FOURNITURE : FOURNITURE → QTE-FOURNIE

2.- Le passage au niveau des occurrences nécessitera d'introduire la notion d'ensemble des instanciations d'une association (ou association-type, relation ship).
[Cette notion sera formalisée en termes de types abstraits au § I/I/C].

Revenons à notre propos initial, c'est-à-dire celui relatif aux cardinalités. Dans les modèles n-aires, elles déterminent aussi la nature des fonctions associées aux rôles des entités-type.

Ainsi, FOURNISSEUR (1-n) est ACTEUR de 1 à n FOURNITURE.

A l'inverse, une FOURNITURE n'associe qu'un FOURNISSEUR à un CLIENT, et à un PRODUIT pour une QTE-FOURNIE.

Un des inconvénients de la définition en termes de fonctions est le risque d'explosion combinatoire. En effet, outre les "rôles"

acteur, receveur, objet et quantification que l'on peut "extraire" d'une FOURNITURE, on peut être amené à s'interroger sur

- qui a reçu quoi : FOURNITURE → (CLIENT×PRODUIT)
- qui a fourni quoi : FOURNITURE → (FOURNISSEUR×CLIENT)
- qui reçoit quoi d'un acteur donné : FOURNITURE×FOURNISSEUR → CLIENT×PRODUIT
- ...

(Une approche relationnelle permettrait de maîtriser cette explosion combinatoire (cf. § III.2) mais rendrait implicite le concept de rôle).

III.1.3.- Modèles Entité-Association et modèles hiérarchiques et réseau

Dans ce paragraphe, nous allons montrer comment

- d'une part, les modèles hiérarchiques et réseau communément utilisés, semblent plus adaptés à l'expression du niveau de représentation logique,
- et d'autre part, comment on peut traduire dans de tels modèles, des schémas constuits en utilisant un modèle de type entité-association.

a) Les modèles hiérarchiques

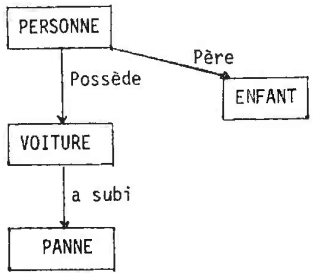
Un schéma construit à l'aide de tels modèles est constitué par :

- un ensemble d'objets (au sens entité-type, appelée RECORD dans la terminologie de CODASYL et que nous emploierons).
- et un ensemble d'associations binaires fonctionnelles (cardinalités ((n,m),(n,1)) avec $n \in \{0,1\}$) appelées SET.

De tels schémas sont représentés par une arborescence dont les

noeuds désignent les RECORDS et les arcs les SET. (Les arcs sont orientés dans le sens inverse de la relation fonctionnelle).

Exemple :



Toutes les relations sont, par définition, de type père-fils.

Une personne possède des voitures et, à l'inverse, une voiture n'est possédée que par une seule personne.

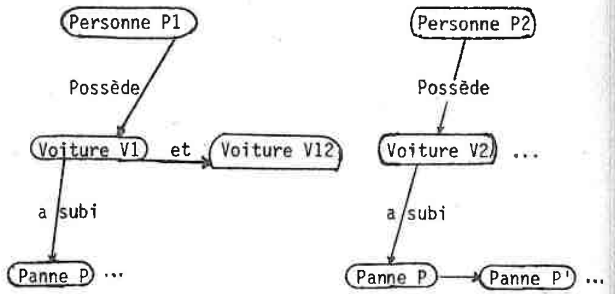
Une voiture peut avoir subi des pannes et, à l'inverse, une panne n'est relative qu'à une seule voiture (!).

Cet exemple de panne permet de mettre en évidence des inconvénients des représentations hiérarchiques :

(1) Tout objet ou famille d'objets, hormis la racine de l'arborescence, n'est connu et accessible que "sous" son père, c'est-à-dire, par exemple, l'accès aux occurrences de PANNE ne pourra être effectué que si l'on connaît une voiture d'une personne.

(2) Une redondance est introduite de ce fait : si des voitures V1 et V2 appartenant respectivement aux personnes P1 et P2 ont subi une panne P, les informations relatives à la panne en question seront "sous" (P1,V1) et (P2,V2), comme le re-

présente le schéma suivant :

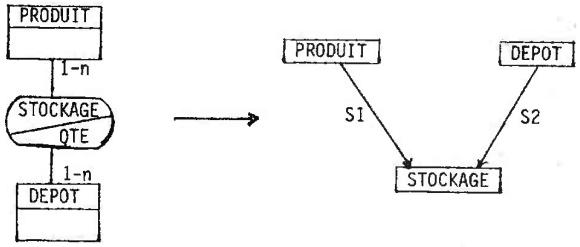


b) Les modèles réseaux

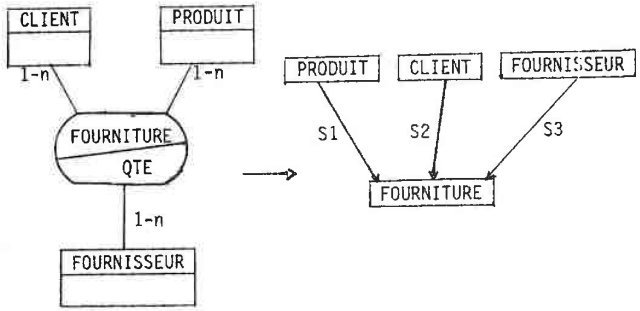
Ils permettent de remédier aux inconvénients des modèles hiérarchiques en autorisant tout objet du schéma à avoir plus d'un "père". Ils permettent l'introduction d'objets ou records "artificiels" pour représenter des associations n-aires (n > 2), ainsi que celles comportant des informations ou encore celles, binaires, qui n'expriment pas des fonctions (au sens mathématique du terme).

Exemples

1.- Association binaire non fonctionnelle



2.- Associations n-aires (n > 2)



Les informations descriptives des associations sont intégrées au sein du record introduit pour symboliser l'association.

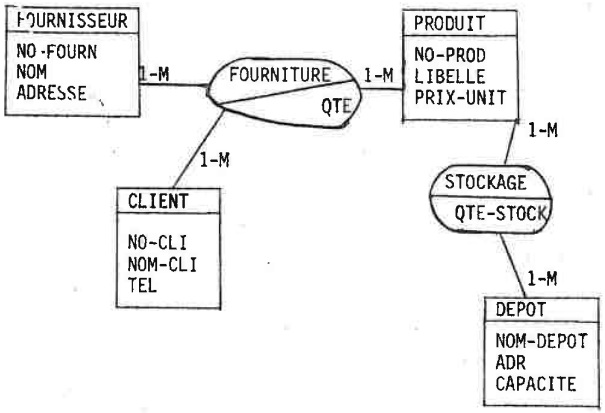
c) Modèles hiérarchiques et réseau et niveau logique de représentation

Nous avions précédemment justifié la nécessité d'un niveau intermédiaire (le niveau logique) entre le niveau conceptuel et le niveau interne, par le fait qu'il devait offrir une forme de représentation adaptée à des tâches de valuation du modèle construit (quantification des volumes, des accès...) afin d'aider à la détermination du modèle d'implémentation (schéma interne).

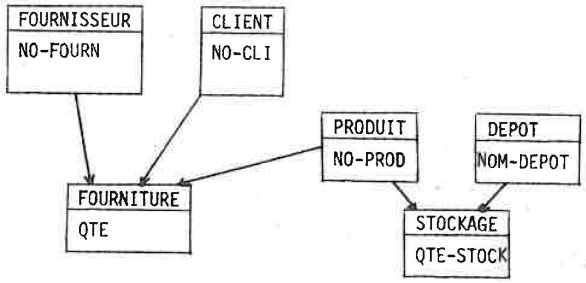
Une autre raison a amené des auteurs de méthodes de conception de bases de données, dont [TARD 76], [FLORY 82], à mettre en évidence ce niveau : c'est le fait qu'il constitue une étape facilitant la définition du modèle d'implémentation. (On pourra trouver dans [OUAD 77] des règles précises de passage d'un schéma conceptuel au schéma logique et une méthode et des outils pour sa valuation réalisés dans le cadre de la méthode MERISE. [BENA 77] est un exemple complet d'application). Les logiciels disponibles ne

gérant que des associations binaires fonctionnelles, le niveau logique permet de s'en rapprocher. On y traduit les associations n-aires ou binaires non fonctionnelles, en des associations binaires fonctionnelles (la phase finale consistera à exprimer le schéma obtenu dans le vocabulaire du langage de définition des données du SGBD utilisé).

d) Un exemple : Considérons le schéma conceptuel suivant



et le schéma logique associé :



Description de ce schéma dans le LDD du SGBD SOCRATE (version 1)

```
ENTITE (      ) FOURNISSEUR
début
    NO-FOURN MOT(5) AVEC CLE UNIQUE
    NOM      MOT(20)
    ADRESSE  TEXTE 2
    ENTITE (    ) FOURNITURE
    début
        REF-CLI REFERENCE FOURN-CLIENT DE UN CLIENT
        REF-PROD REFERENCE FOURN-PROD  DE UN PRODUIT
        QTE DE  1  A  1000
    fin
fin
ENTITE (      ) PRODUIT
début
    NO-PROD ...
    LIBELLE ...
    PRIX-UNIT
    FOURN-PROD ANNEAU
    STOCK-PROD INVERSE TOUT STOCKAGE DE TOUT DEPOT
fin
ENTITE (      ) CLIENT
début
    NO-CLI
    NOM-CLI
    TEL-CLI
    FOURN-CLIENT ANNEAU
fin
```

```
ENTITE (      ) DEPOT
début
     NOM-DEPOT
     ADR
     CAPACITE
     ENTITE STOCKAGE
     début
          NO-PROD
          QTE-STOCK
     fin
fin
```

Quelques commentaires à propos de cette description

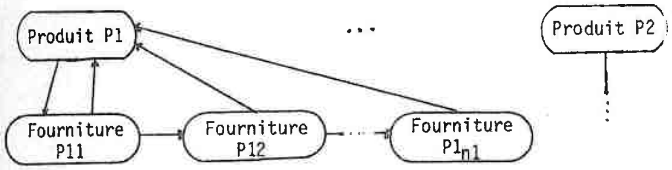
- Les records sont décrits par des blocs ENTITE (n) <nom> début ... fin où n est le nombre maximum d'occurrences possibles de l'entité au sein de la base.
- On associe un type (MOT, TEXTE, intervalle (DE ...A...) ou liste de valeurs) aux attributs des entités.
- Les associations peuvent être décrites par divers procédés :

• L'imbrication d'entités

En incluant l'entité FOURNITURE dans l'entité FOURNISSEUR nous décrivons la relation ACTEUR.

• Les pointeurs ANNEAU-REFERENCE

Le couple de pointeurs Anneau-Référence FOURN-PROD dans PRODUIT et REF-PROD dans FOURNITURE décrit le rôle de PRODUIT en tant qu'OBJET de la fourniture. Ce procédé permet de lier au travers d'une structure en anneau toutes les occurrences de fourniture relatives à un produit.



• La chaîne inverse

STOCK-PROD est une liste de pointeurs logiques vers les entités STOCKAGE associées à un produit.

• "Le lien par clé"

Une clé est un attribut d'une entité dont les valeurs permettent de distinguer les occurrences de l'entité. Elle peut être utilisée pour décrire des associations. Ainsi la présence du NO-PROD dans STOCKAGE établit un lien vers PRODUIT et décrit ainsi le rôle de PRODUIT dans l'association STOCKAGE.

III.1.4.- Conclusion

Au-delà de la description elle-même, nous avons voulu montrer que la représentation des objets et de leurs associations par des données et des pointeurs, d'une part, a pour effet de réduire la lisibilité du schéma et d'autre part, oblige l'utilisateur à "raisonner" et à s'exprimer "au moins" sur le niveau logique et non sur le niveau conceptuel.

Une des idées qui guide notre travail, est d'amener l'utilisateur à travailler sur le niveau conceptuel. En d'autres termes, nous pensons que l'inclusion, dans le logiciel de gestion de la base, de connaissances relatives aux transformations que subit le schéma conceptuel (explicitation des choix faits pour aboutir à un schéma interne), et de mécanismes pour les exploiter, devrait être un moyen qui contribuerait à concrétiser l'idée évoquée.

Par ailleurs, cela présentera comme avantage le fait de n'imposer à l'utilisateur que la connaissance d'un niveau de langage : celui du schéma conceptuel. De ce fait, si l'on ne veut pas imposer un haut degré de spécialisation aux utilisateurs, il faudra prendre soin de fournir un langage simple et lisible.

Nous verrons que notre proposition porte à la fois sur un outil de représentation et sur un outil d'utilisation associé.

III.2.- Le modèle relationnel de données

Introduit par E.F. CODD en 1970, son objectif principal était de proposer un mode de représentation assurant l'indépendance des données (entre elles et vis-à-vis de leurs traitements).

III.2.1.- Notion de relation

- Un attribut A_i est un identificateur auquel est associé un domaine noté $Dom(A_i)$.
- Une relation entre les attributs $A_1,\ldots,A_n$ est un sous-ensemble du produit cartésien $Dom(A_1)\times Dom(A_2)\times\ldots\times Dom(A_n)$. Elle sera notée $R(A_1,A_2,\ldots,A_n)$.
- Un tuple est un élément de la relation. Il est défini par

$$a_1 \times a_2 \times \ldots \times a_n \quad \text{où pour tout } i,\ a_i \in Dom(A_i)$$

(La représentation logique, généralement adoptée, des occurrences d'une relation (les tuples) est un tableau dont les colonnes désignent les attributs et les lignes les tuples).

- La CLE d'une relation est un attribut (ou une combinaison d'attributs) de la relation dont les valeurs sont discriminantes, c'est-à-dire qu'il ne peut exister qu'un seul tuple de la relation associé à chaque valeur que peut prendre la clé.

Exemple : RELATION HEBERGEMENT (NO-HEB, NOMHEB, CAPACITE)

NOHEB	NOMHEB	CAPACITE
1	Les Pingouins	2p
3	Les Flamands	6p
4	Les Rosiers	2p
8	L'abri	12p

III.2.2.- Les opérateurs relationnels

CODD a aussi défini un ensemble d'opérateurs de manipulation des relations permettant de définir une algèbre des relations. Nous ne définirons que deux de ces opérateurs et reportons le lecteur à l'annexe II ou à l'abondante littérature qui existe en la matière ([DELO 82, 78], [FLOR 82], [CADI 75]...) pour une définition plus complète.

a) La projection

Etant donné une relation $R(X,Y)$, où X et Y désignent des attributs ou des listes d'attributs, sa projection sur Y, notée $R[Y]$, est une relation R' de constituant Y, $R'(Y)$ définie comme suit :

$$\forall y,\ y \in R'(Y),\ \exists x,\ x \in R(X,Y) \text{ tel que } x \times y \text{ soit un tuple de } R.$$

Intuitivement, projeter une relation R sur un ou plusieurs de ses attributs revient à constituer une relation R' où ne figurent que les colonnes de R correspondantes aux attributs en question.

b) La jointure

Etant donné deux relations R(X,Y) et S(X,Z), ayant un attribut (ou une liste d'attributs) X en commun, la jointure naturelle de R et S selon X est une relation T(X,Y,Z), notée R*S, définie par

$$\{(x,y,z) | (x,y) \in R(X,Y) \wedge (x,z) \in S(X,Z)\}$$

Les opérateurs définis sur les relations constituent un langage de manipulation que nous présenterons au paragraphe IV de ce chapitre.

III.2.3.- Dépendance de données - Formes normales - Décomposition

La normalisation des relations est un processus de transformation des relations visant à les mettre sous une forme non-redondante et fiable.

Exemple : (1) Relation introduisant de la redondance

Relation PERSONNE (# SECU, NOM, PRENOM, PREN-ENFANT)

# SECU	NOM	PRENOM	PREN-ENFANT
1234	DUPONT	JEAN	LUCIEN
1234	DUPONT	JEAN	THIERY
4567	TOURNESOL	ALBERT	MARCEL
4567	TOURNESOL	ALBERT	PAUL

Le numéro de sécurité sociale, le nom et le prénom du père sont répétés pour chacun des enfants.

(2) Relation "non fiable"

Relation COMMANDE (# CMDE, # PRODUIT, QTE-CMDE, QTE-EN-STOCK)

	# CMDE	# PRODUIT	QTE-CMDE	QTE-EN-STOCK
(1)	01	12 AB	120	200
(2)	01	23 BD	10	300
(3)	02	23 BD	20	300
(4)	03	12 CO	80	122

La redondance est ici introduite de la même façon que dans l'exemple précédent. De plus, il existe un danger de perte d'informations dans le cas d'opérations de suppression de tuples de la relation.

En effet, si nous supprimons les lignes de commande concernant les produits 12 AB et 12 CO (lignes (1) et (4)), les quantités en stock respectives seront aussi effacées de la base.

La définition de formes normales des relations permet de se prémunir contre de tels désagréments. Elles s'expriment par le biais de contraintes que doivent vérifier des attributs des relations : ces contraintes sont appelées dépendances des données.

a) Notion de dépendance fonctionnelle

Etant donné une relation R(X,Y,Z), Z éventuellement vide, on dit que Y dépend fonctionnellement de X par f, noté $X \xrightarrow{f} Y$, (f est le nom de la dépendance fonctionnelle), si pour tout tuple R(x,y,z) et R(x,y',z') de R on a y = y'.

En d'autres termes la connaissance de X détermine au plus un seul Y dans tout tuple de R(X,Y,Z).

Les dépendances fonctionnelles sont ainsi un moyen d'exprimer

des contraintes que doivent vérifier les tuples. Ce ne sont pas les seules contraintes que l'on peut exprimer. Le projet ADONIS [NASS 83] apporte des solutions originales pour la définition et la préservation de contraintes dans le cadre d'une approche multibase relationnelle.

Les dépendances sont aussi utilisées dans le processus de normalisation des relations, dont nous allons décrire le principe, sans nous attarder sur les différentes dépendances de données pouvant exister ([CODD 70], [CADIOU 75], [FAGIN 77,82], [DELO 78,82] et ANNEXE II).

b) Notion de décomposition

La décomposition est un processus réversible visant à obtenir des relations sous forme normale.

Définition :

On dira qu'une relation T(X,Y,Z) est décomposable si et seulement s'il existe deux relations R et S telles que :

1) R et S soient les projections de T sur [X,Y] et [X,Z] respectivement

$R = T[X,Y]$ et $S = T[X,Z]$

2) T est la jointure naturelle de R et de S selon X :

$T = R*S$

(X est appelé domaine de jointure de R et de S).

Exemple : Soit la relation HEBERGEMENT-PAYS (NOMPAYS, ALTITUDE, NOMHEB, CAPACITE, CATEGORIE) caractérisant un pays d'accueil (au sens région) par son nom, son altitude et les hébergements de ce pays (caractérisés par le nom, la capacité (en nombre de lits) et

la catégorie (1*,2*...)).

Exemple d'occurrence de la relation

NOMPAYS	ALTITUDE	NOMHEB	CAPACITE	CATEGORIE
P1	a1	H11	200	*
P1	a1	H12	125	**
P2	a2	H21	87	** NN
P3	a3	H31	82	***

Le fait "q'un hébergement donné ne puisse être géographiquement situé que dans un pays" exprime une dépendance fonctionnelle entre HEBERGEMENT et PAYS que l'on pourra utiliser pour décomposer la relation initiale en deux relations HEBERGEMENT et PAYS,

RELATION HEBERGEMENT (NOMHEB, NOMPAYS, CAPACITE, CATEGORIE)
RELATION PAYS (NOMPAYS, ALTITUDE), ayant comme occurrence :

HEBERGEMENT

NOMHEB	NOMPAYS	CAPACITE	CATEGORIE
H11	P1	200	*
H12	P1	125	**
H21	P2	87	** NN
H31	P3	82	***

PAYS

NOMPAYS	ALTITUDE
P1	a1
P2	a2
P3	a3

c) Les formes normales

Le concept de forme normale des relations a été introduit par CODD. Il a initialement défini trois formes normales.

D'autres formes ont par la suite été définies (3^e BOYCE & CODD NORMAL FORM et 4^e forme normale de FAGIN).

Leur intérêt essentiel est d'être une forme de représentation permettant de réduire la redondance et les risques d'anomalies pouvant se produire en mise à jour de relations non normalisées.

La définition des différentes formes normales est donnée en Annexe II.

III.2.4.- Modèle relationnel et sémantique des données

Dans une approche relationnelle, une base de données est une collection (variable dans le temps) de relations. Cette représentation, d'un monde réel observé, par des relations (ou des tableaux) indépendantes est effectivement un gain dans l'indépendance des données, mais elle présente, à notre avis, un certain nombre d'inconvénients :

- Comme le note A. FLORY dans [FLOR 82], formellement on peut constituer une relation avec n'importe quels attributs, c'est-à-dire même avec des attributs dont la réunion au sein d'une relation n'est la représentation d'aucun phénomène (objet, fait, événement) du monde observé. Quelle serait alors la sémantique d'une telle relation ? (La notion de relation-entité de FLORY & al [FLOR 82] nous semble plus expressive dans ce cas).
- Les dépendances de données peuvent être interprétées selon deux points de vue :

  (1) Elles définissent des contraintes d'intégrité

  "Un Hébergement ne se situe que dans un pays" exprime l'unicité de l'occurrence de l'association entre tout hébergement et sa situation géographique.

  (2) Elles "portent" une sémantique (ex. situation géographique d'un hébergement)

Si leur représentation par des données (domaines de jointure et relations normalisées), réduit la redondance et les risques

d'anomalies, elle a pour inconvénient majeur de rendre cette sémantique implicite. (En effet, comment "voir" dans le couple nom-hébergement×nompays la situation géographique qu'exprimait la dépendance fonctionnelle).

D'autre part, comme nous le verrons dans le paragraphe qui va suivre, cela imposera à l'utilisateur de se "rappeler" des jointures pour ses besoins de manipulation des données.

III.2.5.- Conclusion

Cette critique est du même ordre que celle émise à l'encontre des modèles du type E/A et de leur mode de représentation des associations. Elle nous conforte d'ailleurs dans l'idée de concevoir un système pour lequel la définition de la base ne serait plus uniquement constituée par la définition de son schéma mais comporterait aussi une description explicite des aspects sémantiques, appréhendés au niveau conceptuel, et de leur représentation en termes de données.

L'intérêt en serait, nous le répétons, de décharger l'utilisateur de la connaissance de la représentation pour lui permettre d'exprimer ses besoins sur le niveau de modèle le plus abstrait, à savoir le niveau conceptuel.

L'expression des besoins nécessite, évidemment, de disposer d'un langage. Nous allons, dans ce qui suit, présenter succintement la "philosophie" des langages de manipulation de données dans les systèmes actuels. Nous essaierons d'en dégager celle du/des langage(s) qui nous permettraient d'atteindre l'objectif relaté ci-dessus.

IV - LES LANGAGES DE MANIPULATION DE DONNEES

La volonté de dégager l'utilisation des préoccupations relatives à l'organisation des données sur leur support de stockage,

et aux méthodes pour y accéder, s'est traduite dans les langages de manipulation des données par la mise à la disposition du dit utilisateur de primitives de "haut niveau" lui permettant :

- de rechercher des données (vérifiant certains critères)
- d'en ajouter de nouvelles
- de modifier et d'en supprimer certaines existantes déjà.

Ces opérations s'expriment généralement,

- soit dans un programme écrit dans un des langages de programmation classique (COBOL, PL/1 ou autre) dit alors langage hôte

  ex : • SOCRATE et COBOL
  • MIISFIIT [BOUS 74] et COBOL, ASSEMBLEUR, PL/1 [BOUD 75].

- soit par le biais d'un langage spécifique au SGBD ne nécessitant pas la présence d'un langage de programmation. Ce langage est alors qualifié d'autonome.

On classe traditionnellement ces langages en deux familles :

- celle des langages dits procéduraux où le programmeur doit décrire de façon impérative (par une procédure) les actions qu'il veut effectuer.
- celle des langages dits non procéduraux où le programmeur se contente de caractériser le résultat désiré, sans exprimer la façon d'y parvenir, le SGBD se charge alors du reste.

IV.1.- La manipulation des données dans les modèles hiérarchiques et réseau

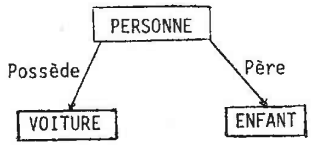
Rappelons que dans ces types de modèle, les "associations" sont binaires fonctionnelles.

Les langages de manipulation associés offrent des possibilités :

- de recherche d'une ou de plusieurs entités
- de recherche d'une entité dépendante (fille) d'une autre
- de parcours de toutes les entités filles
- d'ajout, de modification et de suppression d'entités existantes.

Par le biais de ces commandes, dans un contexte procédural, le programmeur "navigue" au sein de la base, en accédant à une entité racine puis à celles qui en dépendent (en exploitant les associations existantes).

Exemple : Soit le schéma



Rechercher les voitures d'une personne donnée s'exprimerait selon le schéma suivant :

```
(1)  RECHERCHER PERSONNE de NOM = 'XXX'
(2)  RECHERCHER PREMIERE VOITURE (de la personne XXX)
(3)  tantque ¬fin (VOITURE de XXX)
         SORTIR       VOITURE
         RECHERCHER   VOITURE SUIVANTE
     ftque
```

Comme essaie de le montrer ce schéma de programme, le programmeur a la charge du contrôle : recherche, test d'existence, test de fin de parcours d'un ensemble, parcours d'un ensemble ...

Dans un contexte non procédural, on ne se souciera pas de la façon dont on parcourt les ensembles concernés.

La requête de l'exemple précédent s'exprimerait en Pseudo-SOCRATE :

```
Pour une PERSONNE AYANT NOM = 'XXX'
    POUR TOUTE VOITURE X2
        SORTIR MARQUE DE X2
    FIN
FIN
```

OU SORTIR MARQUE DE TOUTE VOITURE DE UNE PERSONNE AYANT NOM = 'XXX'

Dans un cas comme dans l'autre, l'utilisateur explicite les entités à atteindre et le chemin à suivre (parcours de l'association possède) en termes

- des opérations permises par le LMD
- et des identificateurs associés aux objets lors de la définition de la base dans le LDD.

Les requêtes sont ainsi exprimées au "vu" du modèle d'implémentation et non, comme il serait souhaitable, au vu du modèle conceptuel.

IV.2.- La manipulation dans les systèmes relationnels

Les langages de manipulation dans les systèmes relationnels sont classés en deux catégories :

a) Les langages algébriques, qui sont une implémentation des opérateurs de l'algèbre relationnelle.

b) les langages prédicatifs, qui s'appuient sur le calcul des prédicats du premier ordre pour offrir des commandes
   - de recherche d'informations

ex : "Rechercher dans une relation HEBERGEMENT (NOHEB, NOMHEB, CAPACITE, CATEGORIE), tous les hébergements de plus de 4 lits" s'exprimerait en SEQUEL (SYSTEM-R [ASTR 76])

"SELECT NOMHEB FROM HEBERGEMENT WHERE CAPACITE>4"

- de mise à jour

Exemple

(*) INSERT INTO HEBERGEMENT (1234, MENDE,3,2*) permet d'ajouter aux occurrences de la relation HEBERGEMENT, le tuple '1234, MENDE,3,2*' définissant un nouvel hébergement.

(*) "UPDATE HEBERGEMENT
SET CATEGORIE = '3*'
WHERE NOHEB = 1234" a pour effet de modifier le '2*' de l'hébergement 1234 en '3*'

(*) DELETE HEBERGEMENT WHERE CATEGORIE = '1*' efface les tuples associés aux hébergements classés '1*'.

On constate, que, comme pour la manipulation dans les systèmes hiérarchiques et réseau, l'utilisateur doit expliciter dans sa requête la "localisation", dans les relations, des attributs, qu'il veut atteindre. D'autre part, les commandes disponibles sont limitées à celles du langage utilisé (recherche, insertion, modification, suppression, ou jointure, projection, restriction,...).

Ne serait-il pas plus intéressant de disposer d'un langage d'expression de requêtes moins contraignant et "plus riche", c'est-à-dire où

- le vocabulaire (désignateurs des opérateurs, des objets...) ne soit pas limité à celui constitué par les noms des opérations

permises par le SGBD et par les identificateurs utilisés pour décrire le schéma de la base.

- les constructions autorisées (formes des expressions permises) soient les "plus larges" possibles, c'est-à-dire ne "collant" pas à la structure de modèle d'implémentation ?

Explicitons ce dernier point par un exemple.

Considérons les relations PAYS (NOMPAYS, ALTITUDE, POPULATION)
HEBERGEMENT (# HEB, NOMPAYS, CAPACITE, CATEGORIE)
et MONUMENT (# MONMT, TYPEMNMT, NOMPAYS, ERE)

obtenues par décomposition de la relation R (PAYS, MONUMENT, HEBERGEMENT) en utilisant les dépendances

DF1 : PAYS $\longrightarrow\!\!\!\!\rightarrow$ HEBERGEMENT

DF2 : PAYS $\longrightarrow\!\!\!\!\rightarrow$ MONUMENT.

Elles expriment toutes deux les situations géographiques des hébergements (DF1) et des monuments (DF2).

La question "En quel(s) pays trouve-t-on des abbayes du 13^{e} siècle",s'exprime comme suit en SEQUEL :

```
SELECT NOMPAYS, ALTITUDE, POPULATION FROM PAYS
  WHERE NOMPAYS = SELECT NOMPAYS FROM MONUMENT
                    WHERE TYPEMNMT = 'ABBAYE'
                      AND ERE = '13e siècle'.
```

La notion de situation géographique, portée par DF2, n'est plus utilisable en tant que telle dans la requête ; elle est exprimée en termes de domaine de jointure.

La question qui vient immédiatement à l'esprit (et à laquelle nous essayons, dans le cadre de ce travail, d'apporter des éléments de réponse) est de savoir si l'on ne pourrait pas apporter une aide

à l'utilisateur pour parcourir le chemin qui mène de

"En quel(s) pays trouve-t-on des abbayes du 13^{e} siècle"

à "sortir nompays, altitude et population à partir des tuples de la relation PAYS ayant les mêmes valeurs pour l'attribut nom pays que les tuples de la relation MONUMENT où l'attribut TYPEMNMT a pour valeur 'ABBAYE', et ERE la valeur '13^{e} SIECLE'" (qui constitue un paraphrasage de l'expression SEQUEL).

IV.3.- Conclusion : Nos objectifs en matière d'utilisation d'une base de données

Sans aller jusqu'à une utilisation en langue naturelle, nous visons un mode d'utilisation "plus expressif", c'est-à-dire où le vocabulaire utilisé résulterait de l'observation du domaine objet de l'application.

Nous essaierons de ne pas contraindre l'utilisateur à s'exprimer sur la structure de représentation, mais plutôt, sur celle du niveau conceptuel (ou même sans en avoir une parfaite connaissance).

Le système que nous projetons aura alors comme double objectif :

- d'aider l'utilisateur à exprimer ses besoins
- de se charger de leur traduction en des termes intelligibles par le logiciel utilisé pour l'implémentation physique.

Sur un autre plan, l'élargissement du vocabulaire ne visera pas uniquement la désignation des objets et de leurs relations, il touchera aussi les opérations auxquelles ils peuvent être soumis. Par opération, nous entendons tout processus logico-mathématique permettant d'obtenir de l'information (extraction d'informations mémorisées, extraction de nouvelles informations à partir de cellesmémorisées...).

"L'accomplissement de ces objectifs ferait que de tels systèmes ne seraient plus uniquement les mémoires d'un domaine, mais deviendraient des mémoires couplées à un "bout de cerveau" leur conférant un brin (puisse-t-il être épais !) d'intelligence."

Ceci nous amène tout naturellement à parler de travaux similaires menés dans le domaine (bases de données déductives) et dans un domaine voisin, qui est celui des systèmes experts.

CHAPITRE II : LES SYSTÈMES EXPERTS

Parmi les courants actuels visant à produire des systèmes informatiques utilisables par un large public, pas nécessairement spécialisé, celui ayant trait aux systèmes dits experts connaît une forte poussée, ces récentes années [LAUR 82].

Qu'entend-on par système expert, qu'en attend-on, quels en sont les principaux constituants ...? C'est à ces questions que nous allons essayer d'apporter de brefs éléments de réponse.

I - INTRODUCTION

I.1.- Définition

Une première définition d'un système expert pourrait s'énoncer ainsi : c'est un système visant à se substituer à l'expert dans l'accomplissement de sa tâche. C'est un programme de résolution de problèmes nécessitant une spécialisation. Il est dit basé sur la connaissance parce que sa puissance (ou degré d'expertise) et ses performances sont dépendants fortement de l'étendue des connaissances dont il dispose sur le domaine-objet de l'expertise et sur les heuristiques qu'utilise l'expert (auquel il veut se substituer) pour accomplir sa tâche.

I.2.- Quelques caractéristiques de l'activité d'un expert

On pourrait reconnaître à un expert un certain nombre de facultés, parmi lesquelles, celles de :

a) Interprétation

dit intuitivement, cela consiste à analyser des données pour en déterminer le sens. Cette tâche exige de l'expert une aptitude

à prendre en compte des informations partielles, de rejetter celles qui sont douteuses, d'en fournir une interprétation et de la justifier.

b) Diagnostic

C'est un processus de découverte d'anomalies éventuelles au sein d'un système. C'est la tâche la plus délicate et qui comporte le plus de risque. Elle impose d'avoir une bonne connaissance du système, de ses sous-systèmes et de leurs inter-actions. Le degré de difficulté de cette tâche est bien évidemment fonction de la taille et de la complexité du système. (Par exemple, il n'y a aucune commune mesure entre la compréhension d'un système tel que celui de la comptabilité générale et celle d'un système tel que l'être humain).

c) Prévision et planification

Il s'agit d'envisager le futur à partir de modèles du passé et du présent et d'élaborer des plans d'actions afin d'atteindre certains objectifs. Comme toute prévision, cette tâche doit prendre en considération une part d'aléas ; aussi nécessitera-t-elle de l'expert une capacité de prise en compte de divers futurs possibles.

Cette liste de facultés requises de la part d'un expert, bien que non exhaustive, permet de mettre en évidence la diversité des tâches dont l'accomplissement est tributaire d'un certain nombre de facteurs.

I.3.- Facteurs régissant l'activité d'un expert

Un facteur, à nos yeux, essentiel est celui du degré d'expertise, c'est-à-dire celui de l'étendue des connaissances détenues par l'expert.

Un second, conditionnant le premier, est celui de la nature du domaine d'expertise. Il peut se caractériser, essentiellement, par :

* sa dimension
* sa stabilité : est-ce un domaine sur lequel on dispose de suffisamment de connaissances et où celles-ci sont quasi-figées (ex : la comptabilité générale), ou au contraire, est-ce un domaine encore en mouvance (ex : la cancérologie) ?

Selon que l'on se trouve dans l'un ou l'autre des cas, la connaissance est soit bien ou mal cernée, soit susceptible de subir peu ou, au contraire, de nombreux enrichissements. Ceci, bien évidemment, ne manquera pas d'avoir des incidences sur l'activité de l'expert, sur sa façon d'accomplir sa tâche (prise en compte de nouvelles techniques, actualisation de ses connaissances, remise en cause éventuelle de certaines d'entre elles...).

Un troisième facteur est celui de la diversité des problèmes auxquels l'expert est confronté dans son domaine. En effet, la tâche de l'expert se trouve facilitée ou, au contraire, rendue plus ardue, selon qu'elle consiste à résoudre des problèmes d'une même nature ou, de natures différentes mais dont la résolution peut être menée d'une même manière, ou encore qu'elle consiste à prendre en compte des problèmes nécessitant des modes de résolution différents.

Enfin un dernier facteur qu'il nous semble essentiel de mentionner est celui de la masse d'expérience accumulée par l'expert, qui, selon son volume, lui permettra de s'acquitter de sa tâche avec plus ou moins de rapidité.

Ainsi, substituer à un expert humain un outil informatique amènera à incorporer au sein de cet outil des mécanismes tenant compte des différents facteurs et réalisant les diverses tâches mentionnées plus haut.

II - "ANATOMIE" D'UN SYSTEME EXPERT

II.1.- Fonctionnalités

Par analogie avec l'humain qu'il a pour prétention de remplacer, on peut attendre d'un système expert qu'il soit apte à :

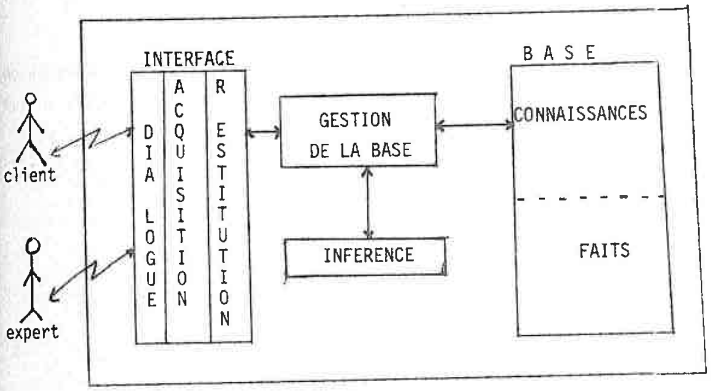
- détenir un certain volume de connaissances (acquis par le biais d'une formation, d'une pratique ou d'une documentation...).
- utiliser ses connaissances et son expérience pour résoudre les problèmes inhérents à son domaine d'expertise. Ceci nécessite généralement, outre la capacité de résolution, celles de :
  - compréhension du problème émis par le "client" en des termes très différents du "jargon" de l'expert.
  - explication, de préférence en des termes intelligibles par le client, du raisonnement tenu, de la voie suivie pour la résolution, ainsi que celle de la solution éventuellement obtenue.
- "rester compétitif" au sens où il doit sans cesse se tenir au courant de l'évolution des connaissances et/ou des techniques relatives à son domaine d'expertise.

II.2.- Une architecture de système expert

Pour concrétiser les objectifs ci-dessus, on attendra d'un système expert :

- qu'il fournisse des outils pour que l'expert humain lui transmette ses connaissances et son expérience.
- qu'il comporte des moyens de structuration et de mise à jour des connaissances.
- qu'il soit apte, face à une expression de problème, à conduire, tout en le documentant, un processus de résolution du dit problème.

Nous regroupons ces différents composants dans le schéma qui suit et nous explicitons brièvement le rôle de chacun d'eux.



Rôle des différents modules :

a) L'interface

Il fournit :

- à l'expert, un langage d'expression lui permettant de communiquer au système ses connaissances et souvent sa façon de les utiliser.
- au client, au langage d'expression de problèmes.

Il assure en outre, la gestion du dialogue qui pourrait s'établir entre le système et l'un ou l'autre des utilisateurs potentiels que sont l'expert et le client.

b) La base

Elle est la mémoire du système et contient :

- les connaissances relatives au domaine
- les règles d'utilisation de ces connaissances
- des faits.

c) Le gestionnaire de la base

C'est l'outil informatique ayant la charge

- de ranger des informations dans la base
- d'en rendre disponible tout ou partie
- de modifier le contenu en enlevant, ajoutant ou modifiant des informations.

d) Le moteur d'inférence

Organe essentiel du système, il a pour rôle de conduire le processus de résolution en se basant sur les connaissances et leurs règles d'utilisation. Le déroulement de ce processus est généralement guidé par une stratégie dont l'efficacité est un des critères déterminant de la puissance et des performances du système.

De l'exposé des fonctionnalités d'un système expert nous retiendrons trois aspects déterminants pour la construction de tels systèmes :

- le premier concerne l'acquisition de la connaissance dans toute sa diversité : faits connus, règles régissant des faits ou des actions, règles d'utilisation de la connaissance pour mener un raisonnement ou rechercher une solution à un problème ...

- le second est relatif à la mise en forme de cette connaissance afin qu'elle soit exploitable par le système.
- enfin le troisième concerne son utilisation pour la résolution de problèmes, la déduction de nouvelles connaissances ...

C'est au détail de ces trois points que nous allons nous attacher dans ce qui suit.

III - NATURE DE LA CONNAISSANCE

Sans prétendre fournir une classification universelle, nous pouvons considérer que la connaissance à incorporer au sein du système est constituée par des :

a) objets du monde réel

ex : Jean, Paul, Voiture 2301 Ra 54...

b) Définitions portant sur ces objets

ex : Jean est un homme ; Jean est bipède ; Jean est mortel

c) Concepts ou généralisations des objets ou des propriétés de ces objets

ex : Tout homme est mortel
Tout homme est bipède

d) Relations exprimant des propriétés des éléments de base ou traduisant des relations de cause à effet. Cette connaissance peut être affectée de coefficient de vraisemblance ou de corrélation avec des situations données.

ex : S'il fait froid et que votre voiture ne démarre pas, il y a de fortes chances que l'état de la batterie en soit la cause.

e) Heuristiques, que l'on peut définir comme étant des éléments innés ou acquis (par l'expérience, la documentation..) servant à guider le choix d'actions à entreprendre (ou à éviter). Elles spécifient généralement leurs conditions d'application.

f) méta-connaissance qui "intervient à plusieurs niveaux par paliers successifs". Il s'agit d'abord de savoir ce qui est su et quel coefficient de confiance lui accorder, quelle importance donner à une information élémentaire par rapport à l'ensemble des connaissances" [LAUR 82].

Cette connaissance doit être transmise au système qui l'organisera et la mettra sous une forme qu'il saura exploiter. La transmission est faite par le biais d'un langage de communication ("situé" dans l'interface sur le schéma 1) que l'on voudrait aussi naturel que possible afin de réduire au maximum la contrainte informatique pour des non-informaticiens.

La forme interne de représentation (tableaux, listes, index ...) adoptée ne nous intéressera pas dans cet exposé. Nous nous préoccuperons davantage des formes logiques de représentation.

IV - REPRESENTATION DE LA CONNAISSANCE

La diversité de la connaissance pose un problème épineux quant à sa représentation. A cette diversité, il convient d'ajouter :

- le caractère fragmentaire des connaissances
- leur nature parfois déclarative (comme l'expression de faits ou la définition d'objets du monde réel) parfois procédurale (comme, par exemple, l'expression d'une suite d'actions à entreprendre pour atteindre un objectif). Comme nous allons le

voir, certaines formes de représentation s'attachent davantage à appréhender l'aspect déclaratif, d'autres l'aspect procédural, alors que d'autres essaient d'intégrer les deux.

IV.1.- Les représentations déclaratives

IV.1.1.- Le calcul des prédicats du premier ordre (CP1)

L'utilisation du langage de la logique pour représenter la connaissance se justifie par son assise formelle, et par le fait qu'il dispose de mécanismes (règles d'inférence, déduction logique...) [cf. Annexe I] permettant d'inférer de nouvelles connaissances à partir de celles existantes.

Dans cette forme de représentation, les faits et assertions sont exprimées sous la forme de formules bien formées (wff) [cf. Annexe I]. De nouveaux faits peuvent être déduits de ces wffs par l'utilisation d'axiomes et de règles d'inférence. Les problèmes sont généralement résolus par des méthodes de preuve de théorèmes basées sur le principe de résolution [ROBI 65].

Ainsi, si W est l'ensemble des wffs décrivant la connaissance relative au domaine d'expertise, un problème P sera exprimé sous la forme d'une formule w dont il s'agira de démontrer la validité dans W.

Nous détaillerons davantage le processus dans le paragraphe traitant des méthodes de résolution (§ 5 de ce chapitre).

IV.1.2.- Les règles de production

Elles se présentent sous la forme d'une liste de conditions (ou prémisses) et d'une liste d'actions :

Si Condition 1$\wedge$Cond 2$\wedge$...$\wedge$Cond n alors Action 1$\wedge$...$\wedge$Action m

Elles sont utilisées en relation avec l'espace de travail, c'est-à-dire celui qui contient la description de l'état du sys-

tème à un instant donné de son fonctionnement. La satisfaction des prémisses d'une règle dans un état donné du système aura pour effet de déclencher la liste d'actions mentionnées dans la règle.

Ce mécanisme, simple a priori, est étendu par des procédures de choix à activer dans les cas où plus d'une règle ont leurs prémisses satisfaites à instant donné. Nous y reviendrons plus en détail lorsque nous parlerons de l'utilisation des connaissances dans les systèmes utilisant ce type de représentation.

IV.1.3.- Les réseaux sémantiques

Ce sont des graphes orientés où :

(1) Les noeuds sont des points où sont mises des informations relatives à des éléments appartenant au monde observé. Ces informations peuvent désigner :

- un objet : JONN, LOZERE
- une assertion : Le Brésil est un pays
- des événements : La terre a tremblé le 10/10/80 à 13 H 23'.

Généralement, les noeuds représentent une classe d'objets par abstraction de caractéristiques communes aux objets (ou individus) de la classe. Les caractéristiques sont elles-mêmes représentées au sein du réseau par des liens émanant du noeud-abstraction. [Une caractérisation assez précise des noeuds d'un réseau sémantique peut être trouvée dans BRAC 77].

(2) Les arcs sont la représentation de "liaisons sémantiques" entre noeuds. Ils peuvent exprimer notamment :

- l'appartenance d'un objet à une classe

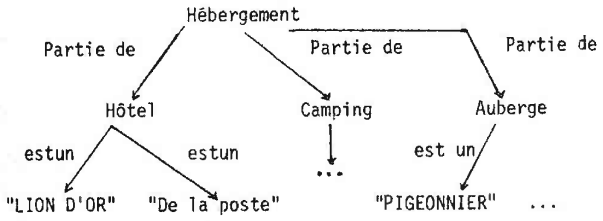
ex : SOCRATE ←ESTUN— Homme

- des propriétés des instances potentielles d'une classe

ex : Homme —EST→ MORTEL

- une notion de sous-classe

ex :



Le lieu PARTIE-DE exprime une relation de sous-classe entre le noeud HEBERGEMENT (désignant la classe de tous les hébergements) et les types d'hébergement en HOTEL, CAMPING et AUBERGE.

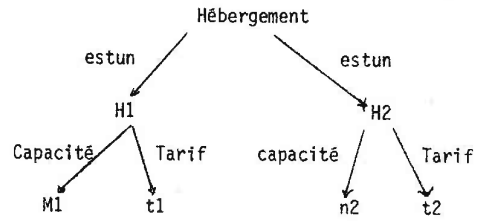
On se doit de remarquer que si la représentation des liaisons sémantiques est uniforme, il ne peut en être de même quant à leur prise en compte (ou interprétation) par une machine.

Par ailleurs, ce mode de représentation, souvent utilisé dans le cadre des travaux relatifs aux langues naturelles (*) s'il permet d'exprimer des "connaissances relationnelles", n'offre par contre pas de facilité pour la prise en compte de celles de type procédural ou encore celles incluant des quantifications.

Néanmoins, on peut compléter une description en termes de réseaux sémantiques par des wffs exprimant des règles de déduction de nouvelles connaissances à partir de celles déjà présentes au sein du réseau.

(*) ([COUL 81], [KAYS 81])

Exemple :



Noeuds : {Hébergement, H1, H2, n1, n2, t1, t2}

Relations

estun (Hébergement, H1)
estun (Hébergement, H2)
Tarif (H1, t2)
Tarif (H2, t2)
Capacité (H1, n1)
Capacité (H2, n2)

Règles de déduction

(1) $\forall x,y/\text{est un (Hébergement,x)} \wedge \text{est un (Hébergement,y)}$
$\text{Tarif}(x,v) \wedge \text{Tarif}(y,u) \wedge (v < u) \rightarrow \text{MOINS-CHER}((x,y),x)$

Cette règle exprime la relation de coût entre deux hébergements. Celle qui suit exprime une relation de dimension.

(2) $\text{Capacité}(x,n) \wedge \text{Capacité}(y,m) \wedge (n > m) \rightarrow \text{PLUS-GRD}((x,y),x)$

Il est alors nécessaire de dissocier les relations primitives des autres, d'expliciter, par exemple comme ci-dessus, la composition des relations et éventuellement le mode d'application de celles obtenues par composition.

Dans cet ordre d'idée, KOWALSKI & DELIYANNI, dans [KOWA 79], définissent une forme de réseau, ne comportant que des relations binaires, qu'ils montrent équivalent à l'expression d'un problème sous forme de clauses.

IV.2.- Les représentations procédurales

Si dans les représentations dites déclaratives, les fragments de connaissance étaient délivrés en "vrac", le système ayant pour charge de les relier, de les combiner pour conduire un raisonnement, les représentations procédurales, elles, rendent explicites les relations entre les fragments d'une part, et d'autre part, elles imposent un ordre déterminé dans leur utilisation.

IV.2.1.- Les automates finis

Ce procédé de représentation permet la description d'un domaine en termes d'états et de règles de transition d'un état à un autre. Il nécessite la connaissance :

- des différents états et de leur caractérisation
- des règles de transition et de leurs conditions d'activation.

VM, Ventilator Manager, peut être classé parmi les systèmes utilisant ce type de représentation. C'est un système de surveillance post-opératoire d'un patient nécessitant une assistance respiratoire. Celle-ci est fournie par un "ventilateur mécanique" adapté aux besoins du malade. [FAG 79]

Le modèle de raisonnement du système tient compte d'informations de tests et d'observation du sujet pendant la progression de la maladie ou après des interventions thérapeutiques.

La connaissance est organisée en plusieurs types de règles :

- des règles de transition utilisées pour la détection des changements d'états du patient.
- des règles d'initialisation de contextes. Ceux-ci sont mis en place quand les prémisses d'une règle de transition sont satisfaites. Ils correspondent à des situations ou à des états spécifiques. Les règles spécifient ce qui varie et ce qui reste fixe dans le nouveau contexte (en rapport avec le contexte précédent).
- des règles d'états.
- des règles de thérapie.

VM se limite aux états précédents et suivants. Pour pouvoir raisonner sur des événements plus distants dans le temps, ce mode de représentation devra être étendu pour permettre la prise en compte de divers futurs possibles et non forcément proches.

IV.2.2.- Les programmes

Ils permettent de décrire de façon impérative et dans un ordre déterminé les suites d'actions à entreprendre pour atteindre des buts définis ou pour réaliser certaines fonctions.

Leur activation est faite de façon directe ou par un "démon" (mécanisme de surveillance perpétuelle de la vérification de conditions d'activation d'un programme) ou selon un schéma ("pattern direct invocation").

L'inconvénient majeur de ce type de représentation est sa rigidité et par conséquent la difficulté d'en modifier tout ou partie et aussi de contrôler l'incidence d'une modification locale sur la totalité du système.

IV.3.- La représentation mixte

Comme son nom l'indique, elle vise à permettre l'appréhension

des aspects à la fois statiques (déclaratifs) et dynamiques (procéduraux) de la connaissance.

IV.3.1.- Les Frames

Les frames [BOBR 77, CHAR 78] peuvent se définir comme étant une structure de données représentant un stéréotype d'objet ou de situation. Ils peuvent être constitués :

- d'un niveau fixe de représentation d'entités (objets et/ou relations) devant toujours être présentes ou vraies dans la situation où elles sont décrites.
- de niveaux variables (les slots) destinés à contenir des données dont la présence n'est pas impérative. Il peut leur être associé des dispositifs particuliers. Ainsi dans GUS [BOBR 77], deux types de procédure peuvent leur être attachées :
  - les démons ou procédures activées automatiquement lors de l'instanciation d'un slot (pour, par exemple, vérifier des contraintes sur les valeurs qu'il peut prendre).
  - les servants (ou domestiques) qui eux sont activés à la demande (pour, par exemple, mettre des valeurs prédéfinies dans les slots non valorisés par l'utilisateur).

Exemple : Prototype d'une frame pour un objet DATE (inspiré de [BOBR 77])

```
FRAME  DATE
       MOIS CHAR
       JOUR ENTIER DANS [1..31]
       AN   ENTIER (DEFAUT 1983)
       JOUR-EN-CLAIR (DANS (LUNDI MARDI ... DIMANCHE))
                     (SI-VALORISE TROUVER-DATE A-PARTIR-DE
                                  JOUR-EN-CLAIR)
       EDIT LISTE (JOUR,MOIS) OU JOUR-EN-CLAIR
```

Ce frame comporte diverses clauses :

- celles descriptives de la nature de l'information

  ex : ENTIER, CHAR

- celles exprimant des contraintes

  ex : JOUR DANS [1..31]

- celles qui permettent d'indiquer les valeurs par défaut (notion de servant)

  ex : DEFAUT 1983

- d'autres définissant les procédures à activer et leurs conditions d'activation

  ex : JOUR-EN-CLAIR : si une instance de l'objet date est fournie par indication du nom du jour (Lundi, mardi...), la procédure TROUVER-DATE calculera la date sous sa forme JOUR-MOIS-AN à partir du JOUR-EN-CLAIR indiqué.

- enfin, d'autres définissent, à l'intention du programmeur, les alternatives quant aux formes externes des instances du frame (dans ce cas, ce peut être par l'indication d'un jour de la semaine ou par celle du jour et du mois, l'année étant facultative).

Par ces divers mécanismes, les frames permettent de représenter tout ou partie d'une situation ou d'un objet donné. Ils peuvent être, parfois, reliés pour constituer un réseau de frames pour décrire la totalité de la situation ou de l'objet en question.

Leur utilisation en tant qu'outil de représentation des connaissances présentent à notre avis quelques problèmes, résumés par les deux interrogations ci-dessous :

1.- Comment réduire des situations, prises dans des domaines empiriques, à des situations stéréotypées ? (Ne serions-nous pas inévitablement amenés à en dénaturer quelques

aspects ?).

2.- Comment, du fait de cet empirisme, prévoir tous les démons et servants nécessaires pour obtenir du frame un comportement, qui, s'il ne peut être réel, soit "au moins" réaliste ?

Les questions restent ouvertes ...

IV.3.2.- Les scripts

Ils constituent une spécialisation de la notion de frame. Ils sont constitués par une séquence prédéterminée d'actions définissant une situation donnée. De la même façon qu'en ce qui concerne lesframes ils peuvent être conçus de sorte que des informations absentes sur une de leurs instances puissent être inférées à partir de celles présentes. [SCH 75]

IV.4.- Conclusion

Nous achevons ainsi la présentation du paragraphe concernant la représentation des connaissances. Nous avons essayé d'y insister sur la diversité des phénomènes à appréhender. Nous avons brièvement exposé les principales formes de représentation utilisées en essayant de critiquer chacune d'elles.

Nous ne nous hasarderons pas dans une discussion sur la "meilleure" d'entre elles. Nous pensons, néanmoins, que le choix d'une forme de représentation doit être étroitement lié à la nature du domaine de l'expertise et des connaissances qu'il englobe (diversité, volume, crédibilité, connaissances fortement ou faiblement liées...).

V - UTILISATION DES CONNAISSANCES

Les connaissances représentées et mémorisées le sont en vue d'être soumises à des opérations visant, notamment

- à en rendre disponible, à la demande, tout ou partie
- à en obtenir de nouvelles à partir de celles existantes
- à en élargir l'étendue.

V.1.- Caractérisation de l'utilisation des connaissances

Une des caractéristiques principales des systèmes experts est la façon dont ils procèdent pour essayer de trouver une solution à un problème qui leur serait soumis. Ce mode de recherche de solution est tributaire de nombreux facteurs dont les plus essentiels nous semblent :

a) La taille de l'espace des solutions

Des mécanismes d'optimisation du parcours de l'espace des solutions sont nécessaires, en rapport avec l'étendue de cet espace.

b) La fiabilité des données

Le degré de confiance accordée aux données a aussi une incidence sur le mode de recherche de solutions, au sens où celui-ci devra tenir compte du caractère éventuellement incertain des valeurs de certaines données. Il devra donc comporter des mécanismes qui s'en accommodent.

Un système comme MYCIN [CLAN 81, LAUR 82] associe à ses faits et heuristiques un coefficient de vraisemblance mesurant leur exactitude. Il utilise sa propre méthode de calcul du coefficient de vraisemblance d'un fait déduit. Ce processus lui permet de décider du rejet ou de la préservation de ces faits, en fonction de la valeur du coefficient obtenu.

(D'autres approches, telles l'approche probabiliste ou celle par la logique floue, ont été utilisées pour le raisonnement dans un "monde incertain").

c) La diversité des sources de connaissances

Les sources de connaissances peuvent être diverses. Le système devra alors être capable de les classifier puis de les réutiliser.

Ainsi, HEARSAY II [LESS 77], système conçu pour la compréhension du discours dans des domaines spécialisés, considère différentes sources indépendantes (syntaxique, sémantique, phonétique ...). Une structure de données globale, le "tableau noir" (blackboard), est le moyen de communication et d'inter-action entre les sources de connaissances.

d) La(les) classe(s) de problème à résoudre

Peut-on, dans un domaine donné, suivre le même schéma de résolution pour tout problème rencontré ou au contraire, des schémas différents sont-ils nécessaires, en liaison avec la différence des natures des problèmes ?

Dans R1, système utilisé pour la détermination de la configuration des systèmes VAX à partir d'une commande d'un client, le même schéma de recherche de solution est appliqué : la tâche de configuration est découpée en six sous-tâches indépendantes, exécutées toujours dans le même ordre ; chaque état de l'espace de recherche obtenu, est une instantiation partielle de la configuration en cours d'élaboration. [DERM 80]

Par contre, ABSTRIPS, qui réalise des plans de transport d'objets par un robot, opère de façon différente. Il utilise des plans constitués par des abstractions associées à chaque sous-problème. Les plans se distinguent par leur niveau de détail ; la

résolution est menée "top-down", c'est-à-dire du plus abstrait au plus spécifique. Au sein de chaque niveau, les sous-problèmes sont résolus dans un ordre indépendant du problème.[SACE 74]

e) La disponibilité d'heuristiques

Comme déjà dit, les heuristiques dictent le choix d'actions à entreprendre (ou à éviter) dans des situations données. Leur détermination fait appel à beaucoup d'intuition et d'expérience.

D.B. LENAT, dans [LENA 82], les caractérise par trois propriétés :

α) L'observabilité

La définition et l'évaluation des heuristiques ne peuvent se faire que si l'on est capable"d'amasser"des données et d'en observer le comportement.

β) La continuité

L'environnement doit présenter un caractère de continuité car la validation des heuristiques ne peut se faire si cet environnement subit des changements brutaux.

γ) La stabilité

Des changements perpétuels de l'environnement risquent de faire en sorte que la durée de vie des heuristiques soit trop courte pour permettre de juger de leur utilité et de leur efficacité.

D'autre part, LENAT a déterminé (empiriquement par expérimentation du système AM) trois origines des heuristiques :

- la spécialisation d'heuristiques plus générales existantes
- la généralisation de règles spécialisées
- les analogies que l'on peut faire entre des domaines a priori différents et l'utilisation des mêmes heuristiques dans chacun des domaines.

A ces caractéristiques globales ayant une incidence sur les mécanismes de recherche de solution, il faut bien évidemment ajouter le mode de représentation des connaissances retenu car, même si, comme nous allons le voir dans le paragraphe qui suit, le principe de la recherche de solution peut s'énoncer indépendamment du mode de représentation des connaissances, il n'en demeure pas moins que sa concrétisation nécessitera souvent des techniques ad hoc.

V.2.- Méthodes de résolution de problèmes

S'accomoder des facteurs énoncés plus haut n'est pas une tâche aisée. Aussi généralement les systèmes comportent des mécanismes pour contrôler la résolution, résoudre des conflits, choisir la règle (dans les systèmes de production), l'action (représentation en termes d'états) ou l'axiome (CP1) à appliquer dans une situation donnée, annuler les décisions qui auraient mené à des impasses ou à des contradictions ...

Nous n'avons pas la prétention de faire ici une présentation exhaustive des solutions existantes ([STEF 82] contient une analyse assez complète et étayée par de nombreux exemples des méthodes et techniques utilisées en fonction de la nature du domaine, de celle des données, de la taille de l'espace des solutions...). Nous nous bornerons à la présentation d'une classification des méthodes généralement admise en intelligence artificielle, puis nous nous attarderons quelque peu sur la résolution et son contrôle dans les systèmes de production et dans ceux utilisant le calcul des prédicats.

V.2.1.- Classification des méthodes de résolution

Nous distinguerons trois aspects :

- la méthode à proprement parler
- son mode de déroulement
- son efficacité

Traditionnellement, on considère deux familles de méthodes :

- celle des méthodes dites aveugles, c'est-à-dire procédant par exploration systématique de l'espace des solutions, sans souci d'optimisation de l'exploration. Ce type de méthode n'est envisageable que dans la mesure où l'espace est réduit.
- celle des méthodes dites non aveugles qui, par souci de performances, explorent l'espace des solutions de façon ordonnée et partielle.

Par ailleurs, le déroulement de la recherche est dit opérant :

- en chaînage avant s'il démarre des faits ou des données du problème pour essayer d'aboutir à sa solution (but).
- en chaînage arrière s'il s'exécute à l'inverse du chaînage avant (du but vers l'origine).
- en chaînage mixte, s'il allie chaînage avant et chaînage arrière.

De façon générale, le problème est de doter le système d'un mécanisme performant de recherche de solution. Ses performances se mesurent en termes de minimisation du temps et de l'espace de recherche.

Nous avons ainsi, d'une part un mécanisme d'inférence, et d'autre part, des mécanismes de contrôle de l'inférence.

Nous allons examiner ces deux points dans le cadre de systèmes de production puis de ceux de programmation logique.

V.2.2.- Utilisation des connaissances dans les systèmes de production

La connaissance est représentée sous forme de règles de production.

V.2.2.1.- Schéma de l'inférence

Le mode de raisonnement utilisé est de "type modus ponens" (cf. Annexe 1).

$p, p \rightarrow q \models q$

On peut exprimer intuitivement ce schéma d'inférence de la façon suivante :

si les prémisses (p) d'une règle sont vérifiées et qu'elles déterminent des actions (q), alors déclencher celles-ci.

Le processus de résolution des systèmes qui utilisent ce schéma d'inférence, consiste alors en une série d'applications des étapes suivantes :

(1) Reconnaître la(les) règle(s) applicable(s) dans une situation donnée.

(2) En choisir une parmi elles.

(3) Déclencher sa partie action et répéter le processus jusqu'à atteindre le but poursuivi (en cas de succès) ou explorer toutes les éventualités (cas d'échec).

Le fait de prévoir ou pas en (2) des critères de choix de la règle à appliquer classe la méthode parmi celles dites non aveugles ou au contraire, parmi celles dites aveugles.

V.2.2.2.- Contrôle de la déduction dans les systèmes de production

Il vise à guider le système tout au long de sa recherche afin de l'aider à résoudre des conflits, à opérer des choix "judicieux" de règles à appliquer...

On distingue [LAUR 82] trois grandes familles de structure de contrôle :

- le choix par évaluation

- la recherche exhaustive
- le contrôle par méta-règles.

a) Choix par évaluation

Ce type de contrôle suppose l'existence d'une fonction H dont l'application aux règles candidates permet d'établir un ordre parmi celles-ci.

b) La recherche exhaustive

Elle n'est concevable que dans les cas où les domaines de recherche sont "volumineusement restreints".

d) Le contrôle par méta-règles

Le système peut disposer de méta-connaissances (connaissance sur la connaissance) exprimées à l'aide de méta-règles (cf. § III de ce chapitre).

Ces dernières indiquent au système :

- leurs conditions d'application et, si ces conditions sont vérifiées,
- l'action à effectuer en priorité ou
- l'ordre dans lequel une suite d'actions doit être réalisée pour atteindre un but ou un sous-but donné.

Exemple de méta-règles (extraites de [LAUR 82])

(1) Si l'on recherche une thérapie
alors, dans cet ordre, considérer les règles qui permettent de
1 - acquérir les informations cliniques sur le patient
2 - trouver quels organismes, s'il en existe, sont cause de l'infection
3 - identifier les organismes les plus vraisemblables
4 - trouver tous les médicaments potentiellement utiles
5 - choisir les plus adaptés, en plus petit nombre.

(2) Si 1) le site de la culture est non stérile
et que 2) il existe des règles qui mentionnent dans leurs prémisses un organisme déjà rencontré auparavant chez le patient, et qui est le même que celui dont on recherche l'identité
alors il est sûr (1,0) qu'aucune de ces règles ne peut servir...

Ces deux méta-règles illustrent bien nos propos.

La première est un exemple de règle qui dicte une suite ordonnée d'actions alors que la seconde fournit au système un critère sûr d'élimination de certaines règles.

V.2.3.- La déduction et son contrôle dans les systèmes logiques - Application à PROLOG

L'élaboration de systèmes basés sur la logique a été entreprise dans de nombreux domaines. Nombre de ces systèmes se sont limités à des formes particulières des formules de la logique : les clauses de HORN.

(Dans ce paragraphe, nous ne reviendrons pas sur les définitions des notions de formule, de substitution, d'unification et de consistance données en Annexe 1).

Les clauses de Horn peuvent prendre l'une des trois formes suivantes :

(1) A1 ← B1,B2,...,Bn
(2) A2 ←
(3) ← B1,B2,...,Bn, dont les interprétation respectives sont :

(1) $\forall x,y,\ldots$ si $B1 \wedge B2 \wedge \ldots \wedge Bn$ alors $A1$

(2) $\forall x,y,\ldots$ $A2$

(3) $\neg\exists x,y,\ldots$ tels que $B1 \wedge B2 \wedge \ldots \wedge Bn$, avec $x,y\ldots$ désignant l'ensemble des variables présentes dans les clauses.

Elles peuvent s'interpréter également [KOWA 81, GAL 79]

- en termes procéduraux :

(1) et (2) sont des procédures, la partie gauche désignant l'entête de la procédure et la partie droite le corps.

(3) est un appel des procédures B1, B2,...,Bn.

- en termes de résolution de problèmes

(1) est un opérateur (règle d'inférence)

(2) est une assertion

(3) est le problème à résoudre.

1.- Le principe de résolution

a) Notion de résolvant

Soit deux clauses $A \leftarrow C1,\ldots,Cn$ (1)

et $B \leftarrow A1,\ldots,Ai,\ldots,Am$ (2)

Si une substitution θ est un unificateur de Ai et A alors la clause $B\theta \leftarrow (A1,\ldots,A_{i-1},C1,\ldots,Cn,A_{i+1},\ldots,Am)\theta$ est appelée résolvant de (1) et de (2).

On appellera alors étape de résolution une exécution du processus de calcul du résolvant de deux clauses.

b) Principe de la résolution

Le principe de la résolution trouve ses fondements théoriques dans [ROBI 65] et, particulièrement, dans le théorème qui y est défini. Exprimé de façon intuitive, celui-ci stipule qu'un

ensemble $\mathcal{C}$ de clauses est inconsistant si et seulement si on peut aboutir à un résolvant vide, noté $\square$, par une suite finie d'application d'étapes de résolution.

On dira alors qu'on procède à la réfutation de $\mathcal{C}$. Celle-ci se définit comme une séquence finie $C_1,\ldots,C_n$ de clauses où

1 - $\forall i,\ i \in [1,n]$, C_i est - soit dans $\mathcal{C}$,
- soit un résolvant de clauses précédentes dans la séquence.

2 - C_n est la clause vide $\square$.

Concrètement, il s'agit de montrer que, dans l'ensemble $\mathcal{C}$ de clauses initiales, l'introduction de la négation de la clause définissant le problème à résoudre introduit une inconsistance dans $\mathcal{C}$ (qui se traduit par un résolvant vide $\square$) ; ceci permet de se prononcer sur la forme "positive" de la dite clause. (ie si $\neg P$ est faux alors P).

c) Exemple : le système PROLOG

PROLOG ([COLM 82], [CHABJ 82], [MELO 76]), est un système de programmation logique implémentant le principe de résolution. Conçu initialement pour la compréhension des langues naturelles, il a été par la suite utilisé dans de nombreux domaines (bases de données [DAHL 82], systèmes experts [BUND 79, DINC 79], systèmes de types abstraits de données [CHAB 82]...).

Les clauses y sont exprimées par une partie positive (partie gauche) et une partie négative (partie droite).

(1) +A1-B1-B2-...-Bn

(2) +A2

(3) -B1-B2 ...-Bn

Un programme PROLOG se présente alors sous forme d'une suite

de clauses. La demande d'exécution est exprimée par une clause de la forme (3) (partie positive vide).

Par ailleurs, PROLOG met à la disposition de l'utilisateur des prédicats dits évaluables. Ils permettent de réaliser :

- des entrées-sorties de caractères, de termes, de clauses
- ajout et suppression de clauses
- opérations arithmétiques
- des contrôles de la résolution ('/' et IMPASSE).
- ...

Exemple 1

(1) + Personne (1)
(2) + Personne (2)
(3) + Personne (3)
(4) + Père (1,2)
(5) + Père (2,3)
(6) + GP(x,y) - Père(x,z) - Père (z,y)

On doit se prononcer sur GP(1,3)

On exprime sa négation : - GP(1,3) et on applique le mécanisme décrit ci-dessus.

(1) GP(1,3) et GP(x,y) s'unifient par l'unificateur $\theta_1 = \{x/1, y/3\}$ c'est-à-dire en substituant dans GP(x,y) 1 à x et 3 à y.
D'où par application du méta-théorème dans (6)

(a) - Père(1,z) - Père(z,3)

(2) Suivant le même principe on obtient successivement

(b) - Père(1,2) - Père(2,3) avec $\theta_2 = \{z/2\}$

(b') - Père(2,3)

puis (c) □ avec $\theta_3 = \emptyset$.

Nous ferons remarquer que le même procédé permet de répondre à des problèmes d'existence de valeurs de variables pour lesquelles un prédicat (ou une suite de prédicats) est vraie.

Exemple 2 : Quel est le grand-père de la personne 1

- GP(1,x)-SOR(x)

La succession des opérations est la suivante :

(1) - Père(1,z) → Père(z,x) avec $\theta_1 = \{x/1; y/x\}$

(2) - Père(1,2) → Père(2,x) avec $\theta_2 = \{z/2\}$

(3) - Père(2,3) avec $\theta_3 = \{x/3\}$
qui permet de conclure.

Une seconde remarque s'impose : Dans l'exemple 1, l'obtention de (c) est faite après une première tentative (infructueuse) d'unification de (b') et (4) puis celle (réussie) de (b') et (5). (Les clauses (4) et (5) sont appelées clauses candidates). En fait, à chaque étape de résolution on peut être confronté à un problème de choix de clauses à unifier. Ce choix peut concerner :

- le littéral au sein de la formule à démontrer :

$- L_1 - L_2 - L_3 \ldots - L_n$: Quel L_i choisir ?

- la clause parmi l'ensemble des clauses candidates (cet ensemble est constitué par l'ensemble des clauses ayant en partie gauche le même symbole de prédicat de même arité que le L_i choisi).

Le paragraphe qui suit concerne ce problème. Sa solution contribuerait à accroître les performances du système en lui évitant le parcours systématique de l'ensemble des clauses (avec les obligations de retour-arrière ("back-tracking") en cas d'aboutissement à une impasse).

2.- Mécanismes de contrôle dans les systèmes logiques

Un certain nombre de stratégies peuvent être développées pour réduire l'effort de recherche de solution dans les systèmes logiques. ([NILS 71], [ROBI 65], [GAL 79], [LAUR 82]).

a) Stratégies par simplification

Elles visent à réduire l'ensemble de clauses par élimination de certaines clauses et/ou certains littéraux. Parmi les règles d'élimination utilisées citons :

- la suppression des tautologies (présence simultanée de $P \wedge \neg P$ dans une clause).
- l'évaluation précoce de prédicats (par exemple, la formule P(x)vQ(y)vEgal(1,2) sera remplacée par P(x)vQ(y),Egal(1,2) étant toujours évalué à Faux).

b) Stratégies par raffinement

Dans ces stratégies les résolutions sont faites en choisissant des clauses vérifiant certains critères.

Les critères de raffinement peuvent concerner la nature des clauses (ainsi dans la "P1-réfutation", chaque étape de résolution met en jeu des clauses dont l'une au moins ne comporte pas de négation de littéral.

D'autres permettent d'établir un ordre au sein de l'ensemble des clauses candidates. (ainsi dans "l' unit-preference", l'ordre est celui induit par la taille des clauses, c'est-à-dire le nombre de littéraux qu'elles comportent ; on mène la résolution unité (clause à un seul littéral) contre unité, puis en cas d'échec unité contre double-unités, etc...).

GALLAIRE & LASSERE [GAL 79] proposent un méta-langage pour le contrôle de la déduction dans PROLOG. Dans ce méta-langage, deux types de méta-règles permettent d'exprimer le contrôle, soit sur

la clause candidate (par définition de priorités entre les éléments de l'ensemble des clauses candidates : règle METAL), soit sur le littéral candidat à l'unification (en en gelant certains jusqu'à ce que des variables essentielles pour la poursuite de la résolution, soient instanciées : règle METAX).

V.3.- Conclusion

Dans ce paragraphe, nous avons essayé de mettre l'accent sur les caractéristiques des méthodes de résolution, en distinguant mécanisme d'inférence et mécanisme(s) de contrôle.

Nous n'insisterons pas davantage sur la nécessité de disposer d'un outil puissant d'inférence.

CHAPITRE III : LES BASES DE DONNÉES DÉDUCTIVES

I - INTERET ET OBJECTIFS DES SYSTEMES DE DONNEES DEDUCTIFS

L'adjonction de mécanismes de déduction à des systèmes de gestion de données a comme principaux objectifs, outre ceux visés par les systèmes classiques (ajout, suppression, modification, recherche d'informations mémorisées), :

- de permettre l'extraction d'informations non stockées explicitement.
- d'étendre le langage du système pour une plus grande adéquation aux besoins de l'utilisateur.
- de fournir, éventuellement, des justifications des réponses faites à des demandes émises par l'utilisateur.

La différence avec les systèmes dits classiques se ressent tant au niveau représentation qu'au niveau utilisation des informations.

Dans les systèmes classiques, les ensembles d'objets (c'est-à-dire les occurrences des entités-type et des associations, ou les tuples des relations) étaient définis et mémorisés de façon explicite, par énumération de leurs éléments. (On dira en Extension). Des règles qui définissent des relations entre des objets ou qui expriment des contraintes sont généralement utilisées pour maintenir la cohérence de l'ensemble des données mémorisées.

Prenons l'exemple des personnes et de leurs liens de PATERNITE et de GRANDE-PATERNITE.

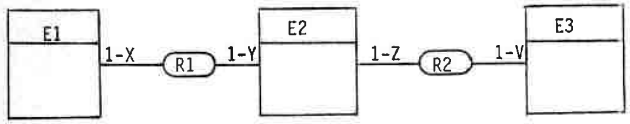
(1) Dans un système classique, on définira les ensembles de personnes, "qui est père de qui", "qui est grand-père de qui" en énumérant respectivement les personnes, les couples (père, fils) et les couples (petit-fils, grand-père).

PERSONNE = {JEAN, PAUL, JACQUES, ARTHUR}
PERE = {(JEAN, PAUL), (PAUL, JACQUES), (PAUL, ARTHUR)}
GRAND-PERE = {(JACQUES, JEAN), (ARTHUR, JEAN)}

(2) Le fait qu'une personne ne puisse avoir qu'un père est vu comme une contrainte d'unicité portant sur les occurrences de la relation PERE.
Le SGBD devra préserver cette unicité durant toute la vie de la base.

(3) D'autres concepts inhérents au modèle conceptuel utilisé peuvent et doivent s'interpréter comme des contraintes. En effet, si nous nous remémorons les diverses natures des associations définies au § III/I/A en liaison avec les cardinalités individuelles, celles exprimant des fonctions totales (cardinalité minimale de 1) devraient induire des actions spécifiques lors d'opérations de mise à jour les concernant ou concernant des entités-type qui y participent.

Voyons comment sur un exemple



Soit trois entités-type E1, E2, E3 et les associations R1 entre E1 et E2 et R2 entre E2 et E3. Les cardinalités minimales égales à 1 expriment le fait que toute occurrence
- de E1 doit être associée par R1 à 1 à X occurrences de E2

- de E2 doit être associée à 1 à Y occurrences de E1 par R1 et à 1 à Z occurrences de E3 par R2.
- de E3 doit être en relation avec 1 à V occurrences de E2 par R2.

Préserver la cohérence des données impose de préserver la nature des associations. Cela amène à définir des mécanismes ad hoc.

Par exemple, l'adjonction d'une occurrence de E1 à l'ensemble d'occurrences qui existent dans la base, doit s'accompagner de la définition des occurrences de E2 associées. S'il n'en existe pas, dans l'état courant de la base, il sera nécessaire de les ajouter, et, en faire de même quant à l'association des nouveaux-venus avec des occurrences de E3, qui, s'ils n'existent pas...

Ce processus dit de propagation des opérations de mise à jour, n'est, à notre connaissance, supporté par aucun des systèmes commercialisés. La raison est due, à notre avis, d'une part à la difficulté de son contrôle (risques de cycle évidents...) et d'autre part, d'un point de vue pratique, au fait que la propagation pourrait s'avérer une contrainte à l'utilisation, dans la mesure où elle impose à l'utilisateur de disposer d'une masse d'informations (celles requises tout au long du chemin de propagation) qu'il ne peut soit prévoir a priori, soit détenir au moment où il effectue son opération.

Des solutions ponctuelles et "localisées dans l'espace" existent cependant.

Ainsi, dans les systèmes hiérarchiques, (tels que SOCRATE ou MIISFIIT [BOUS 74]) lors de la création d'une "entité-mère", le système demande à l'utilisateur s'il veut créer des "entités-filles". De même, la suppression d'une entité-mère entraîne celle de "ses filles".

(4) Dans un contexte d'interrogation, l'exploitation des règles comme "TOUT père d'un père est un grand-père", de part leur mode de représentation en extension, n'était faite que par exploration de l'ensemble des occurrences les matérialisant. Au mieux, si l'on voulait qu'il en soit autrement, c'est-à-dire les utiliser pour "calculer" les valeurs des objets qui les vérifient, on est amené à exprimer le processus de calcul par le biais d'un programme.

En d'autres termes, les systèmes classiques n'offrent pas d'outils généraux d'expression et d'utilisation de telles règles.

En résumé, ces règles peuvent être utilisées :

- en tant que contraintes (c'est le cas de l'unicité du père ou celui des contraintes de nature syntaxique).
- en tant que règles permettant d'effectuer un calcul logico-mathématique (tout père d'un père est un grand-père)
  - soit pour obtenir de nouvelles informations à partir de celles mémorisées (par exemple, pour savoir qui est grand-père de qui ou de qui une personne donnée est le grand-père, à partir de la mémorisation des personnes et des couples (père, fils))
  - soit pour engendrer de nouvelles informations à mémoriser. Par exemple, si l'on choisit de mémoriser les couples (petit-fils, grand-père), l'ajout d'une occurrence de PERSONNE au sein de la base doit entraîner le calcul de son grand-père et de ses petits-fils et leur adjonction à la base.

[Dans le cadre de ce travail, nous ne nous sommes pas préoccupés de ce type de problème. Il est clairement posé dans [NIC 82] et une solution est proposée dans le système BDGEN du même auteur [NIC 82b]].

Pour atteindre ces objectifs, il ne s'agira plus de ne décrire (et instancier) que l'ensemble des faits et des objets du domaine observé mais il faut de plus décrire et exploiter les règles régissant ce domaine.

On appellera alors Base de données intentionnelle la description, dans un formalisme donné, des objets, des relations entre ces objets et des règles qui régissent le domaine observé.

Son extension désignera les occurrences mémorisées d'objets et de relations.

De même, on dira qu'un ensemble est défini en extension si ses éléments sont énumérés et en intention si une ou plusieurs règles caractérisent l'appartenance d'un élément à cet ensemble (ces règles régiront le "calcul" des éléments de l'ensemble ainsi défini).

Par analogie avec la logique [cf. Annexe I], l'intention peut être vue comme un système formel pour lequel l'extension est un modèle, c'est-à-dire une interprétation définie sur le domaine-objet de l'application. Les règles régissant le domaine constituent l'ensemble des axiomes et des règles d'inférence grâce auxquelles pourront être conduites les déductions. (Nous aurons l'occasion de parler dans le paragraphe suivant, de systèmes conçus de la sorte).

II - PRESENTATION DE QUELQUES SYSTEMES EXISTANTS

Les expériences menées dans le domaine se distinguent par le fait qu'elles dissocient ou pas

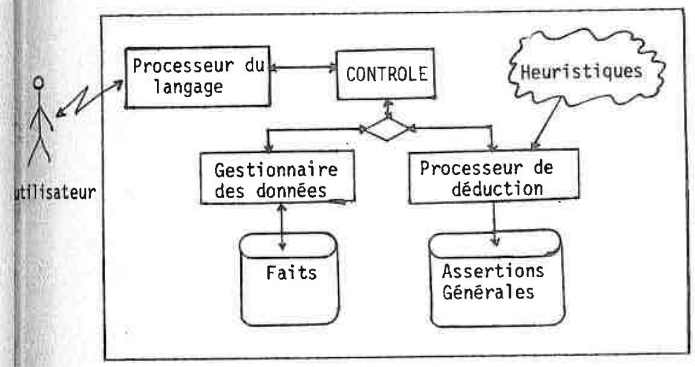
- l'intention et l'extension de la base
- le mécanisme de déduction du mécanisme de recherche des données dans l'extension.

Nous allons rapidement, passer en revue quelques uns de ces systèmes.

II.1.- Le système DADM (KELLOG & al)

L'approche prise par KELLOG & al est de dissocier le processeur de déduction de celui de gestion de données et de séparer les faits (extension) des assertions générales (intention).

La figure ci-dessous schématise l'architecture du système.



Les objectifs visés sont, outre l'extraction d'informations explicites :

- l'extension du langage du SGBD pour l'adapter aux besoins des utilisateurs.
- l'adjonction de mécanismes de justification des réponses.
- la réponse à des requêtes de "haut niveau" (de la forme "qu'adviendrait-il si"...).
- fournir des réponses "conditionnelles" (ie, dans certains cas, ne pas répondre par oui ou par non, mais fournir des réponses de la forme : "il n'existe pas de X répondant à la demande, mais Y pourrait faire l'affaire, moyennant les conditions suivantes..).

Les informations utilisées pour la déduction sont constituées par :

- un fichier des assertions
- un fichier des substitutions
- un fichier de "trucs" sémantiques (ou heuristiques basées sur la sémantique des données)
- un graphe de connection des prédicats qui définissent les assertions. Les noeuds du graphe représentent des occurrences de prédicats et les arcs des interactions déductives possibles entre les noeuds.

Le processeur de déduction utilise ce graphe selon la stratégie dite mixte (chaînage avant et chaînage arrière).

II.2.- DEDUCE (CHANG)

DEDUCE est implémenté en LISP ; c'est un langage pour bases de données relationnelles. Il permet d'exprimer des requêtes, des axiomes, des règles dites de préférence et des heuristiques.

CHANG distingue deux types de relation :
- les relations de base qui sont celles définies en extension.
- les vues qui sont des relations définies par des axiomes.

L'évaluation des requêtes est faite en deux phases :

a) la première transforme la requête en une formule ne contenant que des relations de base (elle utilise une méthode de réécriture [JOUA 83]).

b) la seconde fait évaluer la formule ainsi obtenue par un SGBD relationnel.

Une des originalités de DEDUCE est de permettre la définition de préférences, qui sont, en gros, des expressions de critères de priorité. Elles peuvent être définies tant dans les requêtes que dans les axiomes.

D'autre part, des notions de requête et de sous-requête permettent de structurer les questions adressées au système. Enfin,

le langage autorise l'utilisation des fonctions d'agrégation (somme, moyenne, maximum...) dans les requêtes.

Exemples

(1) Trouver le salaire moyen des employés du département "JOUET" ou "CHAUSSURE"

```
(QUERY
    (SUBQUERY
    (EMPLOYE # SALAIRE = X DEPT = Y)
    (OR (EQ Y JOUET)(EQ Y CHAUSSURE)))
(COMPUTE *(AVG X)))
```

Notes :

- Le caractère '*' précède les attributs ou les variables dont il faut restituer les valeurs à l'utilisateur.
- Le caractère '#' joue, dans une sous requête, le même rôle que '*'.
- QUERY et SUBQUERY sont des mots-clés qui introduisent, respectivement, une requête et une sous-requête.
- (COMPUTE *(AVG X)) permet de calculer la moyenne des X, X étant la variable associée à SALAIRE.

(2) "A quel(s) étage(s) doit-on s'adresser pour avoir des téléviseurs, des GRUNDIG de préférence".

```
(QUERY
(SITUATION DEPT = X  *ETAGE)
(VENTES DEPT = X ARTICLE = 'TV')
$(FOURNITURE FOURNISSEUR = 'GRUNDIG' DEPT = X
                ARTICLE = 'TV')).
```

Le '$' précède le critère de préférence. Dans ce cas, le système produit comme résultat le ou les étages où sont vendus des téléviseurs. Le(s) étage(s) où on trouve des GRUNDIG sont donnés en tête de la liste des résultats, du fait du critère de préférence.

(3) Exemple de définition d'axiomes

```
(AXIOM PERE (PERE X1 X2)
            (PERE X2 X3)
            IMPLIQUE
            (GRDPERE (X1 X3))
```

Les axiomes ainsi définis peuvent être utilisés dans des requêtes.

Exemple : (QUERY (GRDPERE JEAN *X)).

Remarque :

Nous sommes tentés d'émettre, à l'encontre de DEDUCE, les mêmes critiques que celles émises à l'encontre des langages relationnels (cf § IV.2/I/A).

En effet, on constate que la désignation de la localisation des attributs dans les relations (SALAIRE et DEPT dans la relation EMPLOYE, DEPT et ETAGE dans SITUATION...), et des domaines de jointure (DEPT dans SITUATION, VENTES et FOURNITURE, ARTICLES dans VENTES et FOURNITURE...) sont à la charge de l'utilisateur.

La différence qui existe avec les langages relationnels classiques est que l'on peut exprimer les requêtes sous forme de listes. On utilise alors la puissance du langage (LISP) d'implantation de DEDUCE (notamment les possibilités d'appel de fonction) pour concrétiser les objectifs de déduction.

Néanmoins, les possibilités de définition et d'utilisation d'axiomes sont d'un apport certain pour les capacités déductives du système.

II.3.- QUERYLOG (BAZEX & al)

QUERYLOG est un système qui dissocie le mécanisme de déduction de celui de gestion des données.

Il opère sur une base d'informations constituée par :

- une base de données (gérée par le SGBD relationnel MRDS) et son schéma conceptuel.
- un ensemble de règles de déduction (géré par un RESOLVEUR qui fonctionne selon le même principe que PROLOG ([COLM 82], [CHABJ 82], [MELO 76]).

Le moniteur de QUERYLOG assure le transfert des requêtes vers MRDS ou le RESOLVEUR, selon le cas.

Par son LMD, QUERYLOG permet d'accéder à la base de données et de manipuler les règles de déduction. Les requêtes, émises sous forme de clauses de HORN [Annexe 1] sont :

a) soit traduites immédiatement en requête QUEL, puis transmises à MRDS pour exécution : c'est le cas des requêtes qui ne font pas référence à des règles de déduction, c'est-à-dire qui ne comportent pas de littéral qui soit partie gauche d'une règle de déduction.

Exemple : Soit la base d'information constituée par :

```
        RELATION PERSONNE (NOM, PRENOM, AGE)
        RELATION PERE (NOM1 , NOM2 )
    et  GRANDPERE(X,Y) ← PERE(X,Z), PERE(Z,Y)
```

A la base de données, qui comporte les relations PERSONNE et PERE, est associé son schéma. Le schéma de la base d'information comporte les relations de la base et celle(s) définie(s) par la (les) règles de déduction.

La requête "Sortir les personnes de plus de 30 ans" s'exprime en QUERYLOG par :

← PERSONNE(X,Y,Z), BOOL(Z > 30),I(X)

Elle ne fait appel à aucune règle de déduction. Elle est, alors, aussitôt traduite en :

"RANGE OF P is PERSONNE
RETRIEVE P.NOM WHERE P.AGE > 30"

puis transmise à MRDS.

b) soit soumise au RESOLVEUR avant d'être exprimée en QUEL

Exemple : "QUEL est le grand-père de JEAN" s'écrit
← GRANDPERE(X, JEAN),I(X).

Soumise au RESOLVEUR, elle est transcrite en une expression ne comportant que des relations de base, soit :

←PERE(X,Z), PERE(Z,JEAN),I(X)

puis traduite en la requête QUEL :

RANGE OF(P,PP) IS (PERE, PERE)
RETRIEVE P.NOM WHERE (P.NOM2 = PP.NOM1) AND (PP.NOM1 = JEAN)

Par ailleurs, QUERYLOG offre des possibilités de mise à jour de la base, par la :

α) création de nouveaux tuples

Exemple : "PERSONNE(HERCULE, POIROT, 45)←" est traduite en
"STORE PERSONNE (HERCULE, POIROT, 45)"

β) modification et suppression

Exemple 1 : "← PERSONNE(X,Y,Z),MODIFY(Z:Z+1)" est traduit en
"RANGE OF P is PERSONNE
RETRIEVE (P.AGE) MODIFY+1"

Exemple 2 : ← PERSONNE(X,Y,Z),BOOL(Z>60),I(X),DELETE permet de détruire les " tuples des personnes" de plus de 60 ans.

Remarque : Dans sa version initiale, QUERYLOG interdit toute modification de données implicites [ex :← GRANDPERE(X,JEAN),MODIFY(X, ARTHUR)]. En outre, il ne prend pas en compte des règles de déduction récursives [comme par exemple :

FRERE(X,Y) ← FRERE(X,Z), FRERE(Z,Y)]

Une des originalités de QUERYLOG est sans contexte la possibilité qu'il offre de modifier ou de supprimer des données par "unification". De plus, la distinction qui est faite entre les requêtes primaires (celles ne faisant pas appel à des règles de déduction) et les requêtes non primaires doit être un moyen d'améliorer la performance du système (pas de "transit" par le RESOLVEUR pour les requêtes primaires).

Une approche analogue (couplage de PROLOG avec un système relationnel) est suivie par MARQUE & al [MARQUE 83] pour doter le SGBD PEPIN de capacités déductives.

II.4.- MONT-LOZERE (CHABRIER & al)

Dans le cadre d'un appel d'offres lancé par l'ADI, nous avons réalisé une maquette de systèmes déductifs pour l'information et la promotion du tourisme en LOZERE. Ce système était destiné à un large éventail d'utilisateurs, pas nécessairement spécialisés. L'accent devait être mis sur le "confort d'utilisation", aussi bien sur le plan du langage que sur celui du matériel utilisé. Le projet devait être mené par une équipe pluridisciplinaire (agents touristiques, informaticiens, ergonomes, personnes chargées d'une interface images...).

L'originalité de notre approche a été d'investiguer la construction d'un tel système en utilisant les types abstraits de données. La maquette a été réalisée en utilisant le système VEGA [CHAB 82], un système de vérification et d'implémentation de types abstraits de données, basé sur PROLOG.

Nous reviendrons plus en détail sur les notions d'abstraction et sur VEGA dans le chapitre suivant. Contentons-nous pour le moment de donner un bref aperçu sur la maquette elle-même.

a) La définition des données

On peut définir des ensembles d'objets d'un même type en extension et en intention.

La définition en extension est faite par l'association d'un type ensembliste (ensemble, liste...) à la classe d'objets à définir.

```
Exemple :← RUN(DEFINE(HEBERGEMENT=TUPLE(NOMM:STRING;CAPACITE:INT;
                                        CATEGORIE:INT)))
         ← RUN(DEFINE(ENS-HEB=SET(HEBERGEMENT)))
         ← RUN(DEFINE(PAYS=TUPLE(NOMPAYS:STRING;ALTITUDE:INT)))
         ← RUN(DEFINE(ENS-PAYS=SET(PAYS)))
         ← RUN(DEFINE(HEB-DU-PAYS=TUPLE(NOMM:STRING;NOMPAYS:STRING)))
         ← RUN(DEFINE(ENS-HEB-DU-PAYS=SET(HEB-DU-PAYS)))
```

Ces six clauses définissent respectivement

- un hébergement comme un tuple dont les constituants sont typés
- ENSHEB comme un ensemble (SET) de tuples HEBERGEMENT
- un PAYS et un ensemble ENSPAYS de PAYS
- la situation géographique d'un hébergement par un couple (nom hébergement, nom pays). ENS-HEB-DU-PAYS est l'ensemble des couples ainsi définis.

La définition en intention consiste à caractériser une classe d'objets par des propriétés qui expriment "ce que signifie l'appartenance à la classe".

Exemple : On peut être intéressé par les hébergements situés à plus d'une certaine altitude.

Plutôt que de les définir comme un ensemble dont on énumérerait les éléments, on le fera à l'aide d'une clause dont la partie gauche désignerait l'ensemble et la partie droite traduirait les propriétés de ses éléments.

```
Ainsi, HEB-EN-ALTITUDE(*ALT) ← SET-HEB(*NOMH,*1,*2),SET-HEB-
                               DU-PAYS(*NOMPAYS,*NOMH),
                               SET-PAYS(*NOMPAYS,*ALTITUDE),ST(INT(*ALTITUDE>*ALT)),
                               SORTIR(*NOMH,*NOMPAYS)
```

est l'expression en intention des "hébergements situés à une altitude supérieure à *ALT".

Cette expression pourra être utilisée, alors, en n'indiquant que sa partie gauche dans laquelle on substitue à *ALT, l'altitude effective.

```
ex : ← HEB-EN-ALTITUDE(1000)
```

Elle peut également servir à définir d'autres intentions.

```
ex : "ALTITUDE-ET-LOISIRS(*NOMLOISIRS,*ALT) ← HEB-EN-ALTITUDE(*ALT),
                LOISIRS(*NOMLOISIRS),LOISIRS-DU-PAYS(*NOMPAYS,
                        *NOMLOISIRS)"
```

permet de s'informer sur les hébergements situés à une altitude supérieure à *ALT et à proximité desquels on peut s'adonner au loisirs *NOMLOISIRS.

Un des principaux avantages de cette forme de définition, est son aspect "incrémental". On peut, en effet, définir progressivement un système de données en explicitant la nature des objets qu'il concerne, leurs inter-relations et certaines déductions où ils sont impliqués.

b) L'utilisation du système

Dans le paragraphe qui précède, nous avons déjà évoqué une des formes d'utilisation permises.

Mais, on doit faire remarquer que la phase de définition, qui a fait l'objet du paragraphe précédent, ne fait qu'informer le système sur la "forme" des objets de la base. Pour que ceux-ci soient manipulables, ils doivent d'abord être instanciés. Le système offre, dans ce cadre, des opérations spécifiques à chacun des types des objets.

C'est ainsi que sur le type SET, on pourra :

- définir des ensembles (vides).

RUN(DEFINE(ENS-HEB(SETHEB ← 0))) définit un ensemble SETHEB de type ENS-HEB que l'on initialise à vide par "← 0"

- y insérer des éléments

RUN(ENS-HEB(SETHEB ← ADD(SETHEB,LES PINGOUINS,5,2))) ajoute le tuple (LES PINGOUINS,5,2) à l'ensemble SETHEB.

- y rechercher des éléments...

Ces diverses opérations peuvent s'utiliser dans toute expression de requête. L'évaluation est faite à l'aide du mécanisme de résolution de PROLOG (cf. chapitre III/C).

Par ailleurs, dans des applications telles que celle de MONTLOZERE, il est souhaitable que le système puisse proposer des réponses "plus larges" que celles strictement recherchées. Cette largeur est fonction de l'application. Pour ce qui concerne l'activité de tourisme, le système pourrait, notamment :

- "documenter" ses réponses en les accompagnant d'informations générales sur la région, par exemple.

- émettre des propositions autres que celles recherchées, dans les cas où il n'arrive pas à satisfaire strictement les demandes qui lui sont soumises.

En d'autres termes, dans un cas, on procède à une sorte d'enrichissement des réponses (par exemple, toute offre d'hébergement, produite en réponse à une requête, sera accompagnée d'informations sur les sites alentours, les musées, les services (restauration, santé...) que l'on peut trouver sur le lieu d'hébergement). Dans le second cas, on propose des réponses "équivalentes" à celles recherchées, en cas d'échec lors de la recherche du "produit" désiré. (Par exemple, si on ne trouve pas d'hébergement pour 4 personnes, en 2*, on proposerait des 4 personnes en 1 ou 3*, ou des 5 personnes en 3*...).

L'idée est donc d'adjoindre aux connaissances du système, des informations relatives à la forme des réponses et à la distance qui peut exister entre ces dernières. Le fonctionnement du système serait alors modifié, en ce sens que l'évaluation d'une requête R ne se fera plus selon le schéma :

s'il existe dans la base des informations I répondant à R
alors SORTIR I^+ sinon "il n'existe pas de I pour R"

mais selon le schéma :

s'il existe I répondant à R alors SORTIR I^+
sinon rechercher I' "équivalent" à I

(I^+ désigne la réponse à R "enrichie" par des informations générales).

Dans la maquette que nous avons réalisée, l'élargissement des réponses est "inclus" dans l'expression des requêtes. (Par exemple, rechercher des hébergements pour 4 personnes était exprimé par une recherche d'hébergements pour 4 à 6 personnes en 1 à 3 étoiles). L'aspect informatif des réponses devait être réalisé en couplant un "système visuel" au système déductif. Il illustrerait les réponses aux demandes d'informations ou de réservation par des images

sur le(s) pays d'accueil et/ou l'hébergement concerné(s).

Il semble évidemment plus intéressant de disposer d'outils plus généraux pour décrire et exploiter cette notion de distance entre les réponses, plutôt que de l'inclure directement dans les requêtes. La disponibilité de ces outils et de mécanismes ad hoc rapprocheraient le fonctionnement du système du fonctionnement d'un système expert.

II.5.- BDGEN (NICOLAS & al)

BDGEN est un prototype de système déductif qui résulte de la superposition d'un modèle déductif à un système relationnel (MRDS).

Il se distingue des systèmes précédents par le fait qu'il utilise les règles de déduction en GENERATION, c'est-à-dire pour rendre explicites les informations déductibles dans le cas d'une opération de mise à jour de la base. (Par exemple, l'ajout d'une personne à la base entraînera l'ajout des couples (petit-fils, grand-père) s'il existe une règle définissant la relation GRAND-PERE).

1 - Structure du système déductif

Il comporte trois modules.

a) Le module de gestion des règles

Il se charge :

- d'ajouter et de supprimer des règles
- de les mettre sous une forme interne puis de les faire mémoriser par le SGBD
- d'activer le processus de recherche d'informations déductibles.

b) Le module d'ajout d'informations

Sa tâche consiste à engendrer les informations déductibles à

partir de celles introduites et ce, en utilisant les règles précédentes.

c) Le module de suppression

Il détermine la présence de l'information à supprimer, gère la suppression des informations qu'elle permet de déduire (ses conséquents) et celle des informations à partir desquelles elle est déductible (ses antécédents).

2 - Les différents niveaux d'utilisation

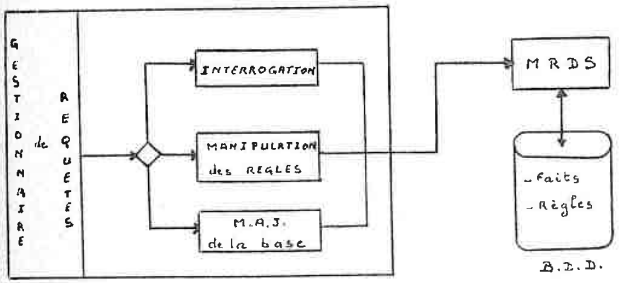
BDGEN permet d'exprimer des requêtes à différents niveaux :

a) sur le schéma de la base (création ou modification de schéma).

b) sur les règles (création, suppression, liste intégrale ou sélective)

c) sur les informations de la base (création, suppression d'une base, ajout et suppression d'informations dans une base).

En outre, pour garantir l'intégrité de la base, toute opération est exécutée sous le contrôle de BDGEN. (L'architecture de BDGEN est schématisée ci-dessous).



3 - Génération et maintenance des informations déduites

a) La génération est un processus qui permet d'engendrer des informations à partir de règles de déduction et d'informations à introduire dans la base. Il est exécuté en cinq phases :

1.- Sélection des règles activables et particularisation (ie instanciation des règles).

2.- Elimination des instances manifestement improductives.

3.- Epuration des instances (élimination des littéraux dont la valuation peut être faite sans accéder à la base).

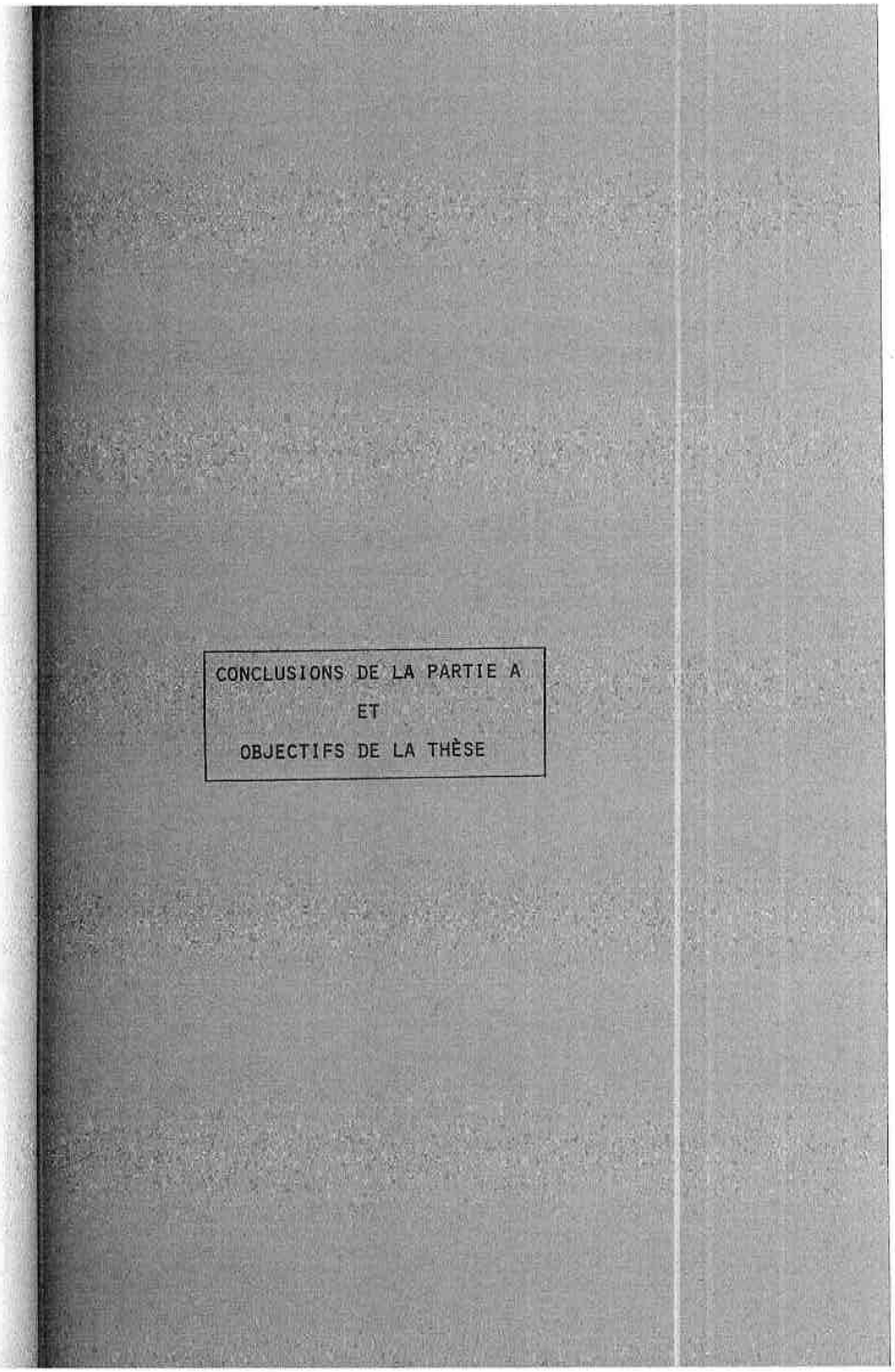
4.- Elimination des instances redondantes (c'est le cas, par exemple, où deux règles R1 et R2 permettent d'engendrer deux ensembles d'informations E1 et E2 respectivement et que l'un est un sous-ensemble de l'autre).

5.- Activation des instances retenues (génération à proprement parler).

b) la maintenance des informations déduites

La suppression d'une information de la base peut avoir des conséquences sur d'autres informations, en l'occurrence sur ses conséquents et ses antécédents. La solution de ce problème doit prendre en compte les diverses natures de définition des informations (intention , extension, mixte). BDGEN opère, face à ce problème, en interagissant avec l'utilisateur.

Le lecteur intéressé trouvera dans [NICO 82] une présentation des règles et de la stratégie utilisées.



CONCLUSIONS DE LA PARTIE A : OBJECTIFS DE LA THESE

Reprenons, en quelques mots, ce qui nous semble être des insuffisances des systèmes "traditionnels" pour poser clairement les objectifs que nous nous sommes assignés dans le cadre de ce travail.

a) Au niveau représentation

Les modèles conceptuels disponibles permettent une représentation abstraite d'un monde observé. Néanmoins, comme nous avons essayé de le montrer, la représentation des phénomènes de ce monde exclusivement par des données rend implicites beaucoup d'aspects de leur sémantique.

D'un autre côté, les seuls concepts des modèles utilisés s'avèrent insuffisants pour exprimer la diversité des phénomènes et de leurs interactions.

b) Au niveau utilisation

Le(s) langage(s) d'utilisation offerts par le SGBD présente(nt), même s'il(s) constitue(nt) une classe de langage "plus évoluée" que celle des langages de programmation classiques, quelques inconvénients déjà cités :

- restriction des opérations aux seules opérations offertes par le langage du SGBD.
- nécessité de "construire ses phrases" (requêtes) au vu de la structure logique de la base (souvenons-nous de l'obligation qu'avait l'utilisateur de se "souvenir" des domaines de jointure et de la liaison sémantique qu'ils matérialisent ou encore des attributs de référence (cf. exemple en SOCRATE) et de ce qu'ils expriment).

L'idée qui ressort est alors d'essayer de contribuer à la résolution des problèmes suivants :

- permettre une "large" description des phénomènes du monde observé.
- permettre l'expression d'opérations spécifiques sur des objets ou des familles d'objets, et ne plus être restreints aux seuls types d'objets permis par le SGBD ; en d'autres termes, chaque base définira son vocabulaire (objets, opérations, contraintes, règles de calcul...).
- mettre à la disposition de l'utilisateur un langage dont le vocabulaire se rapproche de son jargon et dont l'usage ne lui imposerait qu'un apprentissage minimum.
- doter le système de capacités déductives, c'est-à-dire des capacités permettant d'offrir une masse d'informations supérieure à celle que le système connaît "par coeur".

Pour y parvenir, nous avons été amenés à rapprocher des concepts et des techniques de différents domaines (bases de données, BDD déductives, systèmes experts). Un autre domaine, celui des types abstraits de données, apportera aussi sa contribution. Il véhicule des concepts et des mécanismes qui nous aideront dans l'expression des objets d'un monde observé, des opérations qui s'appliquent sur chacun ou quelques uns d'entre eux...

La spécification en termes de types abstraits définira le vocabulaire du domaine observé. La totalité de ce vocabulaire sera alors utilisable dans un langage particulier d'expression des besoins. (Nous nous sommes limités aux besoins en interrogation). Le langage en question et le système qui le prend en compte essaient de plagier un modèle de communication humain :

1 - Une phrase émise par une personne A est une suite de propositions ou certaines, celles supposées connues de l'auditeur B, peuvent être inexprimées.

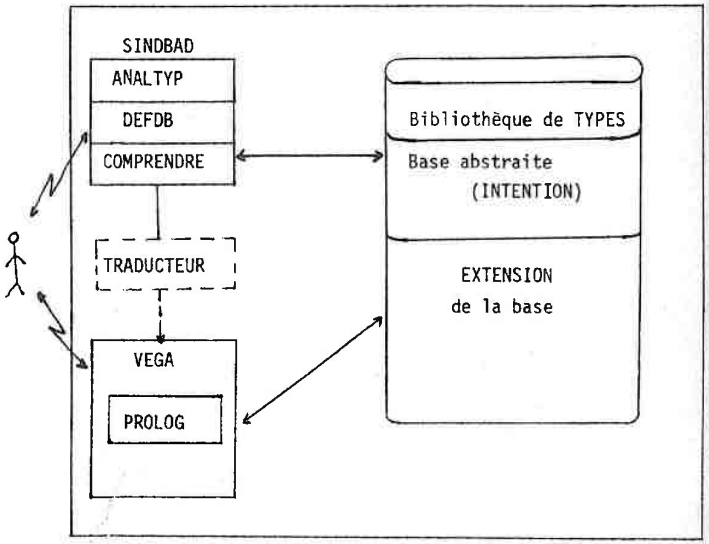
2 - La personne B réagit en deux phases :

a) La première phase est une phase de compréhension. Elle concerne la reconnaissance du vocabulaire utilisé et de la structure globale de la phrase. Elle peut amener à une explicitation mentale des propositions inexprimées et/ou à une demande de propositions complémentaires en provenance de A soit pour lever des ambiguïtés, soit pour parfaire la compréhension.

b) La seconde phase est une réaction comportant un processus d'élaboration de réponse (par "extraction directe" d'informations à partir de la mémoire ou par combinaison, selon certaines règles, des informations mémorisées avec, éventuellement, celles contenues dans les propositions) suivi d'un processus de restitution de la réponse, c'est-à-dire son expression en des termes intelligibles par A.

LE SYSTEME PRESENTÉ DANS CETTE THESE TIENT LE ROLE DE B.

D'un point de vue architecture du système, nous avons dissocié les phases 2-a) (compréhension) et 2-b) (recherche et restitution de solution). La première est l'oeuvre de SINDBAD (Système Interactif de Déduction dans des Bases Abstraites de Données), la seconde est celle de VEGA [CHAB 82]. L'interaction des deux systèmes est symbolisée par le schéma qui suit :



SINDBAD est constitué par trois modules fonctionnant de façon interactive :

- ANALTYP : module d'analyse et de mise en bibliothèque de spécifications de types abstraits de données.
- DEFDB : permet de définir une base de données à partir d'une (ou de plusieurs) bibliothèques de types. Il permet en outre d'étendre ou de restreindre les listes d'opérations initialement définies sur les types.
- COMPRENDRE : est le module central du système. Il a la charge d'analyse et de compréhension des requêtes. Il implémente le schéma de communication que nous venons de décrire.

Dans l'état actuel du système, COMPRENDRE ne réalise que la tâche 2-a) (compréhension), VEGA réalise la tâche 2-b) .

Dans les chapitres qui sont suivre, après avoir défini la notion de types abstraits de données, on s'attachera à montrer :

- comment ils peuvent être utilisés dans le domaine des bases de données et quel peut être leur apport.
- comment VEGA permet la mise en place et l'utilisation de systèmes de données déductifs.
- Enfin, nous nous consacrerons à la présentation du système de compréhension de requêtes en tâchant de mettre en exergue la "philosophie" du langage choisi, son intérêt et les mécanismes mis en jeu pour sa prise en compte par le système.

PARTIE B

ABSTRACTION ET SYSTÈMES DÉDUCTIFS

CHAPITRE I : TYPES ABSTRAITS DE DONNÉES : CONCEPTS ET MÉCANISMES

Dans cette seconde partie, nous allons essayer de présenter, en procédant par analogie avec la spécification de problèmes, les concepts et mécanismes qui permettraient l'élaboration d'un système déductif utilisant les types abstraits.

Nous essaierons de montrer quel peut être l'intérêt et l'apport des mécanismes d'abstraction pour la construction de tels systèmes et comment leur mise en oeuvre permet de doter ceux-ci de capacités déductives.

I - A PROPOS DE LA QUALITE DES LOGICIELS

Les problèmes de qualité et de maintenance des logiciels mettent en relief la nécessité de disposer d'outils formels pour leur spécification. Il fut un temps (qui est toujours en vigueur, d'ailleurs) où face à un problème à résoudre, on s'attelait plus à en décrire une solution sous forme de programme qu'à s'intéresser à sa totale compréhension indépendamment, dans un premier temps, de sa solution technique.

Les difficultés se font ressentir alors à tous les niveaux, et particulièrement

- en phase d'écriture du programme :

Comment prévoir toutes les "entrées légitimes" du programme, c'est-à-dire toutes les données qu'il pourra accepter, et leur prise en compte ?

- en phase de test :

Généralement, le programme est validé sur un échantillon de

données et rarement sur l'ensemble des données qu'il peut admettre en entrée. (On "le teste sur les cas les plus fréquents" et parfois sur des cas particuliers). Comment dans ce contexte de mise au point prévoir une "batterie exhaustive de jeux d'essais" si l'on ne dispose pas d'une caractérisation précise des informations admissibles en entrée ?

- dans la phase d'exploitation du programme produit

Comment lui faire prendre en compte de nouvelles données ?

Comment y incorporer des modifications et surtout contrôler l'impact de ces modifications sur d'autres programmes ?...

L'expérience a montré qu'il est souvent moins coûteux de réécrire un programme que d'y apporter des retouches quelques temps plus tard, surtout si l'on n'en est pas le réalisateur.

Ces constatations ont montré la nécessité de disposer d'outils et de techniques qui, d'une part, aideraient à concevoir et à réaliser des "programmes corrects" et d'autre part, permettraient d'exprimer un problème sous une forme indépendante de sa solution.

Ces techniques diffèrent par leur degré de formalisme.

Certaines procèdent selon le principe qu'il est plus facile de "maîtriser" (comprendre, réaliser, vérifier, modifier) une petite unité de programme qu'une grosse. Des techniques de programmation opèrent de cette façon en préconisant :

- un découpage du problème initial en sous-problèmes
- la réalisation de chacun des sous-problèmes
- l'intégration des solutions des sous-problèmes pour constituer la solution globale.

Les techniques classiques (debugging, batterie de tests...) sont mises en oeuvre lors de la résolution des sous-problèmes et lors de leur intégration. Mais s'il semble raisonnable de dire

que la réduction de la dimension du problème réduit les difficultés pour le résoudre, il n'en demeure pas moins que les expériences menées ont permis de montrer que la phase d'intégration n'était pas exempte de problèmes. Elle exige également un effort de test et de mise au point.

Néanmoins, un des intérêts de ce type d'approche est la notion de modularité introduite ; on peut en effet "voir" la solution d'un sous-problème (module) [PARN 72a,b] comme une boîte noire dont on ne connaît que le comportement. Les autres modules peuvent en connaître les entrées et les sorties associées, sans se préoccuper de la façon dont ces dernières sont élaborées. De ce fait, une modification portant sur l'intérieur de la boîte n'aura pas d'incidence sur le système global. (Les modifications externes, ie concernant les entrées et/ou les sorties, poseront plus de problèmes, mais des expériences [CHAB 79] ont montré que des outils logiciels pouvaient apporter une aide précieuse pour contrôler l'impact d'une modification locale sur un système).

D'autres techniques tentent d'utiliser un haut degré de formalisation. Elles essaient de produire des programmes dont la correction aura été prouvée. Certaines visent à produire automatiquement un programme correct par transformations successives d'énoncé. Ces techniques mettent l'accent sur la nécessité de disposer d'un outil formel d'expression du problème qui pallierait aux inconvénients de son expression informelle (ambiguïté, imprécision, sur-spécification ...) et qui serait un cadre dans lequel des preuves formelles pourraient être menées...

(Une caractérisation plus détaillée et plus précise des méthodes et outils de spécification pourra être trouvée dans [DER & FIN 79], [LISK 75], [CHAB 82]).

L'engouement pour l'utilisation des types abstraits dans des domaines tels que les langages de programmation [WULF 76 & 77, LISK 81, GESC 77, CHAB 79], l'intelligence artificielle ou les

bases de données [BROD 80, DOSC 82, LEAV 81, SMIT 77a,b, TOUN 83, BART 81, CHAB 82, ACM 80...] ne viendrait-il pas alors du fait qu'ils se sont avérés des outils alliant les propriétés des deux familles de techniques que nous avons évoquées ci-dessus ?

Dans ce qui suit, nous allons présenter la notion de types abstraits et leurs techniques de spécification, puis nous nous attacherons, plus particulièrement, à l'examen de leur utilisation dans le domaine des bases de données et des bases de données déductives.

II - TYPES ABSTRAITS : DEFINITION et TECHNIQUES DE SPECIFICATION

Le terme "abstraction" est généralement utilisé en opposition au terme concret. On parle d'abstraction dès que l'on veut ignorer délibérément certains détails. Dans le domaine de l'informatique on parle, par exemple, de machine abstraite sans se préoccuper du hardware qui la concrétiserait, ou d'implémentation abstraite d'une structure ou d'un processus de calcul si on utilise des concepts et des entités mathématiques (ex : théorie des ensembles, logique ...) pour leur expression...

Dans le domaine de la définition des données, parler de type abstrait consiste à considérer celles-ci sans se soucier de leur "représentation informatique". On s'intéresse plus à ce qu'elles sont (ou plutôt ce que sont toutes celles qui se "ressemblent", ie qui sont du même type) et à leur comportement face aux transformations (opérations) qu'elles peuvent subir, qu'à la façon dont elles peuvent être représentées.

II.1.- Quelques définitions informelles

- La spécification d'un type T est la donnée d'un ensemble S de symboles de sortes, d'un ensemble $\mathcal{F}$ de symboles d'opérations (munies de profils) et d'un ensemble E d'axiomes.

(La forme des axiomes permet de distinguer les techniques de spécification - cf. § II.2).

- La représentation d'un type est son expression sous forme d'objets (ou types) plus connus.

On considèrera que tout système dispose d'un ensemble de types connus, dits de base (ex : entier, caractère...). Par ailleurs quand le type spécifié est complexe, il peut être nécessaire de passer par plusieurs représentations successives.

Sur un plan pratique, la représentation d'un type revient à choisir une structure de données appropriée.

- L'implémentation du type est la définition des corps des procédures qui concrétisent les effets des opérations du type sur la structure de donnée qui lui a été associée.

De par ces trois définitions très informelles, on devine aisément la démarche pratique à suivre :

1 - spécifier
2 - réaliser (représenter et implanter).

D'un point de vue formel, il est nécessaire de procéder, à chaque étape, à la vérification de certaines propriétés. Celles-ci concernent :

a) l'adéquation de la spécification au concept dont elle est une abstraction. Une preuve formelle est difficile à établir à ce niveau ; on se contentera d'une "intime conviction". (C'est tout le problème de savoir si le produit d'une phase de conception répond bien aux exigences d'un cahier des charges....).

b) l'adéquation de la représentation vis-à-vis de la spécification. Des preuves formelles sont possibles à ce stade. Elles concernent à la fois l'objet (le type) et ses opérations.

Cette seconde propriété s'exprime en termes de consistance et de complétude de la spécification.

- la consistance d'une spécification est l'assurance que ses axiomes ne peuvent pas mener à des contradictions. D'un point de vue logique cette propriété (dont la preuve formelle peut s'avérer ardue) peut s'exprimer en termes de modèle : un système formel est consistant si et seulement si il admet un modèle. Concrètement, on essaiera d'exhiber un modèle pour prouver la consistance.

D'un point de vue informatique, cette propriété peut s'interpréter comme la capacité de produire une implémentation.

De façon générale, quand il est possible de trouver un modèle mathématique isomorphe à la spécification, la preuve de sa correction peut être établie formellement ; dans le cas contraire, on la validera par le processus, bien connu en informatique, de programmation et tests ([ADJ 78], [DERFI 79]).

- Notion de complétude

D'un point de vue logique, un système formel est dit complet si toute formule y est soit un théorème, soit la négation d'un théorème. En terme d'interprétation, cette propriété exprime l'unicité du modèle : un système formel est complet s'il admet un modèle unique.

Dans le contexte des types abstraits cela voudrait dire que le type ne peut être implémenté que d'une seule façon.

Comme en ce qui concerne la consistance, cette propriété est difficile à établir. Par ailleurs, elle est en définitive non souhaitable parce que trop contraignante.

GUTTAG & HORNING [dans GUT 78] définissent une notion plus faible (la complétude suffisante) et des critères quasi-syntaxiques pour sa preuve et pour la définition d'une spécification la vérifiant.

Remarque :

Nous avons délibérément opté pour des définitions informelles. Le lecteur intéressé par les aspects formels du sujet pourra consulter [DER & FIN 79], [GUT 78], [ADJ 78].

II.2.- Les techniques de spécification

La spécification d'un type abstrait comporte généralement deux parties :

- l'une syntaxique, où l'on définit les opérations et leur profil.
- l'autre sémantique où l'on caractérise les propriétés des opérations (et éventuellement des contraintes que doivent vérifier les occurrences d'objets du type ; ces contraintes sont appelées INVARIANTS).

Les techniques de spécification se distinguent par la "forme" d'expression de la seconde partie.

II.2.1.- La spécification axiomatique (ou pré/post)

Cette technique consiste à spécifier un type T en s'appuyant sur un type T' "plus connu". T' est parfois appelé modèle de T. Les opérations de T sont, dans cette technique, exprimées en termes de pré et de post conditions, c'est-à-dire que l'on exprime des propriétés que doit vérifier le type (ou d'autres éléments tels ses paramètres cf. § II.3...) préalablement à l'application d'une opération, sous forme d'une pré-condition et de la même façon, on caractérise le résultat de l'opération par une post-condition. On s'attache en quelque sorte à caractériser les "états avant et après" sans exprimer la façon dont s'effectue la transition de l'un à l'autre.

Exemple : (inspiré de DER & FIN 79)

Spécification d'un type file d'entiers en termes de suite d'entiers

Nous supposons disposer d'un type suite et de ses opérations :

```
estsuite :        → suite
svide :     suite → bool   % permet de savoir si une suite ne %
                             contient aucun élément
ajts : suite×entier → suite % adjonction d'un élément à la
                              suite %
enleves : suite → suite % retirer un élément %
premier : suite → entier % prendre le premier élément %
longueur : suite → entier % nombre d'éléments dans la suite %
```

Si nous notons une suite s à n éléments sous la forme <e1.e2.e3.....en>, l'effet de ses opérations peut être décrit ainsi :

```
ajts(s,e) = <s.e> (ie <e1.e2. ... .en.e>)
enleves(s) = <e2.e3.....en>
premier(s) = e1
longueur(s) = n
```

On suppose par ailleurs qu'il existe une suite vide notée <> avec les propriétés : longueur(<>) = 0
svide(<>) = VRAI

On peut alors donner une spécification du type file en termes de suite sous la forme suivante :

```
type file (ℓ:entier)        % ℓ désigne la taille maximum de la
                                file %
invariant ℓ > 0∧estsuite(file) ∧0 < longueur(file) < ℓ
initialisation
    initfile = <>
```


opérations

```
videf : file → bool
têtef : file → entier
enlevef : file → file
ajtf : file×entier → file
```

sémantique

soit f,f' : file,e : entier,b:bool

- videf(f) = b : post b = svide(f)
- têtef(f) = e : pré 0 < longueur(f) ≤ ℓ ; post e = premier(f)
- enlevef(f) = f' : pré 0 < longueur(f) ≤ ℓ ; post f' = enleves(f)
- ajtf(f,e) = f' : pré 0 ≤ longueur(f) < ℓ ; post f' = ajts(f,e)

ftype

Remarque :

La représentation d'un type spécifié axiomatiquement consistera en son expression en des objets "concrets" (ou objets informatiques). L'implémentation consistera quant à elle à définir les corps des procédures implémentant les opérations ainsi que leurs conditions d'entrée et de sortie.

(Formellement, on devrait prouver la correction de la représentation et de l'implémentation (cf. [DER & FIN 79])).

II.2.2.- La spécification algébrique

Un type abstrait spécifié algébriquement est défini par la donnée :

- d'un ensemble S de noms de domaines ou sortes dans lequel on distinguera une sorte dite d'intérêt (celle du type que l'on veut définir).

- d'un ensemble $\mathcal{F}$ de noms d'opérations ; à chaque opération est associé son profil (liste de domaines et co-domaines), le type d'intérêt devant figurer dans chacun des profils.
- d'un ensemble $\mathcal{E}$ d'axiomes égalitaires exprimant des relations entre des opérations. Ces axiomes sont de la forme

  DROITE(X) = GAUCHE(X) où X désigne une liste de variables typées (implicitement quantifiées universellement), DROITE et GAUCHE sont des compositions de symboles d'opérations. GAUCHE peut également être une expression conditionnelle (SI...ALORS ...SINON...).

(De façon générale, on considèrera que le type Bool est prédéfini pour tout type T et que l'on dispose, également pour tout T, d'une opération si-alors-sinon : Bool×T×T → T vérifiant les axiomes :

- si-alors-sinon(vrai,x,y) = x
- si-alors-sinon(faux,x,y) = y)

Exemple : Spécification algébrique d'un type ensemble d'entiers

type set
sortes set,bool,entier
opérations
  vide : → set
  ajouter : set×entier → set
  retirer : set×entier → set
  estdans : set×entier → bool
axiomes
  soit s : set ; e,e' : entier
  • estdans(vide(),e) = faux
  • estdans(ajouter(s,e),e') = si egal(e,e')alors vrai
    sinon estdans(s,e')

  • retirer(vide(),e) = vide()
  • retirer(ajouter(s,e),e') = si egal(e,e')alors retirer(s,e')
    sinon ajouter(retirer(s,e'),e)

ftype

L'intérêt d'une telle forme de spécification est de se placer dans un cadre formel qui est celui de l'algèbre universelle et d'y développer donc des raisonnements formels pour établir les propriétés des spécifications (Pour plus de détails voir [GAUD 80, chap. II], [GUT 78], [ADJ 78]).

II.2.3.- Remarque : types abstraits avec erreurs

La spécification des cas d'erreurs est une partie "délicate" de la définition d'un type abstrait. Pour ne pas faillir à l'exigence de rigueur, il est nécessaire d'exprimer ces erreurs de façon aussi formelle et précise que le reste de la spécification.

Les problèmes dont on doit tenir compte dans ce cas concernent à la fois la reconnaissance des cas d'erreurs (par exemple, accès au sommet d'une pile vide, retrait d'un élément d'une file vide...), la caractérisation du résultat de l'opération appliquée, et la définition des effets d'autres opérations sur des résultats-erreur (propagation des erreurs)...

Il existe deux voies possibles pour l'expression de spécifications avec erreur :

a) spécification par des restrictions

Elle consiste à exprimer

- des préconditions à l'application des axiomes

  ex : pré(enlevef(file)) = ¬videf(file) : on ne peut pas retirer un élément à une file qui est vide.

- des cas d'échec (qui précisent les conditions de fin anormale des procédures qui implémenteront les opérations concernées).

ex : si Longueur(file)≥ taille max alors ECHEC(ajouter(f,e))

La procédure qui implémentera ajouter un élément à une file avortera si la file a atteint sa taille maximum.

b) Les algèbres d'erreurs

Dans cette approche, on définit, de façon analogue à la spécification sans erreur, des opérations-erreurs et des axiomes-erreurs.L'intérêt d'une telle formulation est de ne pas particulariser la forme d'expression des erreurs et, par conséquent, de rester dans un cadre algébrique.

ex : ERR-OPNS :
inexistant : → entier
ERR-AXIOME : tetef(filevide) = inexistant

II.2.4.- Mécanismes de construction de types

Nous avons déjà évoqué la notion de types de base, c'est-à-dire des types que l'on considère comme connus et que l'on peut utiliser. (Dans l'exemple du § II.2.2. concernant le type file, nous avons considéré le type entier (et ses opérations <, ≤,>,≥) comme étant connu).

Dans beaucoup de cas, il peut être agréable de pouvoir utiliser des spécifications de types existants, en tant que telles ou pour élaborer de nouvelles spécifications. (Dans cette optique des outils automatisés de gestion de bibliothèques de types (adjonction de spécification, modification, retrait ou encore recherche de spécifications équivalentes [PROC 82]) seraient d'un apport certain).

Dans ce paragraphe, nous allons rapidement passer en revue des mécanismes de construction de types à partir de types existants.

a) L'enrichissement (extension)

On peut être amené à définir, à partir d'un type T donné, un type T' "plus large", en ajoutant à T de nouveaux symboles de sortes et/ou de nouvelles opérations (et axiomes associés).

C'est ainsi, que l'on peut, par exemple,

- définir des relations permettant de "comparer" des objets d'un même type T

(profil : T×T → bool)

- ou spécifier des "opérateurs dérivés", c'est-à-dire définir de nouvelles opérations à partir de celles existantes.

ex : si on dispose, à l'origine, des opérations ∧,¬ sur un type bool, on peut définir l'implication (→) et la disjonction (∨) en termes de ∧ et ¬ :

$A \vee B = \neg(\neg A \wedge \neg B)$
$A \rightarrow B = \neg A \vee B$

b) La restriction

C'est un processus inverse de celui d'enrichissement. Il vise à "restreindre" le type en y adjoignant des axiomes "plus contraignants" ou encore en limitant sa liste d'opérations.

ex : 1 - On peut définir un type ensemble comme étant une restriction d'un type multi-ensemble (ensemble où les éléments peuvent être "répétés") en y formulant des axiomes interdisant la présence multiple d'un même élément au sein de l'ensemble.

2 - Sur un type bool, comportant ∧,∨,→ et ¬, on peut restreindre ses opérations à ∧ et ¬ uniquement.

c) Le produit cartésien ou "tupling" (noté < , >)

Etant donné n types $t_1,\ldots,t_n$, le tupling consiste à

élaborer un type T constitué par le produit cartésien $t_1 \times t_2 \times \ldots \times t_n$.

Exemple : type date = <tjour,tmois,tannée>
jour : date → tjour
mois : date → tmois
année : date → tannée
⋮
ftype

Outre les $i^{èmes}$ projections, on pourra également définir d'autres opérations sur T en utilisant tout ou partie des opérations des t_i. (Par exemple, on pourrait ainsi définir la notion d'égalité de dates en termes de conjonction d'égalités sur les jours, les mois et les années...).

On dira que le type T IMPORTE (tout ou partie) des opérations des t_i. On notera que le processus d'importation s'applique aussi aux sortes des t_i (on n'importera que les seules sortes nécessaires à la spécification de T).

d) La paramétrisation

C'est un procédé permettant de spécifier un type T ayant des types $P_1, P_2, \ldots, P_n$ comme paramètres formels (au même sens que les paramètres formels d'une procédure dans un langage de programmation). T pourra alors être utilisé par substitution de paramètres effectifs aux paramètres formels.

Exemple : Spécification du type ensemble de "quelque chose"

type SET (qqchose)
opérations
vide : → SET
insérer : SET×qqchose → SET
retirer : SET×qqchose → SET
estdans : SET×qqchose → bool

axiomes
% Les axiomes sont identiques à ceux du § II.2.2 %
% L'exemple du § II.2.2 est une instanciation du %
% type SET(qqchose) ayant ENTIER comme paramètre effectif %
ftype

Cette technique pose certains problèmes concernant la validité des paramètres effectifs. En effet, les paramètres formels doivent généralement vérifier certaines conditions. On devra alors se doter de moyens de s'assurer qu'elles sont remplies par les paramètres effectifs. Ces conditions peuvent porter sur la nécessité de disposer de certaines opérations sur le paramètre (c'est le cas de l'égalité sur le type SET - cf. axiomes du § II.2.2).

D'autres problèmes peuvent se poser, comme le "degré" de paramétrisation : doit-on ou pas admettre des paramètres effectifs qui soient des instanciations de types paramétrés et, surtout, sera-t-on capable, alors, de comprendre et d'expliciter le comportement d'objets du type set(tableau(liste(...))) où set, tableau et liste seraient des types paramétrés ? (Se référer pour plus de détails à [ADJ 78], [EHR 81], [TERR 83]).

Remarques

1 - Concrètement, on pourra "combiner" ces divers mécanismes pour spécifier un type T.

2 - D'un point de vue utilisation, on peut ne rendre "visibles" que les seules opérations sur T (ie son comportement externe) assurant ainsi la protection du/des types qui ont contribué à sa spécification.

II.3.- Conclusion

Les concepts et mécanismes que nous venons de présenter de façon très brève et très informelle ont donné naissance à des

logiciels (dont SIMULA et PASCAL sont les "jeunes" ancêtres) de programmation fortement typés ([WULF 76 & 77], [LISK 81], [CHAB & al 79], [GESC 77]) qui permettent une conception et une implantation modulaires des applications.

D'autres systèmes ([MUSS 80], [LESC 83], [BARB 82]) permettent d'établir des propriétés de correction de spécification. Ils utilisent généralement des techniques de réécriture ([HUET 80], [JOUA 83]).

Nous présentons un de ces systèmes dans le paragraphe qui suit. Il s'agit en l'occurrence de VEGA [CHAB 82] que nous avons eu à utiliser pour valider quelques idées concernant la mise en place de systèmes de données déductifs basés sur les types abstraits.

Enfin, ORSEC [PROC 82] est, à notre connaissance, le seul système existant d'aide à la recherche de spécifications équivalentes dans une bibliothèque de spécifications. Il permet ainsi une économie notoire d'effort de spécification.

III - VEGA : SYSTEME DE SPECIFICATION ET DE VALIDATION DE TYPES ABSTRAITS

VEGA est un système expérimental d'aide à la spécification et à la validation de types abstraits de données et de programmes. Il offre un langage de spécification de types (avec erreurs) et de programmes. En ce sens, il est un langage de programmation fonctionnel [BACK 78] fortement typé.

VEGA bénéficie, en outre, de la puissance de PROLOG en matière d'inférence, puisqu'il est écrit en ce langage. Ceci fait de lui un système de programmation logique supportant des spécifications algébriques de types abstraits de données et de programmes.

III.1.- L'environnement initial de spécification

VEGA offre un environnement initial de spécification constitué par un ensemble de spécifications de types et de constructeurs de bases que l'on pourra utiliser pour représenter ou spécifier de nouveaux types.

Cet environnement contient les spécifications de :

- BOOL (Booléen)
- CHAR (caractère)
- STRING (chaîne de caractères avec des opérations de concaténation, d'extraction de sous-chaînes, de détermination de la longueur d'une chaîne...)
- INT (entiers)
- TUPLE (produit cartésien labellé)
- SET (ensemble paramétré)
- SEQ (séquence paramétrée)

Nous ne donnons pas ici la spécification de ces divers types. Elles sont explicitées dans CHAB 82 - Chapitre IX. Néanmoins, nous y ferons souvent référence dans les exemples que nous présenterons dans les paragraphes qui suivent.

III.2.- Langage et mécanismes de spécification

III.2.1.- Langage de spécification

VEGA supporte des spécifications de type

- pré/post avec un cadre formel algébrique (dite technique "modèle abstrait")
- algébrique où les propriétés de l'objet spécifié (type ou programme) sont exprimées par des axiomes équationnels.

Une spécification d'un type, en VEGA, est une unité appelée

SYSTEM dont le corps prend une des deux formes suivantes, selon que le type est spécifié par la technique "modèle abstrait" ou celle dite algébrique.

a) Syntaxe d'une spécification algébrique

SYSTEM (<nom du systeme>)/
SORTS (<noms des sortes>)/
IMPORTS (<noms des sortes externes>)/
BUILD/
<liste des constructeurs et de leur profil>
OKOPNS/
<liste des OK-opérations et de leur profil>
EROPNS/
<liste des opérations-erreur>
OKAXS/
<OK-axiomes>
ERAXS/
<axiomes-erreur>
END

Exemples : (1) Spécification du type pile d'entiers

SYSTEM PILINT/
SORTS(PILINT)/IMPORTS(INT,BOOL)/
BUILD/
PILINT(VIDE → PILINT)/
PILINT(EMPILER(PILINT,INT) → PILINT)/
OKOPNS/
PILINT(DEPILER(PILINT) → PILINT)/
PILINT(SOMMET(PILINT) → INT)/
PILINT(ESTVIDE(PILINT) → BOOL)/
EROPNS/
PILINT(SOMERR → INT)/
PILINT(PILERR → PILINT)/

OKAXS/
PILINT(DEPILER(EMPILER(*P,*I)) = *P)/
INT(SOMMET(EMPILER(*P,*I)) = *I)/
BOOL(ESTVIDE(VIDE) = TRUE)/
BOOL(ESTVIDE(EMPILER(*P,*I)) = FALSE)/
ERAXS/
PILINT(DEPILER(VIDE)) = PILERR/
INT(SOMMET(VIDE)) = SOMERR/
END(PILINT)/

Remarque concernant le paragraphe BUILD et la notion de constructeur

On définit dans ce paragraphe les opérateurs qui permettent de "fabriquer" un objet du type. L'idée intuitive qui est sous-jacente est que tout objet d'un type donné évolue à partir d'un état initial (dans le cas de PILINT, l'état initial est la PILEVIDE) sous l'effet de certaines opérations d'observation de l'objet (encore appelées simplificateurs [DER & FIN 79]) et d'autres opérateurs "de modification" appelées constructeurs.

C'est cette idée qui est à la base de la définition de la notion de complétude suffisante [GUT 78] où l'on partitionne l'ensemble des opérations en

- un ensemble d'opérations à codomaine dans le type d'intérêt (les constructeurs)
- et un ensemble d'opérations à codomaine non dans le type.

Ceci permet d'établir des conditions suffisantes de complétude suffisante et une forme canonique des axiomes (cf. [GUT 78]) de la spécification.

Exemple (2) Spécification du (programme) PGCD de deux entiers

```
SYSTEM (PGCD)/
   IMPORTS(INT;BOOL)/
OKOPNS/
   PGCD(PGCD(INT,INT) ← INT)
EROPNS/
   PGCD(ARGNEG ← INT)/
OKAXS/
   INT(PGCD(*I,*I) = *I)/
   INT(PGCD(*I,0) = *I)/
   INT(PGCD(0,*I) = *I)/
   INT(PGCD(*I,*J) = IT(INT(*J<*I)INT(*J>0),PGCD(*I-*J,*J)))/
   INT(PGCD(*I,*J) = IT(INT(*I<*J)INT(*I>0),PGCD(*I,*J-*I)))/
ERAXS/
   INT(PGCD(*I,*J) = IT(INT(*I<0)OR INT(*J<0),ARGNEG))/
END(PGCD)/
```

Quelques commentaires s'imposent.

Cette spécification est en fait une axiomatisation, avec erreur (si un des arguments est négatif), du PGCD de deux nombres.

Il aurait peut être été plus clair de dissocier le terme PGCD de deux entiers de l'opération PGCD définie dans le paragraphe OKOPNS. Les axiomes du paragraphe OKAXS sont une autre forme d'expression (typée) des axiomes qui définissent le PGCD de a et b, a et b étant des entiers :

- PGCD(a,0) = a
- PGCD(a,a) = a
- PGCD(0,b) = b
- PGCD(a,b) = si 0<b<a alors PGCD ((a-b),b)
- PGCD(a,b) = si 0<a<b alors PGCD (a,b-a)

Par ailleurs, l'ERAXS (axiome-erreur) exprime le fait qu'une erreur ARGNEG se produit si l'un des deux termes est négatif.

On notera enfin, que VEGA permet de spécifier donc dans un formalisme unique, à la fois des types de données et des programmes.

b) Syntaxe d'une spécification par la technigue "modèle abstrait"

```
SYSTEM (<nom de la spécification>)/
   SORTS (<nom-de la sorte définie>)/
   MODEL (<nom-du modèle abstrait>)/
   OKOPNS/
      :
   EROPNS/
   AFUNC/
        % liste des axiomes définissant la fonction d'abs-
          traction %
   OKSEMS/
        % liste des OK-axiomes %
   ERSEMS/
        % liste des axiomes-erreur %
END (<nom de la spécification>)/
```

Cette technique consiste à spécifier un type T en termes d'un type T' (cf. § II.2.1) faisant office de type support de la représentation de T. De façon générale, donner une représentation d'un type abstrait consiste, d'une part à choisir le support de la représentation (dans le langage de VEGA, il est introduit au paragraphe MODEL) et, d'autre part définir une fonction allant soit des objets de la représentation vers œux du type abstrait (fonction d'abstraction), soit des objets du type abstrait vers

ceux de la représentation (fonction de représentation). On s'attachera, à prouver que tout objet du type source est représenté et que les axiomes sont "conservés" sur le type représentant. Dans VEGA, cette fonction est définie au sein du paragraphe AFUNC.

[Quant aux paragraphes que nous n'avons pas évoqués ici, ils ont la même utilité que leurs homologues dans la technique de spécification précédente].

Exemple : Spécification d'une pile d'entiers en termes de séquence d'entiers

Sur le type séquence d'entiers (SEQ(INT)), nous supposons définies les opérations :

```
ADDGCHE : SEQ INT → SEQ   %adjonction d'un entier "à gauche"%
DELGCHE : SEQ     → SEQ   %suppression de l'entier à gauche%
ELEMENT : SEQ     → INT   %accès à l'entier à gauche%
VIDESEQ : SEQ     → BOOL  %permet de savoir si une séquence
                            est vide%
```

La spécification du type pile d'entiers, avec comme modèle SEQ(INT), est alors :

```
SYSTEM (SPINT)/
   SORTS (SPINT)/MODEL(SEQ(INT))/
OKOPNS
   SPINT(VIDE ← SPINT)/  %définition d'une pile vide%
   SPINT(EMPILER(SPINT,INT) ← SPINT)/ %empiler un élément%
   SPINT(DEPILER(SPINT) ← SPINT)/     %dépiler%
   SPINT(SOMMET(SPINT)  ←INT)/   %examiner le sommet%
   SPINT(ESTVIDE(SPINT) ← BOOL)/ %est-ce que la pile est vide%
EROPNS
   SPINT(SOMERR ← INT)/
   SPINT(PILERR ← SPINT)/
```



```
AFUNC
   (1) SPINT(REP(NIL)=VIDE)/
   (2) SPINT(REP(*I.*S)=EMPILER(REP(*S),*I))/

   %Ces deux axiomes définissent la fonction d'abstraction REP
    du type représentant  vers celui représenté.
    L'axiome (1) indique que la séquence vide ou NIL représente
    la pile VIDE.
    L'axiome (2) indique que la représentation de l'ajout à gau-
    che d'un entier I à une séquence S est l'empilement de  I
    dans la pile que représente S.%

OKSEMS
   SPINT(VIDE=REP(NIL))/
   SPINT(EMPILER(REP(*P),*I)=REP(ADDGCHE(*P,*I)))/
   SPINT(DEPILER(REP(*P))=REP(DELGCHE(*P)))/
   INT(SOMMET(REP(*P))=ELEMENT(*P))/
   BOOL(ESTVIDE(REP(*P))=VIDESEQ(*P))/

   %Ce paragraphe explicite la représentation des opérations :
    - EMPILER est associé à ADDGCHE
    - DEPILER à DELGCHE
    - SOMMET à ELEMENT
    - et ESTVIDE à VIDESEQ
    Le paragraphe suivant s'intéresse aux termes-erreurs%

ERSEMS
   SPINT(DEPILER(VIDE)=PILERR)/    %on obtient PILERR ou SOMERR
   INT(SOMMET(VIDE=SOMMERR)/        si on effectue respectivement
END(SPINT)                          une tentative de DEPILER sur
                                 une pile VIDE ou un accès au sommet
                                 d'une pile VIDE%
```

III.2.2.- Les mécanismes de construction de types

VEGA supporte la plupart des mécanismes de spécification de types que nous avons définis au § II.2.4.

a) Définition de sous-types

Deux clauses (RESTRICTO et INVARIANT) permettent de construire des sous-types d'un type donné en apposant des contraintes que doivent vérifier les occurrences du type en question. Ces contraintes sont exprimées sur le modèle abstrait.

α) RESTRICTO permet d'expliciter le domaine du type (intervalle ou liste de valeurs)

Exemple :

```
(1) SYSTEM(DIZAINE)/SORTS(DIZAINE)/
      MODEL(INT RESTRICTO(*I:10.20.30.40.50))/
        :
      %DIZAINE est un "type énuméré" prenant ses valeurs
       dans 10,20,30,40,50%

(2) SYSTEM(ENSENT)/SORTS(ENSENT)/
      MODEL(SET(INT)RESTRICTO(*I:10 TO 50))/
      %ENSENT est l'ensemble des entiers I, I ∈ [10,50] %
```

β) INVARIANT permet d'exprimer des propriétés "constantes"

Exemple : Définition d'un ensemble borné : son nombre d'éléments ne doit jamais excéder 100, par exemple.

```
SYSTEM(ENS-100)/SORTS(ENS-100)/
  MODEL(SET(INT)INVARIANT I-100)/
  INVAR(I-100(*E) ← ENS-100(*E:INT(ENS-100(CARD(*E))<100,
                                   ERRCARD))/
%Le cardinal de l'ensemble (CARD est un opérateur défini sur
 %le type SET, qui rappelons-le, fait partie, comme INT d'ailleurs, de l'environnement initial) doit constamment être
 inférieur à 100 sinon une erreur ERRCARD se produit.%
```


b) Restriction avec renommage

Les types de l'environnement initial ont été délibérément surspécifiés. Leur utilisation en tant que modèle abstrait amène souvent à leur restriction aux seules opérations représentant celles du type source.

L'exemple du § III.2.1.b l'illustre bien : Les opérations du type SEQ(INT) (représentant d'une pile d'entiers) sont réduites strictement à celles nécessitées par la spécification de la pile d'entiers, à savoir : ADDGCHE, DELGCHE, ELEMENT et VIDESEQ.

c) Extension avec coercition et surcharge des opérateurs

- La surcharge d'opérateurs est l'utilisation de symboles d'opérations identiques dans des spécifications de types différents.

Exemple : On peut définir l'opérateur '+' sur les entiers et la séquence (dans le second cas, il désigne la concaténation).

SYSTEM INT	SYSTEM SEQ
+ : INT×INT → INT	+ : SEQ×SEQ → SEQ

- La coercition, quant à elle, consiste à "forcer" une sorte S1 à être une sous-sorte d'une sorte S2.

Exemple : La définition d'une pile avec lecture interne (ie avec accès à l'un quelconque de ses éléments, et non uniquement à son sommet) peut se faire en termes de PILE. On étendra la liste des opérations avec des opérateurs permettant - l'accès à un rang dans la pile spécifié, par un index

- la progession, "vers le haut" ou "vers le bas", de l'index.

d) Fusion de spécifications

Ce procédé consiste à regrouper au sein d'un même SYSTEM de VEGA les spécifications de deux ou plusieurs types. La fusion peut se justifier par le fait que des relations (opérateurs identiques, par exemple) existent entre des sortes des types concernés ou, tout simplement, parce que l'utilisateur désire les "encapsuler" (les mettre dans la même "boîte") ensemble.

On trouvera dans [CHAB 82] un exemple où sont fusionnées les spécifications d'une pile et d'un tableau.

On verra, par la suite, que ce mécanisme peut s'avérer fort utile pour construire des abstractions en bases de données.

III.2.3.- Les instructions de traitement de VEGA

VEGA offre des possibilités de "programmation" en mettant à la disposition de l'utilisateur un certain nombre d'instructions pour

- évaluer une expression (RUN)
- exécuter une suite d'instructions (MRUN)
- sortir des résultats ou données intermédiaires (PRINT) avec possibilité de saut de ligne à l'édition (LINE)
- parcourir tous les éléments d'une collection (EACH)
- retrouver le premier objet d'une collection vérifiant certaines propriétés (FIRST).

Exemple

(1) RUN(PILINT(DEPILER(EMPILER(EMPILER(PVIDE,1),2)))) ? aura pour effet de produire PILINT:EMPILER(PVIDE,1) où PILINT est la sorte du terme résultat EMPILER(PVIDE,1)

(2) MRUN(FIRST(*H IN ENSHEB(HEB):STRING(HEBERG(CATEG(*H)='2*')))? permet d'obtenir le premier Hebergement (*H), de l'ensemble HEB (de type ENSHEB) d'hébergements, de catégorie (CATEG) égale à 2*.

Nous terminons ainsi la présentation des généralités sur les types abstraits et sur le système VEGA.

Nous avons essayé de montrer très brièvement que ce dernier apportait une aide à la spécification de types par son langage, ses mécanismes et son environnement initial. Il est de plus un langage de programmation permettant de valider des spécifications (par des tests) ou de programmer des applications.

Dans [CHAB 82], un exemple d'application Bases de données est développé. Dans ce qui va suivre, nous allons essayer de montrer comment doter des systèmes de types abstraits de capacités déductives, et plus particulièrement, comment VEGA peut aider à les implémenter.

CHAPITRE II : SYSTÈMES DÉDUCTIFS ET TYPES ABSTRAITS : DE L'INTÉRÊT DES TYPES ABSTRAITS DANS LES SYSTÈMES DE DONNÉES DÉDUCTIFS

I - INTRODUCTION

Un net regain d'intérêt se fait sentir pour la formalisation des bases de données. Celle-ci peut être appréhendée de trois façons :

a) l'approche "traditionnelle" où l'on essaie de spécifier un schéma de la base qui soit la spécification d'un modèle abstrait d'un monde observé.

En effet, on est tenté d'établir une analogie entre les niveaux de représentation dans la conception d'une base de données et les concepts d'abstraction, de représentation et d'implémentation de types abstraits. Dans cette optique, la phase de conceptualisation, qui vise à produire un schéma conceptuel exprimé en termes du modèle conceptuel choisi, peut être vue comme une abstraction du monde observé. Les niveaux logique et physique sont eux, en rapport avec la représentation et l'implémentation.

Mais, l'établissement d'analogies ne suffit pas pour permettre de dégager des concepts et des mécanismes pour spécifier une base de données en termes de types, comme on spécifierait des données "aussi simples" qu'une pile ou une file...

b) l'approche axiomatique permet de spécifier une base en termes de la logique, sans se rattacher à un modèle particulier. Les modèles abstraits utilisés dans l'approche traditionnelle sont utilisés alors pour produire une implantation concrète. ([ONER 82], [DAHL 82], [GALL 78]).

c) l'approche algébrique ([MAIB], [TOMP 80], [DOSC 82], [DERN 84]), qui mêle logique et algèbre universelle, pourrait être utilisée. Elle apporterait sa rigueur pour permettre de définir parfaitement la sémantique des modèles de représentation de données utilisées dans l'approche traditionnelle. Les types ainsi spécifiés pourraient alors servir pour la spécification de schémas.

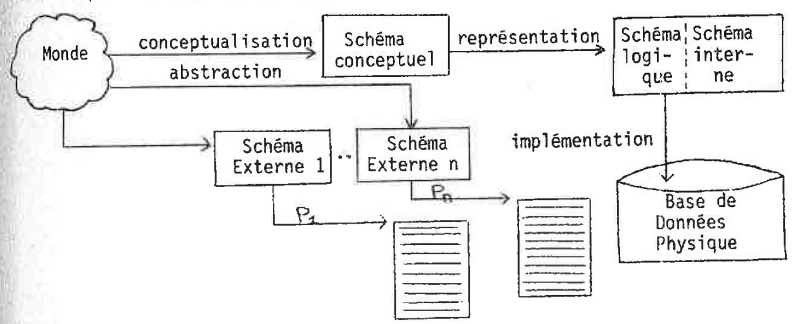
Comme nous aurons l'occasion de le voir, cela pourra nécessiter la mise en oeuvre des mécanismes de construction de types que nous avons présentés auparavant.

Dans les parties qui vont suivre, nous allons successivement
- revenir brièvement sur le problème de la mise en place d'une base de données, puis,
- montrer ce que, d'un point de vue pratique, l'on peut espérer de la mise en oeuvre de certains mécanismes d'abstraction.

Nous nous étendrons plus longuement dans la troisième partie de la thèse, sur les concepts et les mécanismes que nous proposons d'utiliser pour élaborer un système de données déductif basé sur les types abstraits.

II - SPECIFICATION D'UN (META-)SYSTEME DE DONNEES

Le processus de conception et de mise en place d'une base de données peut se schématiser ainsi :



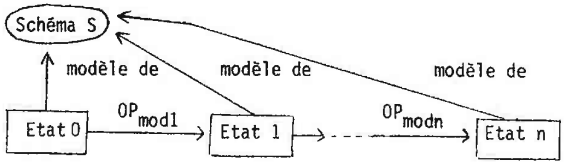
La conceptualisation vise à produire un modèle abstrait du monde observé. Interprété en termes de logique, on peut dire que l'on vise à constituer un système formel S pour lequel un modèle est fourni par la base de données physique.

Les schémas externes sont des moyens de définition, à l'intention d'un utilisateur ou d'une classe d'utilisateurs, des seuls éléments du schéma global qui les concernent. Des procédures "implémentent" les schémas externes, c'est-à-dire permettent d'extraire ou de mettre dans la base les données impliquées par la/les applications associées à chacun des schémas externes.

L'adéquation du modèle (qu'est la base de données physique ou BDP) au système formel S (qu'est son schéma) doit être préservée durant toute la vie de la base. Cette notion de vie de la base impose de considérer un point de vue dynamique que nous allons expliciter en termes d'états et de transformation d'états.

Une BDP, à un instant donné, est le reflet d'un monde qu'elle représente. Sur cette BDP peuvent s'appliquer deux grandes familles d'opérations :

- celle des opérations qui permettent d'observer tout ou partie de la BDP à instant donné. (Nous désignerons de telles opérations par OP_{obs}).
- celle des OP_{mod}, constituée par les opérations qui modifient la BDP, c'est-à-dire qui produisent une BDP' (qui soit toujours un modèle du système formel défini par le schéma). Ces OP_{mod} réalisent donc des transformations d'états, comme symbolisé ci-dessous :



Essayons de formaliser ces notions en termes algébriques.
Pour cela, nous considèrerons trois (méta-) sortes :

- une sorte ETAT sur laquelle s'appliqueront les opérations d'interrogation et de modification.
- une sorte SORTIE correspondant aux "productions" des opérations d'observation (interrogation).
- une sorte MODIF pour les modifications.

a) Sorte ETAT

apsor : ETAT×SORTIE → SORTIE
apmod : ETAT×MODIF → ETAT
eval : ETAT×EXPR → Bool

Ces opérations correspondent, respectivement, à l'application d'une OP_{obs}, d'une OP_{mod} ou a une évaluation d'une expression EXPR dans un état donné de la base.

b) Sorte SORTIE

outvide : → sorvide
interro : EXPR×SORTIE → SORTIE

Cette (méta-)sorte permet de désigner les résultats des opérations d'interrogation. EXPR s'interprète comme une expression caractérisant la SORTIE à effectuer.

c) Sorte MODIF

modifvide : → modifvide
modifier : EXPR×MODIF → MODIF

La (méta-) sorte MODIF quant à elle désigne les modifications causées par une OP_{mod} (adjonction de nouveaux éléments au sein d'un état, suppression ou modification d'éléments existants).

EXPR s'interprète, dans ce cas, comme l'expression de pré-conditions sur l'OP_{mod} en question.

Les axiomes ci-dessous précisent la sémantique du méta-système définissable à l'aide de ces trois sortes :

Soit e : ETAT, s : SORTIE, m : MODIF, b : EXPR

(1) apsor(e,interro(b,s))= si eval(e,b) alors apsor(e,s)
sinon apsor(e,outvide())

(2) apsor(e,outvide()) = sorvide

(3) apmod(e,modifier(b,m)) = si eval(e,b) alors apmod(e,m)
sinon apmod(e,modifvide())

(4) apmod(e,modifvide()) = e

La conception d'une base de données et de son logiciel de gestion consistera alors :

- à caractériser le système formel S pour lequel un objet de la sorte ETAT est un modèle,
- à spécifier les objets de type SORTIE (opérations d'interrogation) et MODIF (mise à jour), ainsi que ceux de type EXPR.

L'implémentation du système (assimilable à la construction informatique du modèle) visera alors à donner une représentation, en termes de structures de données, de l'ETAT, et à fournir les corps des procédures associées aux $SORTIE_S$ et $MODIF_S$.

L'implémentation de apsor et apmod consistera à définir les procédures d'accès et de modification des données de la structure représentant un ETAT.

L'interprétation de eval est légèrement plus complexe. Eval a pour but, dans un contexte d'interrogation (par une OP_{obs}), de s'assurer qu'une expression EXPR (exprimant un "filtre", une sélection d'éléments d'un état) est vraie dans un état donné de la

base (ie que les éléments objets de la sélection y figurent). Dans un contexte de mise à jour (cas d'une OP_{mod}), eval vérifie que l'application de l'OP_{mod} sur l'état de la base n'altère pas la "qualité de modèle de S" de l'état.

On peut alors exprimer la notion de capacité de déduction d'un système en termes d'implémentation de l'opération eval.

Les systèmes classiques travaillent uniquement sur l'état explicite (ie sur les seules informations mémorisées dans la BDP). Ils n'admettent que des expressions complètes (ie désignation non ambiguë et selon une syntaxe rigoureuse des données concernées par une opération). On est généralement, dans l'impossibilité d'obtenir, de façon immédiate, des informations qui ne sont pas explicitées dans un état, même si celles-ci peuvent être issues de cet état par l'application d'une règle logico-mathématique mettant en jeu des informations explicites. On est, dans ce cas, dans l'obligation de réaliser une procédure qui délivre ces informations. (Cette procédure constitue une implantation de la règle logico-mathématique).

Donc, les systèmes dits classiques n'offrent pas de mécanismes suffisamment généraux permettant de combiner, au travers d'une règle déterminée, des informations mémorisées pour obtenir des informations non mémorisées.

Les systèmes déductifs visent à rendre disponibles de tels mécanismes. On considère qu'un système est doté de capacités déductives s'il est capable de combiner des informations d'un état pour en déduire de nouvelles qui ne soient pas explicitement mémorisées dans cet état.

Interprété en termes de la méta-opération eval, cela veut dire que l'implantation de eval est faite de telle sorte que

- dans le premier cas, l'application d'une OP_{obs} est une simple action d'extraction d'informations de l'état (par lecture).

- dans le second cas, elle exploite des règles mettant en jeu des informations de l'état pour restituer des informations implicites (lecture et "calcul" en quelque sorte).

Nous verrons que concrètement, le système déductif devra disposer de connaissances sur l'état, sur les éléments qu'il contient, sur la façon dont certains parmi eux peuvent être combinés...

Nous verrons également qu'outre les mécanismes de rangement, de recherche et de modification des éléments d'un état, le système devra comporter des mécanismes sachant utiliser ces connaissances (mécanisme de déduction).

III - ABSTRACTION ET SYSTEMES DEDUCTIFS

L'utilisation des concepts et des mécanismes d'abstraction, pour élaborer des bases de données, présente a priori un certain nombre d'avantages :

- introduisant, par essence, une approche modulaire, ils permettent de construire une base de façon progresive.
- compte tenu également de leur nature, ils sont un moyen de conceptualiser formellement un monde observé, indépendamment de toute considération de représentation.
- enfin, par leur biais, la conceptualisation produira, conjointement, une abstraction des objets du monde observé et des opérations qui s'y appliquent. (Nous verrons (chapitre I, partie C) que des choix appropriés de symboles d'objets et/ou d'opérations permettront de déterminer un vocabulaire du domaine observé qui soit expressif pour les utilisateurs de la base (au sens où il se rapproche du vocabulaire que ceux-ci ont coutume d'employer)).

Le dernier avantage cité rejoint les points que nous avions

déjà évoqués dans le paragraphe précédent, à savoir que l'abstraction d'un système de données devait concerner à la fois ses états (objets) et leurs transformations (opérations).

Nous allons les expliciter davantage en considérant deux points de vue : l'un statique et l'autre dynamique.

III.1.- Le point de vue statique

Il concerne la spécification de la base, c'est-à-dire l'expression en termes abstraits des résultats de l'observation du monde que la base doit représenter.

Nous considèrerons que, dans tout monde observé, il existe des objets de base (que nous appellerons entités ou agrégats) pertinents. [Par exemple, des personnes et des enfants, ou des étudiants, des enseignants et des programmes d'enseignement (dans un "monde université"), ou encore des capacités d'hébergement et des loisirs (dans un "monde touristique")]. On devra progressivement identifier et caractériser ces objets durant la phase de conceptualisation.

Sur un autre plan, on voudrait autoriser, à ce niveau, l'expression de toute notion (objets ou phénomène) nécessaire à l'expression des besoins des utilisateurs potentiels. L'objectif est de ne pas imposer à ces utilisateurs de raisonner (et d'utiliser) exclusivement en termes d'objets de base.

Par exemple, si pour des besoins spécifiques, on doit disposer d'une notion de CELLULE FAMILIALE, on permettra sa définition (en termes de liaisons époux-épouse et parents-enfants). Comme pour les objets de base, on s'attachera plus à caractériser ces notions (que nous appellerons abstractions relationnelles) - ie à exprimer leur intention (que signifie être une CELLULE FAMILIALE) qu'à considérer comment elles seront représentées.

Le problème essentiel qui a trait à ces abstractions, est

de caractériser leur comportement face à des opérations

- tant d'observation :
  - quel est le chef de la famille X ?
  - quel en est l'aîné ?
  - combien comporte-t-elle d'enfants ?
  - ...
- que de modification
  - un enfant est né chez les X
  - ...

De façon générale, on les caractérisera en termes d'objets de base et d'opération portant sur ces objets de base. L'abstraction résidera alors dans le fait, qu'au niveau externe (celui de l'utilisateur), seul le comportement externe (celui de l'abstraction relationnelle) sera perceptible.

Par exemple, la CELLULE FAMILIALE et les opérations associées seront utilisables en tant que telles au niveau externe. Leur représentation en termes de relations entre des personnes mariées et leurs enfants sera rendue "transparente" aux utilisateurs. La transparence est réalisée par des axiomes qui caractérisent l'abstraction relationnelle et dont l'exploitation par le système déductif permettra de cacher les effets, sur les objets de base, des opérations de l'abstraction relationnelle.

Ainsi, ce point de vue statique correspond à la spécification des phénomènes d'un monde observé. Celle-ci sera exprimée sous la forme de spécifications d'objets ou classes d'objets dit de base et d'abstractions relationnelles.

La construction de ces spécifications se fera à l'aide des mécanismes décrits au § II;2.4 de ce chapitre. Les concepts qu'elle utilise (entité, association, abstraction relationnelle) sont développés de façon plus formelle dans le chapitre I de la partie C qui suit.

III.2.- Le point de vue dynamique

Il concerne la réalisation et l'utilisation d'une base de données physique "conforme" à la spécification qui en a été faite.

On considèrera dans ce cadre :

- la représentation des objets et classes d'objets de la spécification.
- l'implantation de leurs opérations respectives.

III.2.1.- La représentation

Elle met en oeuvre deux procédés :

a) la définition de classes d'objets en extension

Elle consiste à définir un ensemble par énumération de ses éléments. Cette forme de définition sera concrétisée par une structure de représentation (structure de données informatique) des éléments et de leur ensemble d'appartenance et par des procédures de manipulation (accès, modification,...) de cette structure (qui seront les implantations des opérations définies sur l'ensemble et sur ses éléments).

b) la définition en intention

Elle consiste à définir un ensemble d'objets par des propriétés (axiomes). Elle impose de disposer de moyens d'exploitation de ces propriétés pour "manipuler" l'ensemble ainsi défini et ses éléments. C'est le rôle du moteur (ou sous-système) d'inférence.

Il fonctionnera en utilisant les ensembles définis en extension et les axiomes définissant les intentions. Le problème concernant les axiomes, de façon générale, est de déterminer la façon dont ils seront utilisés. En effet :

- certains expriment des contraintes et devront être exploités en tant que tels, c'est-à-dire en tant qu'invariants que doivent respecter intention et extension.

ex : - Un enfant n'a qu'un père
- la polygamie est interdite (sous certains cieux..!!)
- ...

- d'autres peuvent être utilisés, en interrogation pour déduire des informations non mémorisées. Ce sera le cas notamment, de certains axiomes définissant les abstractions relationnelles ou les objets de base que l'on aura décider de définir en intention .
- enfin certains peuvent être utilisés pour propager des actions de mise à jour émises à l'encontre de la base (cf. § I/III/A) [Dans le cadre de ce travail, nous nous sommes plus particulièrement intéressés à la seconde forme d'utilisation c'est-à-dire en interrogation].

Donc, la représentation du système de données spécifié s'intéressera essentiellement à :

- quoi représenter en extension, et, auquel cas, quelles structures de données (et procédures associées) adopter pour la dite extension.
- quoi représenter en intention, et, auquel cas, quels axiomes seront utilisés en tant que règle de déduction de l'intention.

III.2.2.- L'implantation

L'implantation est la phase de concrétisation des opérations applicables sur les objets représentés.

Elle peut être menée suivant plusieurs voies, dont :

- une, en utilisant des outils traditionnels (un langage de programmation classique ou un SGBD existant...). Le problème essentiel est d'adjoindre à ces outils des mécanismes de prise en compte de types abstraits et de déduction.
On fera remarquer, sur ce plan, que les expériences menées

dans le domaine des bases de données déductives ont souvent couplé un sous-système de déduction à un SGBD traditionnel. Nous n'avons pas connaissance de système déductif qui incorpore la notion de type abstrait.

- une autre consiste à utiliser des systèmes ou des langages supportant les mécanismes d'abstraction (TYP [CHAB 79], CLU [LISK 81]...). Dans ce cas, le seul additif à y apporter sera la concrétisation des mécanismes de déduction.
- enfin une troisième consiste à utiliser un système supportant les mécanismes d'abstraction et comportant un mécanisme de déduction. C'est cette voie qu'il nous a été possible de suivre grâce aux possibilités offertes par VEGA :
  - pour spécifier des types abstraits de données et de programmes
  - pour valider des spécifications
  - pour les utiliser au sein de programmes.

D'autre part, l'interface existant entre VEGA et PROLOG permet de bénéficier du mécanisme de résolution de ce dernier pour conduire des inférences.

IV - CONCLUSION

La mise en place de systèmes déductifs nécessite de disposer d'outils permettant de manipuler à la fois des objets concrets et leurs spécifications. On devra, à cet effet, disposer d'une base contenant les ensembles d'objets concrets (ie ceux définis en extension), appelée base de données physique ou base extension, et d'une méta-base contenant la caractérisation des objets et classes d'objets et celle de leurs opérations ; elle sera appelée base abstraite ou base intention.

Un système à capacité de déduction devra être à même d'utiliser l'une et l'autre pour résoudre les problèmes (répondre aux requêtes) qui lui seront soumis.

L'objet de ce chapitre était de montrer quels pouvaient être les bénéfices attendus de l'utilisation des types abstraits (et des mécanismes associés) dans l'élaboration de systèmes de données déductifs. Après une proposition de formalisation d'un méta-système de données, nous avons essayé de montrer, de façon intuitive, que les types abstraits de données pouvaient contribuer à :

- définir rigoureusement un système de données
- faire adopter une démarche progressive de spécification des objets du monde observé "tels qu'ils sont vus" par leurs utilisateurs potentiels
- réaliser des objectifs d'indépendance vis-à-vis de la représentation (tant des objets que des opérations).

Dans la partie qui suit, nous détaillons nos propositions et faisons une présentation de nos différentes expérimentations. Cette partie débute par une formalisation des concepts proposés pour spécifier une base abstraite et s'achève par l'explicitation des mécanismes de réalisation et d'utilisation d'une base. Nous y faisons également, la présentation d'un embryon de méthode dans le but de montrer comment parvenir progressivement de l'observation d'un domaine à son expression en termes des concepts formels définis.

PARTIE C

SINDBAD : UN SYSTÈME INTÉRACTIF DE DÉDUCTION DANS DES BASES ABSTRAITES DE DONNÉES

CHAPITRE 0 : ARCHITECTURE ET PRINCIPES GÉNÉRAUX DU SYSTÈME

1 - INTRODUCTION

Nous avons voulu privilégier deux traits parmi ceux qui peuvent caractériser "l'intelligence" d'un système :

- l'un concerne ses capacités à rendre disponibles des informations qui n'ont pas été explicitement mémorisées.
- l'autre est relatif aux facilités d'utilisation qu'il peut offrir.

Ainsi nous nous sommes préoccupés :

- d'une part, de l'étude des problèmes posés par l'élaboration d'un système doté de capacités déductives, selon une approche par les types abstraits.
- d'autre part, de la définition (et de l'implantation) d'un langage d'utilisation dont les caractéristiques essentielles sont d'autoriser des expressions de requêtes "incomplètes" et de ne requérir que très peu de spécialisation de la part de ses utilisateurs.

Pour atteindre ces objectifs, il importe que le système possède un certain SAVOIR qui lui permet de COMPRENDRE les requêtes qui lui sont soumises avant d'y REPONDRE.

La partie de la thèse que nous abordons maintenant traite ces trois points :

- Au chapitre I, nous présentons la "forme" à donner au SAVOIR du système. Nous y proposons une formalisation, en termes de types abstraits, des concepts et mécanismes utilisables pour représenter un savoir sur un domaine (on dira pour spécifier une base abstraite). Nous y donnons également de brèves indica-

tions d'ordre méthodologique, ayant pour but de montrer comment passer de l'observation du domaine objet de l'application (ou monde observé) à son abstraction à l'aide des concepts présentés.

- Au chapitre II, nous décrivons les outils de SINDBAD destinés à transmettre au système le savoir recueilli sur le monde observé. Ils ont pour principal objectif d'apporter une aide à l'utilisateur dans la spécification d'une base abstraite. Ils présentent, à notre avis, l'intérêt de "cacher" la formalisation de certains concepts,
- Le chapitre III a trait à la réalisation et à l'utilisation d'une base concrète. Nous y montrons, outre la démarche à suivre, les mécanismes que doit mettre en oeuvre un système déductif pour REPONDRE à ses utilisateurs. Dans ce chapitre, nous décrivons, de façon duale,
  - une réalisation de base de données déductive à l'aide d'un système relationnel "idéal" (ABSTRACT-R).
  - une expérimentation menée en VEGA.
- Le chapitre IV concerne l'activité de COMPREHENSION que développe SINDBAD, en réaction à des requêtes incomplètes.

  Nous y décrivons le langage d'utilisation (ou langage des bribes) et le mécanisme de compréhension des requêtes écrites dans ce langage. Nous y présentons également quelques réflexions relatives aux moyens "d'accroître l'intelligence" du système.

II - UN APERCU SUR L'ARCHITECTURE GLOBALE DU SYSTEME

II.1.- Principaux constituants

a) La méta-base contient la définition de l'abstraction faite du monde observé. Cette définition (ou spécification de la base abstraite) comporte, pour chaque objet ou classe d'objets du monde réel :

- des éléments syntaxiques
  - symboles de sortes et d'opérations
  - profil des opérations
- des éléments sémantiques qui caractérisent les objets du monde observé (contraintes sémantiques, définitions de la sémantique des opérations...).

b) La base extension définit un modèle de l'abstraction décrite par la méta-base. La réalisation d'un tel modèle consistera, pour chaque abstraction de données, à faire le choix d'une structure de données concrète et à implanter les opérations abstraites associées.

c) Le but visé est de concevoir un système où l'on distinguerait deux "couches" :

- l'une abstraite, visible, sur laquelle l'utilisateur peut exprimer ses requêtes.
- l'autre concrète, constituée par le logiciel de déduction et de gestion de la base physique (extension).

De ce fait, la réalisation de la base extension ne concerne que l'implémenteur. Par ailleurs, le fait d'autoriser l'utilisateur à exprimer ses requêtes sur la structure abstraite impose de disposer de moyens (explicitation de la fonction de représentation, outil de traduction) de transformation des expressions abstraites des requêtes en des expressions concrètes.

II.2.- Architecture globale du système

Le système que nous avons réalisé traite essentiellement la

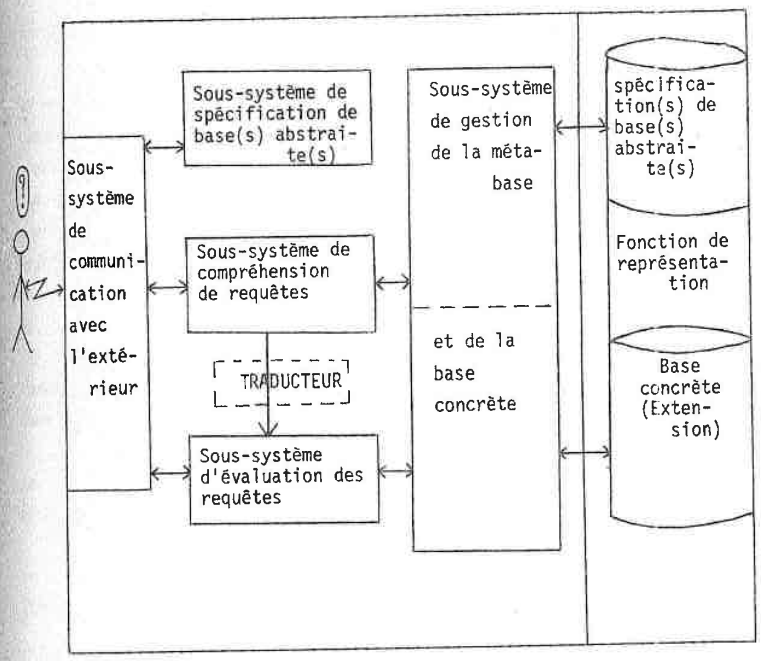
couche abstraite. Il comporte :

- un sous-système de définition de base(s) abstraite(s)
- un sous-système de compréhension de requêtes.

La couche concrète a fait l'objet d'une expérimentation, par le biais de VEGA, dans le cadre de l'élaboration d'une maquette de système déductif pour l'information touristique (projet MONT-LOZERE [BOUD 82], [BOUR 82], [CHAB 82]).

L'intégration des deux couches (abstraite et concrète) nécessiterait une étape de traduction de l'expression abstraite de la requête (après sa complétion par le sous-système de compréhension) en une expression concrète. (Pour valider nos propositions en la matière, nous avons également réalisé un outil de traduction des requêtes complétées en des requêtes PROLOG).

L'architecture (macroscopique) du système est décrite par le schéma ci-dessous :



- Schématisation de l'architecture globale du système -

CHAPITRE I : CONCEPTS ET MÉCANISMES POUR LA SPÉCIFICATION D'UNE BASE ABSTRAITE DE DONNÉES

La conceptualisation d'un monde observé vise à produire une abstraction de ce monde. On appellera schéma de la base ou spécification de la base abstraite le résultat de cette conceptualisation.

Les concepts que nous utiliserons pour la présentation d'un schéma sont pour la plupart ceux des modèles entité/association. La différence essentielle, par rapport aux approches traditionnelles, est que nous ne nous intéresserons pas uniquement aux objets (entités/associations) mais également à leurs opérations.

Le processus de définition du schéma consistera à définir les types et les abstractions relationnelles nécessaires à la conceptualisation du monde observé. Nous supposons disposer d'une bibliothèque de spécification (également appelée environnement de spécification) contenant des types de base et des constructeurs de types. La définition du schéma se fera alors par importation de certains types de l'environnement et/ou par construction d'autres.

Ce chapitre décrit les notions sur lesquelles est fondé notre système. Nous présenterons successivement :

- une formalisation des concepts de base que nous utiliserons pour spécifier une base abstraite
- un (méta)-type C-SCHEMA qui sert de constructeur de schémas de bases de données
- comment utiliser concrètement les notions formelles définies
- enfin nous définirons la notion d'instance de base de données.

I - CONCEPTS POUR LA SPECIFICATION D'UNE BASE ABSTRAITE

Il est souvent plus "raisonnable" de caractériser des objets ou des classes d'objets (entités) par une liste de propriétés que comme étant le résultat de l'application d'une suite d'opérations. Ainsi il semble plus "naturel" de désigner une personne par son nom, sa date de naissance, sa profession... que par une formule exprimant l'application d'un certain nombre d'opérations sur un objet initial (approche algébrique des types abstraits). De même, on caractériserait une entreprise par son siège, sa raison sociale, son capital... et l'on pourrait alors définir des opérations sur l'un et/ou l'autre des objets (exemple : embauche d'une personne par une entreprise, faillite d'une entreprise, mariage de deux personnes...).

C'est l'approche que nous adopterons. Nous allons préciser les concepts qu'elle utilise puis nous exposerons quelques considérations d'ordre méthodologique.

[Le chapitre suivant présentera les outils logiciels existants dans SINDBAD qui ont pour rôle d'apporter une aide à l'utilisateur pour spécifier des bases abstraites de données].

I.1.- Notion d'attribut

Un attribut est un identificateur associé à un domaine ; Les éléments de celui-ci sont atomiques, l'atomicité voulant exprimer qu'un élément d'un domaine d'attribut ne peut pas être une collection d'objets.

On considèrera ainsi :

- des domaines simples

exemple : `type nom = string`
`type age = entier ;` string et entier sont des domaines simples.

- des domaines composés (obtenus généralement par l'application du constructeur <> (produit cartésien) sur des domaines simples).
  Exemple : type date = <J:entier, M:entier, A:entier>

Les attributs permettent de caractériser les entités et les associations. On les appellera alors constituants de l'entité ou de l'association. Un attribut sera également appelé simple ou composé selon que son domaine associé est lui-même simple ou composé.

I.2.- Notion d'entité

Une entité est une classe d'objets dont l'existence peut être décelée lors de la conceptualisation d'un monde.

Pour la définir, de façon plus formelle, nous caractériserons un objet de la classe (type O-ENTITE) puis la classe elle-même (type ENTITE) en tant que collection de O-ENTITE$_S$.

I.2.1.- Définition d'une O-ENTITE

Si S et F désignent respectivement les symboles de sortes et d'opérations de la spécification d'une base abstraite, une O-ENTITE E est le résultat du "tupling" des constituants C d'un ensemble C_E d'attributs qui vérifie les propriétés suivantes :

(P1) il existe T, $T : C_E \rightarrow S$
tel que $\forall c, c \in C_E, \exists t, t \in S : T(c) = t$

(P2) il existe Acc, $Acc : C_E \rightarrow F$
tel que $Acc(c) : E \rightarrow T(c)$
et $\forall c,c' ; c \in C_E, c' \in C_E : Acc(c) = Acc(c') \Rightarrow c = c'$

Ces propriétés expriment le fait que

- chaque constituant d'une O-entité est typé (la fonction T associe à chacun d'eux son type)
- à chaque constituant c est associéeune fonction Acc(c) ayant comme domaine la O-entité et comme co-domaine le type du constituant
- enfin un même attribut c ne peut pas figurer plusieurs fois dans la même entité.

Le type ci-dessous servira par la suite, de constructeur de O-ENTITES.

type O-ENTITE(C1:T1,C2:T2,...,CLE:TCLE,...,C_n:T_n)
  sortes O-ENTITE,T1,...,TCLE,...,T_n,BOOL
  opérations
    Faire-OENT : T1×T2×...×Tn → O-ENTITE
      C1 : O-ENTITE → T1
      C2 : O-ENTITE → T2
      ⋮
      CLE : O-ENTITE → TCLE
      ⋮
      C_n : O-ENTITE → T_n
    Egal-OENT : O-ENTITE×O-ENTITE → BOOL

  axiomes
    soit t1:T1;...;t:TCLE;...;t_n:T_n;O:O-ENTITE
    . C1(Faire-OENT(t1,...,tn)) = t1
      ...
    . CLE(Faire-OENT(t1,...,t,...,tn)) = t
      ...
    . Cn(Faire-OENT(t1,...,tn)) = tn
    . Egal-OENT(Faire-OENT(t1,...,t,...,tn),O) =
        si $EGAL_{T1}$(t1,C1(O)) ∧ ...
        ∧$EGAL_{CLE}$(t,CLE(O))∧...
        ∧$EGAL_{Tn}$(tn,Cn(O))
        alors vrai sinon faux
ftype

I.2.2.- Notion d'ENTITE

Le type O-ENTITE caractérise une classe d'objets. La spécification donnée permet de s'intéresser à un élément de la classe. Mais dans le cadre de beaucoup d'applications, dont ceux de bases de données on s'intéresse également à toute la classe, au sens où on veut pouvoir lui adjoindre des éléments, les observer, en supprimer... Cela nécessite de conserver les objets de la classe.

De ce fait, à chaque O-ENTITE mise en évidence lors de l'observation du monde concerné par l'application, sera associé un type ENTITE qui permettra de considérer une collection de O-entités.

```
type ENTITE(O-ENTITE)
    sortes ENTITE,O-ENTITE,BOOL
    opérations
       S-INIT :                  → ENTITE
       AJOUT : ENTITE×O-ENTITE → ENTITE
       RETRAIT : ENTITE×O-ENTITE → ENTITE
       PRESENCE : ENTITE×O-ENTITE → BOOL
    axiomes  %ils sont similaires à ceux d'un type ensemble%
       soit s : ENTITE ; e,e' : O-ENTITE
       • PRESENCE(S-INIT(),e) = Faux
       • PRESENCE(AJOUT(s,e),e') = si Egal-OENT(e,e')alors 'vrai'
                                   sinon PRESENCE(s,e')
       • RETRAIT(S-INIT( ),e) = S-INIT()
       • RETRAIT(AJOUT(s,e),e') = si Egal-OENT(e,e')alors RETRAIT(s,e')
                                  sinon AJOUT(RETRAIT(s,e'),e)
ftype
```

I.2.3.- Utilisation de la notion d'ENTITE

Intuitivement, une ENTITE est une classe d'objets du monde réel. Lors de la conceptualisation de ce monde, on essaiera de

mettre en évidence l'ensemble des classes qui y sont présentes. Pour chacune d'elles, on caractérisera les propriétés d'un objet de la classe par une liste d'attributs.

Formellement, le constructeur de types O-ENTITE et le type paramétré ENTITE nous permettront de spécifier ces classes d'objets. Nous allons expliciter, sur un exemple, les mécanismes mis en oeuvre.

Exemple : Considèrons un ensemble de personnes. Une personne peut être caractérisée par son numéro d'identification (# P), son nom (NOM), son prénom (PRENOM) et son âge AGE.

```
type 1 PERSONNE=O-ENTITE(# P:entier,NOM:CHAINE,PRENOM:CHAINE,
                         DATENAISS:DATE)
    sortes 1 PERSONNE,O-ENTITE,entier,chaîne,bool,date
    opérations
    (*) PERSONNE : entier×chaîne×chaîne×entier → 1 PERSONNE
    (*) # P      : 1 PERSONNE          → entier
    (*) NOM      : 1 PERSONNE          → chaîne
    (*) PRENOM   : 1 PERSONNE          → chaîne
    (*) DATENAISS: 1 PERSONNE          → date
        AGE      : 1 PERSONNE×DATE     → entier
        PLUSVIEUX: 1 PERSONNE×1 PERSONNE → BOOL
    (*) MEMEPERSONNE : 1 PERSONNE×1 PERSONNE → BOOL
    axiomes
      soit p1,p2:1 PERSONNE;d,d':DATE;s:Entier,n,p:CHAINE
    (1) PLUSVIEUX(PERSONNE(s,n,p,d),p1)=
            si infd(DATENAISS(p1),d)alors vrai sinon faux
    (2) AGE(PERSONNE(s,n,p,d),d') = moinsd(d',d)
```

```
(3)  % axiomes de 0-ENTITE dans lesquels
        on renomme FAIRE-0ENT en PERSONNE
                   CLE        en #P
                   C2         en NOM
                   C3         en PRENOM
                   C4         en DATENAISS
                   Egal-0ENT  en MEMEPERSONNE %
ftype
```

La collection des personnes peut alors être spécifiée à l'aide du type ENTITE de paramètre 1 PERSONNE.

```
type SPERSONNE = ENTITE(1 PERSONNE)
    sortes SPERSONNE, 1 PERSONNE, BOOL, ENTITE
    opérations
       SINIT :                    → SPERSONNE
    (*) AJTPERS : SPERSONNE×1PERSONNE → SPERSONNE
    (*) SUPPERS : SPERSONNE×1PERSONNE → SPERSONNE
    (*) EXISTPERS : SPERSONNE×1PERSONNE → BOOL
    axiomes
       % ceux de ENTITE dans lesquels on effectue les renommages
         suivants
            S-INIT      en S-INIT
            AJOUT       en AJTPERS
            RETRAIT     en SUPPERS
            PRESENCE    en EXISTEPERS %
ftype
```

Dans ces spécifications, les lignes marquées d'un astérisque (*) auraient pu être omises. En effet, elles définissent des opérations de 0-ENTITE ou de ENTITE. Les mécanismes de construction de types (à l'aide de constructeurs ou de types paramétrés) permettent de faire hériter les types construits, des opérations des

types utilisés pour leur construction.

Pour notre part, nous les avons faites figurer sur les spécifications de 1PERSONNE et PERSONNE pour expliciter le mécanisme de construction et pour montrer d'autres mécanismes que l'on peut mettre en oeuvre.

Le premier est celui de "renommage" des opérations dont l'intérêt réside dans la lisibilité de la spécification obtenue. En effet, un "choix judicieux" des symboles d'opérations utilisés pour le renommage peut contribuer à faciliter la lecture (puis l'utilisation) de la spécification par un non spécialiste. Ce mécanisme permet de rapprocher le vocabulaire de la spécification de celui en vigueur dans le domaine qui fait l'objet de la conceptualisation. (Par exemple, dans le cadre d'une gestion de l'ETAT-CIVIL d'une mairie, renommer AJOUT en NAISSANCE et RETRAIT en DECES serait un "choix judicieux").

Sur un autre plan, on remarquera que dans 1PERSONNE, AGE et PLUSVIEUX sont des opérations dérivées (cf. chapitre I, partie B) de celles de la spécification de base. Les axiomes qui les caractérisent sont définis à l'aide d'opérations de base (infd, moinsd sont des opérations du type DATE qui permettent respectivement de comparer deux dates sur le critère d'infériorité et de faire la différence de deux dates).

Ainsi, outre le mécanisme de renommage et de dérivation d'opérateurs, on pourra également

a) enrichir la spécification d'une entité ou d'une 0-entité par d'autres opérations

Exemple : PERE : 1PERSONNE →1PERSONNE

b) opérer des restrictions

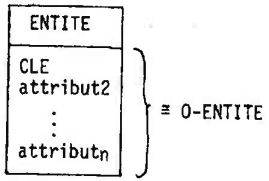
- sur les sortes

Exemple : On peut restreindre la sorte des NOMs à une CHAINE de moins de 20 caractères ou encore aux seules chaînes commençant par 'BOU'

- sur les opérations

Exemple : On pourra interdire l'accès à certains constituants confidentiels en ne rendant pas disponibles les fonctions d'accès associées aux dits constituants.

Enfin, dans ce qui suit, on utilisera le graphisme ci-dessous pour symboliser à la fois une ENTITE et son O-ENTITE paramètre.



I.3.- Notion d'association

L'association est un procédé permettant d'exprimer des "liaisons sémantiques" entre des entités. Elle est caractérisée par
- sa collection (ou liste des entités associées)
- sa dimension (ou nombre d'entités associées)
- et les cardinalités de ses fonctions rôles.

Nous avons vu, dans le chapitre I de la partie A concernant les SGBD traditionnels, que la sémantique d'une association pouvait être exprimée à l'aide d'un prédicat et de "fonctions rôles", ces dernières étant elles-mêmes caractérisées par leur nature (totale, partielle, mono ou multivaluée). Nous avons également essayé de montrer (au § III du même chapitre) "l'explosion combinatoire"

que peut occasionner une explicitation fonctionnelle de la sémantique des associations, en liaison avec leur dimension.

De ce fait, dans le sous-système de compréhension de requêtes, nous nous sommes délibérément limités aux associations binaires (même si dans le cadre de la maquette du système d'informations touristiques sur la LOZERE nous avions considéré des relations d'ordre supérieur à deux). On pourrait ultérieurement s'intéresser à la généralisation des mécanismes de complétion de requêtes, mis en oeuvre par le système de compréhension, à des associations de dimension supérieure à deux.

Nous allons spécifier la notion d'association en suivant une démarche analogue à celle suivie par la spécification de l'ENTITE. On considèrera l'association "atomique" (type O-ASSBIN) définie entre deux objets puis la collection d'associations (type ASSBIN) définie entre deux classes d'objets.

I.3.1.- Spécification d'une association de deux objets : le type O-ASSBIN

Une association binaire "atomique" est caractérisée par la donnée des O-entités qu'elle relie et de ses attributs éventuels. A chaque O-entité et chaque attribut est associée une fonction d'accès (fonctions "rôles").

type O-ASSBIN(f_1:O-ENTITE,f_2:O-ENTITE,C_1:T_1,...,C_n:T_n)
  sortes O-ASSBIN,O-ENTITE,T_1,...,T_n,Bool
  opérations
    faire-O-ASSBIN : O-ENTITE×O-ENTITE T_1×...×T_n → O-ASSBIN
      f_1 : O-ASSBIN → O-ENTITE
      f_2 : O-ASSBIN → O-ENTITE
      C_1 : O-ASSBIN → T_1
      ...
      C_n : O-ASSBIN → T_n
    Eg-orel : O-ASSBIN×O-ASSBIN → BOOL

axiomes

soit e_1,e_2:O-ENTITE;t_1:T_1;...;t_n:T_n;r:O-ASSBIN

-f_1(Faire-OASSBIN($e_1,e_2,t_1,\ldots,t_n$))=e_1

-f_2(Faire-OASSBIN($e_1,e_2,t_1,\ldots,t_n$))=e_2

-C_1(Faire-OASSBIN($e_1,e_2,t_1,\ldots,t_n$))=t_1

-...

-C_n(Faire-OASSBIN($e_1,e_2,t_1,\ldots,t_n$))=t_n

-Eg-ore1(Faire-OASSBIN($e_1,e_2t_1,\ldots,t_n$),r) =

si Egal-OENT(e_1,f_1(r))
∧ Egal-OENT(e_2,f_2(r))
∧ $Egal_{T_1}$(t_1,C_1(r))
∧ ...
∧ $Egal_{T_n}$(t_n,C_n(r))
alors Vrai sinon faux

ftype

Exemple : Une liaison entre une personne et un de ses enfants s'exprimera par une O-ASSBIN entre une O-ENTITE 1PERSONNE et une O-ENTITE 1ENFANT.

type PATERNITE=O-ASSBIN(1PERE:1PERSONNE,1ENF:1ENFANT)
ftype

I.3.2.- Spécification d'une collection d'associations : le type ASSBIN

Le type ASSBIN spécifie une association entre deux entités. A chacune d'elles est attachée une cardinalité, exprimée par un couple (min,max) d'entiers, qui indique le nombre minimum et maximum de fois qu'une même O-ENTITE d'une ENTITE de l'association peut apparaître dans les instances de l'association.

L'utilisation de la notion de cardinalité dans la spécification nous évite d'avoir à spécifier plusieurs types ASSBIN, chacun correspondant à une des natures d'associations déterminées par les fonctions (VALF et INVERSF) qui définissent l'association. Ainsi les cardinalités et les invariants (1) et (2) qui s'y rapportent permettent de "couvrir" leurs diverses natures. [Ainsi, des cardinalités ((1,1),(1,1)) expriment une bijection, des cardinalités ((0,1),(n,m)) une fonction partielle à gauche..]

Par ailleurs, la spécification de ASSBIN utilisera des opérations supposées définies sur les entiers et les intervalles d'entiers. Il s'agit en l'occurrence de :

≥ : ENTIER×ENTIER → BOOL

> : ENTIER×INTERVAL(ENTIER,ENTIER) → BOOL.

[Leur sémantique est "intuitivement claire"].

Elle utilisera également, l'opération CARD sur ENTITE (Nous ne l'avions pas faite figurer sur la spécification de ENTITE). Elle a comme profil CARD : ENTITE → ENTIER, et comme axiomes associés :

CARD(S-INIT()) = 0

CARD(AJOUT(s,e)) = si EXISTE(s,e)alors CARD(s)
sinon SUCC(CARD(s))

(où SUCC est l'opération successeur d'entier).

type ASSBIN(O-ASSBIN,(min1:entier,max1:entier),(min2:entier, max2:entier))

sortes ASSBIN,O-ASSBIN,entier,O-ENTITE,ENTITE,BOOL

opérations

ASS-INIT : → ASSBIN
AJTASS : ASSBIN×O-ASSBIN → ASSBIN
SUPRASS : ASSBIN×O-ASSBIN → ASSBIN
EXISTASS : ASSBIN×O-ASSBIN → BOOL
VALF : ASSBIN×O-ENTITE → ENTITE
INVERSF : ASSBIN×O-ENTITE → ENTITE

sémantique

soit A:ASSBIN;a_1,a_2:0-ASSBIN;e_1,e_2:0-ENTITE

invariants

(1) max1 ≥ min1∧max2 ≥ min2

(2) ∀e_1,e_2

CARD(VALF(A,e_1)) ∈ [min1,max1]

CARD(INVERSF(A,e_2)) ∈ [min2,max2]

axiomes

- SUPRASS(ASS-INIT(),a_1)=ASS-INIT( )
- SUPRASS(AJTASS(A,a_1),a_2)=

si Eg-orel(a_1,a_2)alors SUPRASS(A,a_2)

sinon AJTASS(SUPRASS(A,a_2),a_1)
- EXISTASS(ASS-INIT(),a_1) = Faux
- EXISTASS(AJTASS(A,a_1),a_2)=si Eg-orel(a_1,a_2)

alors vrai sinon EXISTASS(A,a_2)
- VALF(ASS-INIT(),e_1) = vide( )
- VALF(AJTASS(A,a_1),e_1)=

si Eg-OENT(f_1(a_1),e_1)

alors AJOUT(VALF(A,e_1),f_2(a_1))sinon VALF(A,e_1)
- INVERSF(ASS-INIT(),e_1) = vide( )
- INVERSF(AJTASS(A,a_1),e_2) =

si Eg-OENT(f_2(a_1),e_2)

alors AJOUT(VALF(A,e_2),f_1(a_1))

sinon INVERSF(A,e_2)

ftype

Exemple : On peut généraliser l'association 1PATERNITE à un ensemble de PERSONNEs et d'ENFANTs de la façon suivante :

type PATERNITE = ASSBIN(1PATERNITE,(0,n),(1,1))

sortes PATERNITE,1PATERNITE,ENFANT,1ENFANT,PERSONNE,1PERSONNE, ENTIER,BOOL

opérations

PATER-INIT : → PATERNITE

AJT PATER : PATERNITE×1PATERNITE → PATERNITE

SUPR PATER : PATERNITE×1PATERNITE → PATERNITE

EST-PERE-DE : PATERNITE×1PERSONNE → ENFANT

EST-ENF-DE : PATERNITE×1ENFANT → PERSONNE

PERE-ENFANT : PATERNITE×1PATERNITE → BOOL

sémantique

soit P:PATERNITE;p:1PERSONNE;e:1ENFANT

invariants

-CARD(EST-PERE-DE(P,p)) ∈ [0,n]

-CARD(EST-ENF-DE(P,e)) ∈ [1,1]

axiomes

%ceux de ASSBIN avec les "renommages" suivants :

ASS-INIT → PATER-INIT

AJT-ASS → AJT PATER

SUPRASS → SUPR PATER

EXISTASS → PERE-ENFANT

VALF → EST-PERE-DE

INVERSF → EST-ENF-DE %

ftype

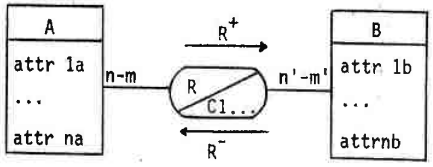
Ainsi ASSBIN est, en quelque sorte, un moyen de "typer" les associations. Il permet également de ne faire aucune supposition sur la représentation qui en sera faite (ie en extension, par énumération de l'ensemble des couples 0-entités associées, ou en intention, c'est-à-dire grâce à "une procédure" de calcul des couples associés).

Sur un autre plan, dans certains cas, on pourra mettre en oeuvre des mécanismes de restriction sur les opérations d'une association, en fonction du domaine d'application considéré. Par

exemple, la relation de PATERNITE étant "immuable", on pourrait interdire toute suppression de couple de valeurs associées en "ôtant" l'opération SUPRASS de l'association PATERNITE.

Nous montrerons dans le chapitre qui suit, comment l'utilisateur de SINDBAD pourra se départir de ces formalisations. En effet, des outils interactifs de SINDBAD l'aideront à spécifier une base de données en le déchargeant de certaines formalisations (comme celles des ASSBIN de sa base) et lui offrant des possibilités de mettre en oeuvre des mécanismes de restriction, d'enrichissement, ..., de types.

Enfin, dans ce qui suivra, nous utiliserons le graphisme ci-dessous pour symboliser une association R((A,B),(n,m),(n',m'))



[C_1,... sont les attributs de l'association]
[R^+ désigne VALF et R^- INVERSF]
[n-m et n'-m' sont les cardinalités respectives de R^+ et R^-]

I.4.- Notion d'abstraction relationnelle

L'utilisation des notions d'entité et d'association permet de représenter des classes d'objets et des classes de laisons entre objets, considérées toutes deux comme étant de "base". En effet, nous faisons la supposition que dans tout monde observé existent de telles classes que l'on s'attachera à "découvrir" en premier

lieu lors de la conceptualisation de ce monde.

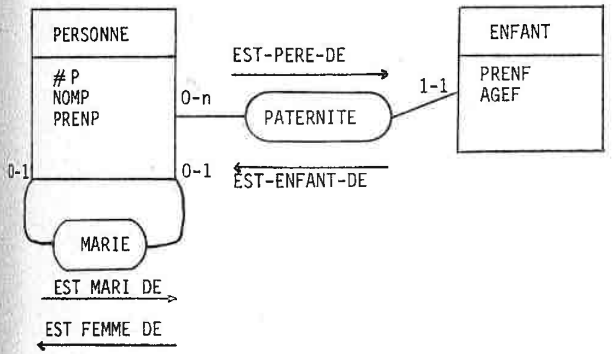
Mais dans certains cas, nous pouvons avoir besoin d'autres formes de représentation ; c'est le cas où l'on doit considérer des classes dont les objets peuvent ne pas être élémentaires (de base). Ces classes seront appelées abstractions relationnelles.

Elles ont pour objectif de fournir des formes de conceptualisation de certaines parties du monde observé qui soient les plus adaptées aux besoins d'un ou d'une classe d'utilisateurs. Elles permettent de "faire voir" ces parties du monde "comme leurs utilisateurs les perçoivent". Elles sont, par conséquent, un moyen de ne pas déformer leur perception par leurs utilisateurs.

En effet, elles permettent à ces derniers de ne pas exprimer tout objet ou classe d'objets exclusivement en termes de ceux que nous avons qualifié "de base" (ENTITE,ASSOCIATION).

Elles sont, pour la plupart, induites par les besoins en traitement sur la future base de données, ressentis et recensés dans le monde qui fait l'objet de la conceptualisation.

Exemple : Considérons le SCHEMA suivant :



Certaines applications sur la base définie par ce SCHEMA, pourraient nous amener à considérer :

- les couples de personnes mariées, ou
- les cellules-familiales (mari, femme et enfants éventuels), ou
- les familles nombreuses (ayant au moins trois enfants).

La notion d'abstraction relationnelle va nous permettre d'exprimer cela.

Nous allons, dans un premier temps, décrire les moyens techniques que l'on doit mettre en oeuvre pour le spécifier, puis, dans un second temps, la confronter avec la notion de vue que l'on rencontre dans les SGBD traditionnels.

I.4.1.- Techniques de spécification des abstractions relationnelles

De façon générale, une abstraction relationnelle sera spécifiée en termes d'objets ou classes d'objets de base, sur lesquelles elle est définie. Sa spécification comportera une caractérisation

- de la "structure" d'un objet de la classe que détermine l'abstraction relationnelle
- des opérations qui peuvent éventuellement lui être appliquées.

1) Caractérisation de la structure

Elle consiste à déterminer précisément les constituants d'un objet de la classe et à exprimer les relations éventuelles qui les lieraient (expression de la sémantique de l'abstraction).

a) Les constituants sont le plus souvent :

- des attributs d'entités de base

Exemple : La "structure" d'un couple pourrait être définie par la donnée de ((NOM1,PRENOM1),(NOM2,PRENOM2)) où NOM1, NOM2 correspondent à l'attribut NOMP de la O-ENTITE PERSONNE et PRENOM1 et 2 correspondent à PRENP de la même O-ENTITE.

- des entités ou des collections d'entités

Exemple : CELLULE-FAMILIALE pourrait être définie par la donnée de
- 1PERSONNE:O-ENTITE (mari)
- 1PERSONNE:O-ENTITE (femme)
- ENFANT:ENTITE (1ENFANT)

- des entités de base dont on restreint les attributs aux seuls attributs nécessités par l'abstraction relationnelle

Exemple : Dans CELLULE-FAMILIALE, on aurait pu restreindre les attributs de ENFANT un seul attribut PRENF.

- des collections d'entités restreintes aux seules occurrences d'entité vérifiant une propriété donnée.

Exemple : L'abstraction relationnelle FAMILLE NOMBREUSE "concernerait" dans la collection de PERSONNES uniquement celles ayant plus de deux enfants.

- enfin dans certains cas, ils peuvent être le résultat de l'application de quelque fonction que l'on devra alors expliciter.

Exemple : Supposons que l'on veuille considérer une CELLULE FAMILIALE définie comme précédemment, et par un constituant supplémentaire TAILLE indiquant le nombre de membres de la cellule (2+nombre d'enfants).
Sa "structure" devient :

```
CELLULE FAMILIALE (1PERSONNE:O-ENTITE,
                   1PERSONNE:O-ENTITE,
                   ENFANT:ENTITE,
                   TAILLE:INTEGER)
```

avec TAILLE : CELLULE FAMILIALE → INTEGER,

définie comme suit :

$\forall C,C$:CELLULE FAMILIALE

TAILLE(C) = CARD(ENFANT(C))+2

- ...

b) L'expression des relations entre les constituants de l'abstraction relationnelle permet d'expliciter la sémantique de celle-ci. Ces expressions utiliseront généralement des éléments de la spécification de la base abstraite (entités, associations de base et opérations associées).

Dans ce qui suit, tant pour exprimer les relations entre les constituants d'une abstraction relationnelle que pour expliciter la sémantique des opérations de l'abstraction, nous utiliserons des formules du calcul des prédicats du premier ordre (avec l'égalité).

Exemple :

soit C : CELLULE FAMILIALE

p_1, p_2:1PERSONNE

$\{e_1, e_2, \ldots, e_n\}$:ENFANT

$\forall C$, C = CELLULE($p_1, p_2, (e_1, \ldots, e_n)$)

(1) $n \geq 0$

(2) EST-MARI-DE(p_1) = p_2

(3) $n > 0 \Rightarrow \forall i \in [1,n]$, EST-PERE-DE($p_1$)=$e_i$

Ainsi, caractériser un objet de la classe déterminée par une abstraction relationnelle revient à expliciter de façon rigoureuse ce que signifie être membre de cette classe, en faisant usage des éléments de base de la base abstraite.

La caractérisation des opérations qui concernent éventuellement l'abstraction relationnelle se fera selon le même principe.

2) Caractérisation des opérations d'une abstraction relationnelle

Si l'on avait (l'obligation nous étant faite par les utilisateurs) à définir des opérations sur l'abstraction relationnelle, on devra donner la syntaxe et la sémantique de chacune d'elles.

La spécification des abstractions relationnelles semble relever beaucoup plus de la spécification de problème ([FIN 79], [DER & FI 79]), que de celle, à proprement parler, de types de données. Elle vise à découvrir :

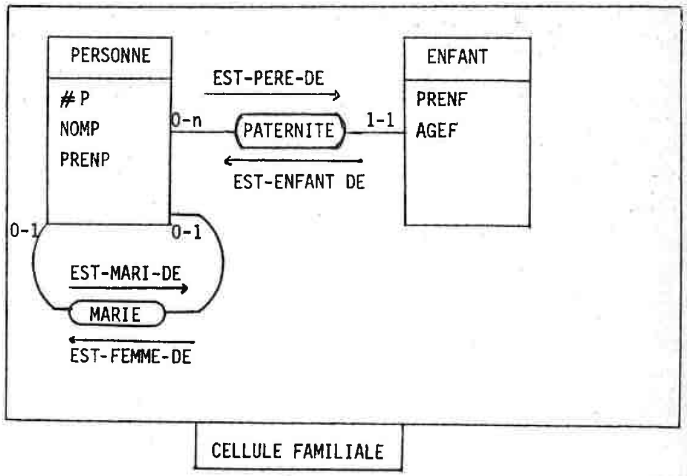
- un univers (ou structure de l'abstraction) constitué à partir d'objets de base (que l'on caractérise et dont on caractérise également les relations).
- un (des) énoncé(s) formulé(s) en termes de cet univers abstrait. Les énoncés sont ultérieurement transformés en des énoncés sur les objets de base.

Pour notre part, nous présenterons la spécification d'une abstraction relationnelle en indiquant :

- les objets/classes d'objets sur lesquels elle est construite
- le profil de ses éventuelles opérations
- les axiomes qui caractérisent ses objets et/ou ses opérations.

Enfin, d'un point de vue graphisme, nous représenterons une abstraction relationnelle par un rectangle que l'on nommera par le nom donné à l'abstraction et dont l'intérieur contient le sous-ensemble du schéma de la base sur lequel elle est construite (Nous ferons remarquer toutefois que cette visualisation schématique peut être insuffisante pour exprimer le sens de l'abstraction relationnelle ; il faudra alors se référer à sa spécification pour en saisir la signification).

Pour clore ce paragraphe, nous donnons une spécification de CELLULE FAMILIALE qui résume notre discours.



ABSTRACTION RELATIONNELLE : CELLULE FAMILIALE

objets : 1PERSONNE,ENFANT,ENTIER

opérations

CELLULE : $1PERSONNE^2 \times ENFANT \rightarrow$ CELLULE FAMILIALE

PERE : CELLULE FAMILIALE → 1PERSONNE

MERE : CELLULE FAMILIALE → 1PERSONNE

ENFT : CELLULE FAMILIALE → ENFANT

AINE : CELLULE FAMILIALE → 1ENFANT

NOUVEAUNE : CELLULE FAMILIALE×1ENFANT → CELLULE FAMILIALE

axiomes

Soit C : CELLULE FAMILIALE

p_1, p_2 : 1PERSONNE

$\{e_1, e_2, \ldots, e_n\}$: ENFANT

(1) $CELLULE(p_1, p_2, (e_1, \ldots, e_n))$
$\Rightarrow EST\text{-}MARI\text{-}DE(p_1) = p_2$

(2) $n \geq 0 \wedge CELLULE(p_1, p_2\ (e_1, \ldots, e_n))$
$\Rightarrow \forall i, i \in [1,n]: EST\text{-}PERE\text{-}DE(p_1) = e_i$

(3) $PERE(CELLULE(p_1, p_2, (e_1, \ldots, e_n))) = p_1$

(4) $MERE(CELLULE(p_1, p_2, (e_1, \ldots, e_n))) = p_2$

(5) $ENFT(CELLULE(p_1, p_2, (e_1, \ldots, e_n))) = (e_1, \ldots, e_n)$

(6) $AINE(CELLULE(p_1, p_2, (e_1, \ldots, e_n))) = e_i / \forall j, j \in [1,n] \setminus \{i\}$
$AGEF(e_i) > AGEF(e_j)$

(7) $NOUVEAUNE(CELLULE(p_1, p_2, (e_1, \ldots, e_n)), e_{n+1}) =$
$CELLULE(p_1, p_2, (e_1, \ldots, e_n, e_{n+1}))$

fabstraction

I.4.2.- Abstraction relationnelle VS notion de vue

Cette notion d'abstraction relationnelle n'est pas sans rappeler celle de VUE. Une VUE est une fenêtre dynamique sur la base. Dans le cadre de systèmes relationnels, elle définit une relation virtuelle à l'aide de relations de base (ie du schéma de la base) ou d'autres vues. Ainsi dans SYSTEM-R [ASTR 76], la formule

DEFINE VIEW V[(liste de constituants)]
AS <expression relationnelle de qualification>

permet de définir une vue V comme une relation de constituants <liste de constituants> et dont les tuples vérifient l'<expression relationnelle de qualification>.

La vue se comporte comme les relations de base. Néanmoins sa mise à jour pose certains problèmes dont celui que nous avions dénommé problème de propagation de mises à jour (ie dans ce cas, comment répercuter (et contrôler) la mise à jour d'une vue sur les relations qui ont été utilisées pour la définir).

L'abstraction relationnelle peut s'assimiler à une vue sur laquelle peuvent être définies et appliquées des opérations. La caractérisation de sa structure peut être vue comme un "mapping" de l'abstraction sur le schéma de la base. [La fonction de mapping est définie inductivement sur la nature des constituants de la structure de l'abstraction (par localisation de ceux-ci dans les entités de base ou par qualification de la façon de les obtenir à partir des objets ou classes d'objets (entités, associations, opérations) de la spécification de la base abstraite].

[Nous donnons au paragraphe III de ce chapitre quelques indications pratiques pour la détermination des abstractions relationnelles].

I.5.- Conclusion

Sur certains plans, la spécification des abstractions relationnelles ressemble fort à celle des programmes. On essaie d'y décrire un univers (structure de l'abstraction) et des énoncés (opérations sur l'abstraction) ([FIN 79], [DERFI 79]). Nous avons utilisé pour cela, le langage de la logique du premier ordre [ANNEXE 1] (avec l'égalité) car nous pensons que, compte-tenu de la diversité des natures des abstractions relationnelles, une spécification algébrique peut être impossible ou trop difficile à produire.

On trouvera dans [DERFI 79], [FIN 79], [LIV 78], [CHAB 82], [YEH 78], [PARN 72] des éléments concernant les méthodes de spécification de programmes .

L'intérêt de ces abstractions est multiple :

- d'une part, ils permettent de ne pas déformer la perception qu'ont certains utilisateurs de "leur monde observé".
- d'autre part, ils peuvent amener à découvrir des insuffisances dans la conceptualisation (Exemple : La cellule familiale aurait permis de montrer la nécessité de disposer de l'association MARIAGE si celle-ci n'avait pas été initialement explicitée).
- enfin, ils aident à faire apparaître des opérations que l'on pourra rattacher à certains types.
  Exemple : La spécification de AINE impose de disposer de l'opération MAX sur les AGEs.

Dans le paragraphe qui suit, nous allons montrer comment utiliser les concepts que l'on vient de définir, pour spécifier une base abstraite de données.

II - LA SPECIFICATION D'UNE BASE ABSTRAITE : LE TYPE C-SCHEMA

II.1.- Remarques préliminaires

La spécification du type qui fait l'objet de ce paragraphe, a pour but de caractériser une base abstraite. Elle essaie de présenter, de façon statique, l'activité d'élaboration d'une base abstraite.

Elle suppose, néanmoins, que les entités et les associations ont été préalablement définies (ie spécifiées) et permet de les utiliser pour spécifier un schéma.

On notera, au vu de la spécification du C-SCHEMA, que l'on dispose d'une opération ADDOP qui permet d'introduire les opérations qui portent sur la base. ADDOP a comme éléments de son profil SYMBOP (symbole d'opération) et PROFIL (domaine(s) et codomaine(s) de l'opération). Nous demandons au lecteur de bien vouloir se contenter de ces définitions intuitives de SYMBOP et PROFIL

car les définir formellement (en spécifiant, par exemple, un méta-type OPERATION) nous éloignerait de nos propos.

Le type C-SCHEMA permet de spécifier une base abstraite comme un ensemble de spécifications d'entités (DEFENT) et d'associations (DEFASS). Pour une raison identique à celle invoquée pour ADDOP, nous ne donnerons pas de spécification formelle de DEFENT ou de DEFASS. Intuitivement, on considèrera que :

a) DEFENT est la spécification d'une ENTITE ; elle comporte

- un symbole d'ENTITE (de la sorte IDENT)
- les symboles des attributs de l'entité et les symboles de leurs types respectifs (spécification de la O-ENTITE)
- les symboles d'opérations et leur profil
- les axiomes
- l'indication de l'attribut CLE de l'ENTITE.

b) DEFASS est celle d'une ASSOCIATION (ASSBIN) ; elle comporte

- un symbole d'ASSOCIATION (de la sorte IDASS)
- les symboles des entités qu'elle relie (O-ASSBIN)
- les symboles des fonctions inverses l'une de l'autre qui la définissent (VALF et INVERSF)
- l'indication des cardinalités.

Par ailleurs, la spécification du type C-SCHEMA utilisera :

- les opérations ensvide et insérer supposées définies sur un type ENSEMBLE (elles permettent respectivement d'instancier un ensemble vide et d'ajouter un élément à un ensemble).
- les opérations symbrel et symbent que nous supposons également définies sur DEFASS et DEFENT, respectivement. (symbrel permet d'obtenir le symbole d'association et symbent celui d'une ENTITE).

Exemple :

Soit type PERSONNE = ENTITE(NOM:CHAINE,PRENOM:CHAINE)
et type PATERNITE = ASSBIN(PERSONNE,ENFANT,(0,n),(1,1))

Les types PERSONNE et PATERNITE (avec éventuellement, le corps de leur spécification) sont respectivement de type DEFENT et DEFASS.

L'application de symbent sur le premier fournira PERSONNE comme résultat, celle de symbrel sur le second fournira PATERNITE.

symbent (type PERSONNE=...) = PERSONNE
symbrel (type PATERNITE=...) = PATERNITE

- des relations d'égalité entre des IDENT (symboles d'entités), des IDASS (symboles d'associations) et des SYMBOP (symboles d'opérations). Ces relations ont toutes été notées ≡ dans la spécification du C-SCHEMA. Nous leur donnons une définition plus formelle dans les spécifications qui suivent.

a) le type identificateur

type identificateur
  sortes identificateur,bool
  opérations
    make-id : → identificateur
    egal-id : identificateur×identificateur → bool
  axiomes
    soit a_1,a_2,a_3 : identificateur
    • egal-id(make-id,a_1) = 'Faux'
    • egal-id(a_1,a_2)∧egal-id(a_2,a_3) ⇒ egal-id(a_1,a_3)
    • egal-id(a_1,a_2) ⇒ egal-id(a_2,a_1)
    • egal-id(a_1,a_1) = 'vrai'
ftype

b) Les types IDENT et SYMBOP

Ils sont du type identificateur et de ce fait leur égalité s'exprime en termes d'égalité sur les identificateurs (egal-id).

c) Le type IDASS

Il concerne les symboles d'association. Une association (cf type ASSBIN) est définie par son identificateur, ceux des entités qu'elle relie et ses cardinalités. Pour ne pas encombrer la spécification du type IDASS nous allons délibérément ignoré les cardinalités. Nous utiliserons par ailleurs, l'expression syntaxique rel (ida1, ida2) dans le typeCSCHEMA pour désigner une association r_k entre les entités de nom ida1, ida2. Dans le type IDASS, on désignera r_k par:

$<r_k, ida1, ida2>$

type IDASS
  sortes identificateur,IDASS,BOOL
  opérations
    make-idass : identificateur×identificateur×identificateur → IDASS
    egal-idass : IDASS×IDASS → BOOL
  axiomes
    soit $(r_1,a_1,a_2),(r_2,a_3,a_4),(r_3,a_5,a_6)$:IDASS
    • egal-idass$((r_1,a_1,a_2),(r_1,a_1,a_2))$ = 'vrai'
    • egal-idass$((r_1,a_1,a_2),(r_1,a_2,a_1))$ = 'vrai'
    • egal-idass$((r_1,a_1,a_2),(r_2,a_3,a_4))$
        ⇒ egal-idass$((r_2,a_3,a_4),(r_1,a_1,a_2))$
    • egal-idass$((r_1,a_1,a_2),(r_2,a_3,a_4))$
        ∧ egal-idass$((r_2,a_3,a_4),(r_3,a_5,a_6))$
        ⇒ egal-idass$((r_1,a_1,a_2),(r_3,a_5,a_6))$
    • egal-idass$((r_1,a_1,a_2),(r_2,a_1,a_2))$ = 'Faux'
ftype

II.2.- La spécification du type C-SCHEMA

type C-SCHEMA
  sortes C-SCHEMA,DEFENT,DEFASS,IDENT,IDASS,BOOL,ENSEMBLE( ), SYMBOP,PROFIL
  opérations
    schvide : → C-SCHEMA
    addent : C-SCHEMA×DEFENT → C-SCHEMA
    addass : C-SCHEMA×DEFASS → C-SCHEMA
    addop : C-SCHEMA×SYMBOP×PROFIL → C-SCHEMA
    entit : C-SCHEMA → ENSEMBLE(DEFENT)
    assoc : C-SCHEMA → ENSEMBLE(DEFASS)
    delent : C-SCHEMA×IDENT → C-SCHEMA
    delass : C-SCHEMA×IDASS → C-SCHEMA
    delop : C-SCHEMA×SYMBOP → C-SCHEMA
    oper : C-SCHEMA → ENSEMBLE(SYMBOP×PROFIL)
  axiomes
    soit a_1,a_2,a_3 : DEFENT;O1,O2:SYMBOP;p:PROFIL;s:C-SCHEMA
    ida1,ida2,ida3,ida4:IDENT;rel(ida1,ida2),rel(ida3,ida4):
        DEFASS,r:IDASS
    • entit(schvide( )) = ensvide( )
    • entit(addent(s,a_1))=ajtens(entit(s),symbent(a_1))
    • entit(addass(s,rel(ida1,ida2)))=ajtens(ajtens(entit(s),ida1),ida2)
    • entit(addop(s,O1,p))=entit(s)
    • assoc(schvide( )) = ensvide( )
    • assoc(addent(s,a_1))=assoc(s)
    • assoc(addop(s,O1,p))=assoc(s)
    • assoc(addass(s,rel(ida1,ida2)))=ajtens(assoc(s),symbrel rel(a_1,a_2)))
    • oper(schvide( ))=ensvide( )
    • oper(addent(s,a_1))=oper(s)
    • oper(addass(s,rel(ida1,ida2)))=oper(s)
    • oper(addop(s,o1,p))=ajtens(oper(s),(O1,p))
    • delent(schvide( ),ida1)=schvide( )
    • delent(addent(s,a_1),ida2)=si symbent(a_1)≡ida2 alors delent(s,ida2) sinon addent(s,ida2),a_1)
    • delent(addass(s,rel(ida1,ida2)),ida3)=si ida3≡ida1∨ida3≡ida2 alors delent(s,ida3) sinon addass(delent (s,ida3),rel(ida1,ida2)))
    • delent(addop(s,O1,p),ida1)=delent(s)

- delass(schvide( ),rel(ida1,ida2))=schvide( )
- delass(addent(s,a_1),r)=addent(delass(s,r),a_1)
- delass(addass(s,rel(ida1,ida2)),r))=
  si symbrel(rel(ida1,ida2))≡r
  alors delass(s,r)
  sinon addass(delass(s,r),rel(ida1,ida2))
- delass(addop(s,01,p),r) = delass(s,r)

- delop(schvide( ),01)=schvide( )
- delop(addent(s,a_1),01)=delop(s,01)
- delop(addass(s,rel(ida1,ida2)),01)=delop(s,01)
- delop(addop(s,01,p),02)=si 01≡02
  alors delop(s,02)
  sinon addop(delop(s,02),01,p)

ftype

Ce type se veut être une abstraction de la spécification d'une base abstraite. Dans SINDBAD, nous avons développé un sous-système d'aide à la spécification de base(s) abstraite(s), DEFDB, qui réalise le type SCHEMA. Nous en détaillerons le fonctionnement au cours du chapitre suivant.

Le type C-SCHEMA caractérise une classe de schémas. Un objet de la classe est un schéma abstrait, au sens conventionnel de schéma de base de données. Nous reviendrons sur ce point à la fin de ce chapitre et tout au long du chapitre III de cette partie.

Dans l'immédiat, nous allons donner de sommaires indications, d'ordre méthodologique, qui pourraient aider l'utilisateur à déterminer de façon concrète les entités, les associations et les abstractions relationnelles pour la conceptualisation d'un monde observé.

III - DE L'OBSERVATION D'UN DOMAINE A SON SCHEMA

Les concepts que nous venons de présenter doivent servir à exprimer formellement l'abstraction de l'observation d'un domaine.

Mais comment mener l'observation d'un domaine, d'une part, et, d'autre part, comment arriver à décrire, à l'aide de ces concepts formels, les phénomènes, les faits, les objets... du domaine sachant

qu'ils sont souvent exprimés en termes informels ?

L'objectif de ce paragraphe est d'essayer d'apporter quelques éléments de réponse. Nous y mettrons en évidence deux nouveaux concepts, ceux d'AGREGAT et d'ASSOCIATION, destinés à faire l'objet "d'un usage externe" (niveau utilisateur). Nous montrerons également comment "passer" des AGREGATs et ASSOCIATIONs aux ENTITEs et ASSBINs.

III.1.- Quelques considérations d'ordre méthodologique pour l'observation d'un domaine d'application

Nous avertissons d'emblée le lecteur que le but de ce paragraphe n'est pas de présenter une méthode de conception de bases de données, mais de donner quelques indications qui peuvent guider un utilisateur dans la conduite de l'observation du domaine, tout en lui permettant de faire abstraction, dans un premier temps, des notions formelles que nous avons décrites.

Le lecteur intéressé par plus de détails méthodologiques pourra consulter [SMIT 82], [FLOR 82], [TARD 79 et 83], [FOUC 82], [DUB 84]...).

On s'intéressera, lors de l'observation d'un domaine, à tout ce qui peut le caractériser, comme,

- des objets qui y sont présents

ex : - URBAIN V est un hôtel
- MARGERIDE est une région

- des faits qui peuvent y survenir

ex : M. DUPONT a réservé une chambre à l'hôtel du LION D'OR, pour une durée de trois jours, à partir du 02/04/84.

- la description de faits et/ou d'objets

ex : - en MARGERIDE, on peut faire du ski, se baigner dans des lacs naturels...

- M. DUPONT n'a pas versé d'avance pour sa réservation. Celle-ci ne sera pas effective s'il n'a pas payé la totalité du coût de la location, au plus tard une semaine avant le début du séjour.

- des propriétés de faits et/ou d'objets

ex : - La durée d'un séjour varie en fonction de la saison touristique (haute, basse ou moyenne saison) et de la nature de l'hébergement souhaité (camping, auberge, hôtel). Aussi, toute demande de séjour formulée par un client devra être faite pour une durée qui soit un multiple des durées de séjour ainsi définies.

- La classification de tous les hébergements est faite en termes de catégories exprimées par un nombre d'étoiles.

Pour aboutir à une représentation abstraite d'éléments aussi disparates a priori, nous proposons deux niveaux de représentation :

- le premier est exprimé sous la forme d'un réseau sémantique (cf. § IV.13/II/1).
- le second est le résultat de l'application de deux mécanismes (agrégation et généralisation) sur le réseau sémantique du niveau précédent.

III.1.1.- L'élaboration d'une représentation initiale

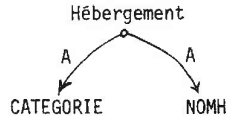
III.1.1.1.- Caractérisation des noeuds et des liens du réseau

a) Un noeud désignera, généralement, une classe d'objets. Il devra, de ce fait, exprimer ce que signifie être membre de la classe. En ce sens, il est une abstraction de points communs aux objets

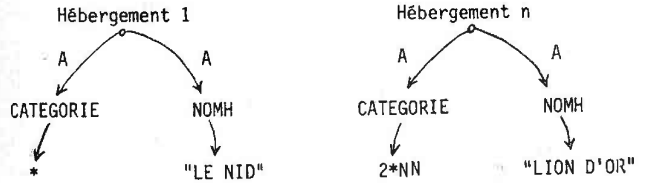
de la classe.

Ces points communs seront définis par des propriétés ou des prédicats représentés par des liens qui émanent du noeud.

Exemple :



Ce réseau est une abstraction d'un ensemble d'objets hébergements.



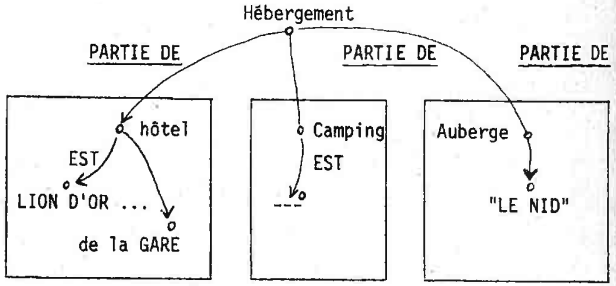
b) Les liens expriment des relations sémantiques entre des noeuds. Ils naissent de l'observation du domaine et peuvent notamment exprimer :

- l'appartenance d'un objet à une classe (EST)

ex : LION D'OR est un nom d'hébergement

- des relations de sous-classe (PARTIE DE)

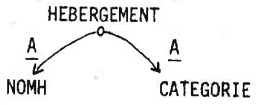
ex : Dans les hébergements, on distingue les campings, les hôtels et les auberges.



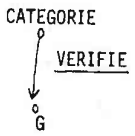
- des propriétés d'un noeud. Certaines le décrivent (exemple(1)), d'autres peuvent exprimer des contraintes (exemple (2)).

Exemples

(1)



(2)

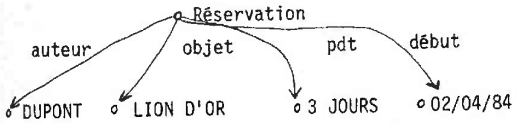


où G est la règle qui définit la syntaxe de CATEGORIE (<G> → '*'|'*'<G>|'*NN'| <entier>'*NN'...)

- ...

Parfois, un ensemble de liens est nécessaire pour décrire un fait ou un objet.

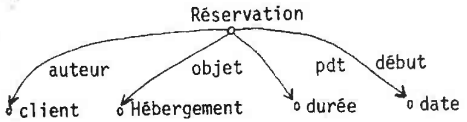
Exemple : - "M. DUPONT a fait une réservation pour trois jours, au LION D'OR, à partir du 02/04/84" pourrait être représenté ainsi :



Ces concepts de noeuds et de liens sont fort utiles pour une première appréhension du problème de la conceptualisation d'un monde observé. Ils constituent un outil simple pour schématiser les phénomènes de ce monde. Néanmoins, la difficulté résidera essentiellement dans le volume des phénomènes à appréhender, ce qui risque de réduire la lisibilité du réseau.

Aussi, généralement, on opèrera rapidement une première opération de généralisation de certaines observations, donnant ainsi d'emblée, une version plus synthétique du réseau initial.

Ainsi, par exemple, au lieu de considérer la réservation de M. DUPONT, on considèrera celle de tout client, que l'on représentera par :



Nous reviendrons sur ce mécanisme au § III.1.2 qui suit.

III.1.1.2.- L'analyse prédicative comme guide pour une représentation initiale

Dans le chapitre I/A, nous avions déjà établi une analogie avec des éléments d'une méthode d'analyse en linguistique ("analyse prédicative") et la sémantique des modèles conceptuels les plus connus dans le domaine des bases de données. Nous allons ici, nous inspirer, une nouvelle fois, de ces éléments pour montrer une façon de déterminer un réseau sémantique initial à partir de l'observation et de la description informelle d'un domaine.

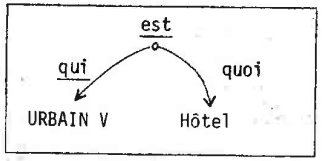
Rappelons que dans l'analyse d'une phrase, la méthode consiste à reconnaître :

- une structure principale (exprimée par un prédicat principal et ses arguments).
- des structures secondaires éventuelles qui "modifient" la structure principale.
  (Ces structures secondaires peuvent être assimilées aux fonctions adverbiales et adjectives présentes dans la phrase).

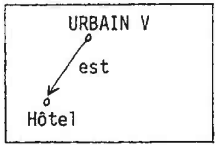
Da façon générale, on pourra représenter une phrase de la façon suivante :

- on associera un noeud (que l'on appellera noeud-concept) au prédicat principal ; à chaque argument seront associés un noeud (dit noeud-objet) et un lien entre le noeud concept et le noeud objet. Ce lien exprime le rôle de l'argument dans le prédicat.
  [Dans certains cas, simples, nous serons amenés à utiliser une forme de représentation "plus dense" (voir exemples qui suivent)].
- A chaque structure secondaire simple seront associés un noeud-objet et un lien-rôle liant le noeud-objet et le noeud-concept. Certaines structures secondaires nécessiteront une représentation plus complexe (un sous-réseau sémantique) ; le dernier de nos exemples illustre ce cas.

Exemple 1 : "URBAIN V est un hôtel" peut être représenté par

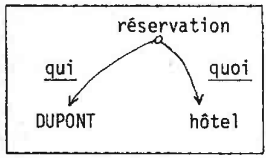


ou sous une forme "plus dense"



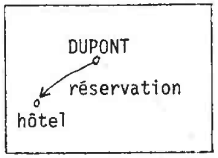
(Le noeud-concept disparaît et on établit un lien entre les arguments à l'aide du prédicat)

Exemple 2 : - "DUPONT a réservé un hôtel"
Comme précédemment, la représentation peut être :



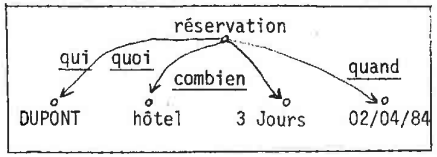
(a)

OU



(b)

Exemple 3 : - "DUPONT a réservé un hôtel pour trois jours à dater du 02/04/84"



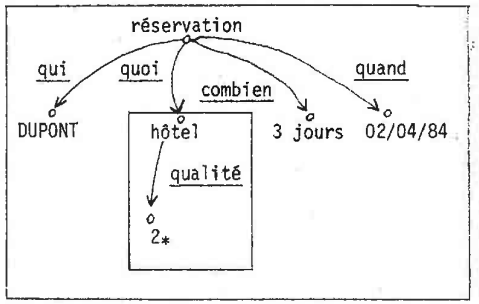
Cet exemple, en liaison avec ceux qui précèdent, permet de montrer l'inconvénient de la représentation "dense". En effet, si dans l'exemple 2, on adopte la représentation (b), l'apparition (éventuellement, à une étape ultérieure) de structures secondaires amènerait à modifier le réseau, alors que dans le cas de la représen-

tation (a), il suffit de l'étendre par la représentation de la/des structure(s) secondaire(s) (exemple 3).

Voyons maintenant un dernier exemple illustrant le cas où une structure secondaire peut donner naissance à un sous-réseau.

Exemple 4 : -"DUPONT a réservé en hôtel 2*, pour 3 jours à partir du 02/04/84".

La caractérisation de l'hôtel (catégorie 2*) détermine un second niveau de lien, comme le montre le schéma de représentation ci-dessous :



L'application de ces quelques mécanismes devrait permettre d'obtenir une représentation initiale d'un domaine. Mais l'on se rend bien compte que celle-ci risque d'être volumineuse, et, par conséquent, perdrait de sa lisibilité. [Des outils similaires à ceux développés dans le cadre de [[TARD 79] et [TARD 83]] seraient alors d'un apport certain].

Les mécanismes d'agrégation et de généralisation (inspirés de [SMIT 77a,b et 82]), que nous allons présenter, vont permettre de "condenser" cette représentation initiale.

III.1.2.- Agrégation et généralisation

III.1.2.1.- Définitions

- La généralisation est un procédé d'abstraction de plusieurs objets, ayant des propriétés communes, en un objet générique (cf. exemple 1).
- L'agrégation est un procédé d'abstraction d'objets et de liens entre ces objets en un objet générique que l'on appellera AGREGAT (cf. exemple 2).

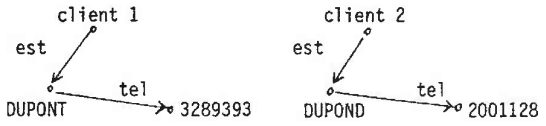
On représentera par un noeud du réseau, le résultat de l'un ou l'autre de ces mécanismes. On dira également qu'un noeud est constituant d'un agrégat A s'il fait partie des noeuds agrégés pour définir A.

III.1.2.2.- Mise en oeuvre de ces mécanismes

Nous allons, à partir d'exemples, montrer comment utiliser ces mécanismes.

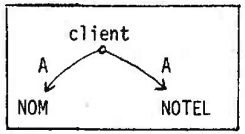
a) Exemple 1 : La généralisation

Soit la représentation initiale :

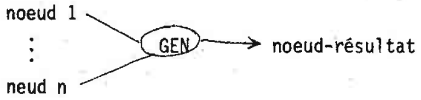


On peut généraliser 3289393 et 2001128 en NOTEL (numéros de téléphone) et DUPONT, DUPOND en NOM.

D'où la nouvelle représentation

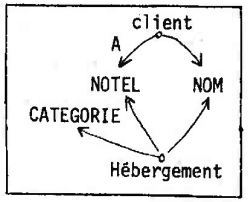


Par la suite, on schématisera une généralisation de la façon suivante :

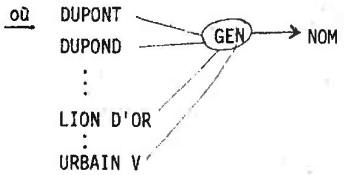


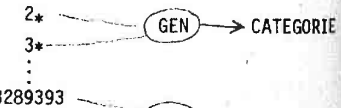
b) Exemple 2 : Agrégation et recouvrement d'agrégats

Soit la représentation :

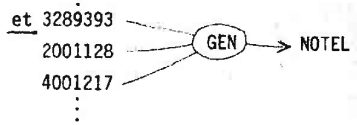


où

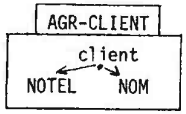




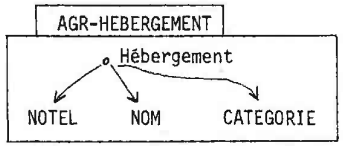
et



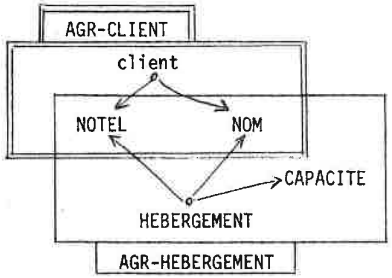
On peut agréger NOTEL et NOM en CLIENT



On peut faire de même pour HEBERGEMENT



Cet exemple montre que certains noeuds peuvent participer à des agrégations différentes. La "fusion" des représentations de AGR-CLIENT et AGR-HEBERGEMENT nous permet de le symboliser.

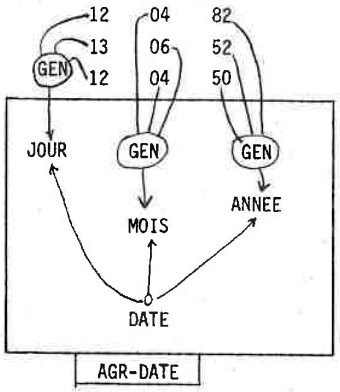


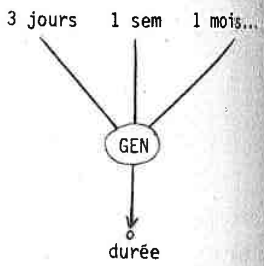
Dans ce cas, on parlera de recouvrement d'agrégations et on dira que des agrégats $A_1,\ldots A_n$ se recouvrent s'ils ont au moins un constituant commun.

Les constituants communs a plusieurs agrégats sont appelés constituants d'intersection

Exemple 3 : Degré d'agrégation

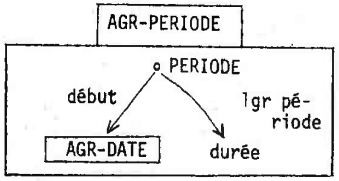
Considérons maintenant le schéma suivant :





Il est issu de l'agrégation de jour, mois, année, qui, eux-mêmes sont nés d'une généralisation, au même titre que durée.

On peut opérer une agrégation (que l'on nommera PERIODE) qui englobera l'agrégat DATE et le noeud durée.



En d'autres termes, un agrégat peut comporter des constituants qui soient eux-mêmes des agrégats.

On appellera degré d'un agrégat A, le nombre d'applications du mécanisme d'agrégation nécessaire pour obtenir A.

On appellera dimension d'un agrégat, le nombre de constituants qu'il comporte.

La mise en oeuvre des mécanismes d'agrégation et de généralisation permet ainsi, d'opérer certaines abstractions. A ce niveau de conception, aucune supposition n'est faite sur la façon dont seront représentés les noeuds obtenus par agrégation ou par généralisation.

La phase suivante, de cet embryon de démarche que nous proposons, va s'intéresser à cet aspect. Elle a pour objectif de déterminer une représentation non redondante, de résoudre le problème de recouvrement d'agrégats... C'est également dans cette phase que se fera "la découverte" des associations.

III.2.- Hiérarchie d'abstraction et construction d'un SCHEMA

A ce niveau de la démarche, nous disposons d'une représentation d'un domaine sous la forme d'agrégats de degrés divers et pouvant, éventuellement, avoir quelques constituants d'intersection.

Dans ce paragraphe, nous allons essayer de montrer comment passer d'une telle représentation à une représentation en termes d'entités et d'associations.

III.2.1.- Le passage aux entités et aux associations

Cette phase vise à fournir une représentation non redondante. On s'intéressera alors :

- à éliminer des homonymies que peuvent occasionner des constituants d'intersection
- à choisir des "entités de base". Intuitivement, une entité est une famille d'objets du monde observé. Formellement, elle sera exprimée ultérieurement à l'aide de O-ENTITE et de ENTITE. Pour ne pas créer de confusion, nous désignerons ces entités par le terme A-entité (Agrégat-entité). Une A-entité "deviendra" une ENTITE par représentation. La fonction de représentation uti-

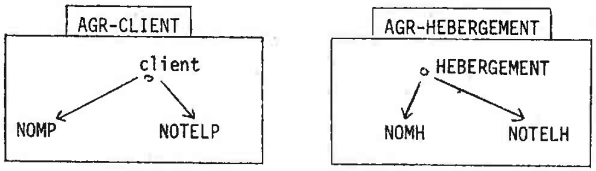
lisée est définie au § III.2.2.

a) Elimination des homonymies

Nous avons vu que certains noeuds pouvaient être généralisés en un même noeud générique qui peut participer à des agrégations différentes. Néanmoins, dans certains cas, il peut être nécessaire, en fonction de la "sémantique" des agrégats construits, de différencier les noeuds (ou une partie des noeuds) qui ont fait l'objet d'une généralisation et opérer ainsi, plusieurs généralisations, au lieu d'une seule.

Ainsi, sur l'exemple 2 du paragraphe précédent, on distinguerait les NOTEL et NOM des CLIENTS de ceux de HEBERGEMENT. (Conceptuellement, on établit des relations de sous-classe entre les classes NOTEL et NOM et, NOTEL, NOM de CLIENT et NOTEL et NOM de HEBERGEMENT).

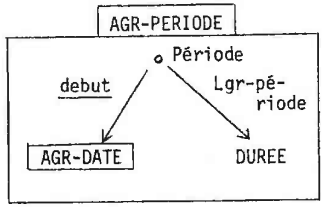
Ainsi l'exemple 2 serait schématisé comme suit



b) Le choix des A-entités

Le choix des entités vise à passer d'une représentation en termes d'agrégats à une représentation en termes de A-entités. La détermination de ces dernières est, généralement, guidée par l'intuition ; elle est, comme l'affirment J.M. & D.C.P. SMITH et TARDIEU, notamment, un choix du concepteur.

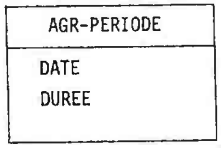
Exemple : Considérons l'agrégat PERIODE obtenue par l'agrégation de l'agrégat DATE et du constituant DUREE



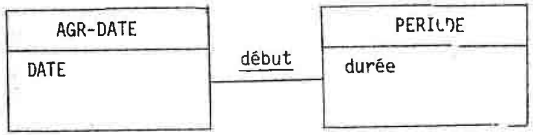
Deux choix sont possibles :

1.- on décide d'associer une A-entité à l'agrégat période

Dans ce cas, l'agrégat DATE sera un constituant de PERIODE au même titre que DUREE ; ce que nous chématisons par



2.- On peut décider, également, d'associer une A-entité à l'agrégat DATE. Dans ce cas, on devra expliciter la nature du lien début entre PERIODE et DATE



- Quoiqu'intuitif, le choix des A-entités devra, néanmoins, aboutir à des entités qui vérifient la propriété suivante :

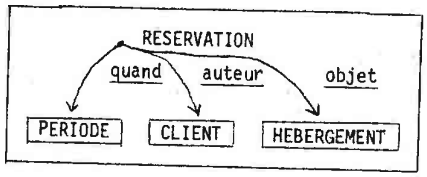
Pi : Toute instance de A-entité (ie tout objet de la class

qu'elle détermine) est constituée par une seule instance de chacun de ses constituants.

ex : si HEBERGEMENT agrège NOMH et CATEGORIE, toute instance de HEBERGEMENT est constituée par un couple (NOMH, CATEGORIE), comme par exemple (LE PIGEONNIER,*), (LE NID, 2*NN)...
HEBERGEMENT est dit bien formé selon P1.

- Sur un autre plan, on remarquera que la décision prise à propos de la représentation d'un agrégat A pourra avoir une incidence sur la représentation d'autres agrégats.

exemple : Soit l'agrégat réservation qui agrège PERIODE, CLIENT et HEBERGEMENT



Le choix qui sera fait pour représenter l'agrégat PERIODE aura de façon évidente une incidence sur la représentation de RESERVATION. (Nous reviendrons plus en détail sur la représentation de RESERVATION dans quelques instants).

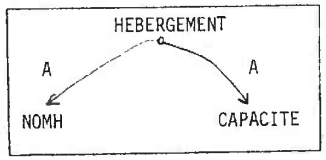
- De façon générale, intuitivement, on peut dire que le choix des A-entités vise à déterminer deux types de liens.

1.- des liens attributs

Ce sont les liens qui sont "absorbés" par la représentation d'un agrégat par une A-entité. Ils relient l'agrégat à ses constituants.

Exemple :

La représentation de l'agrégat HEBERGEMENT par une A-entité HEBERGEMENT détermine les liens-attributs



A entre HEBERGEMENT et NOMH
et HEBERGEMENT et CAPACITE

2.- des liens association

Ce sont les liens qui persistent sur le schéma après le choix des A-entités. Ils expriment le plus souvent des liaisons inter-agrégats.

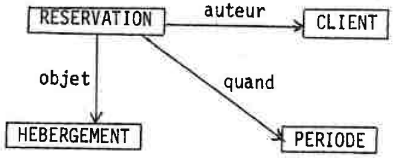
Exemple : Reprenons l'exemple de l'agrégat RESERVATION et essayons de voir par son biais quelques solutions de représentation.

Supposons HEBERGEMENT et CLIENT représentés par des A-entités.

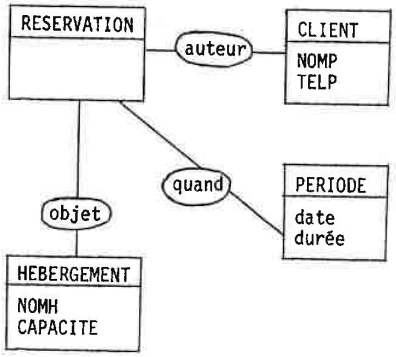
Cas 1 : Représentation de PERIODE par une A-entité

RESERVATION peut être représenté selon deux cas de figure.

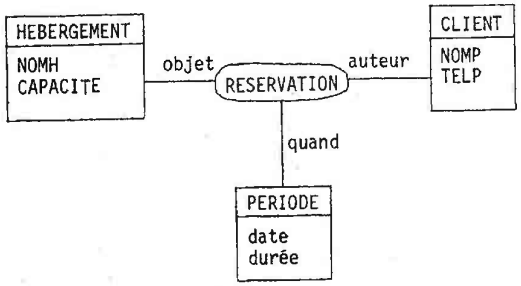
a) par une A-entité



On aura ainsi trois liens associations explicitant les liens entre RESERVATION et ses constituants. Selon notre graphisme Entité/association, la représentation de RESERVATION est donc



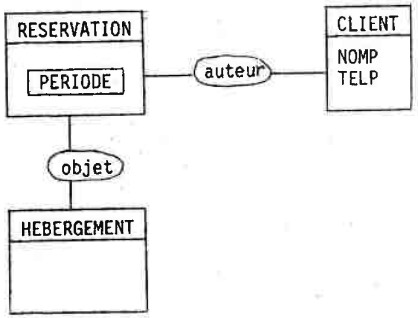
b) par un lien association



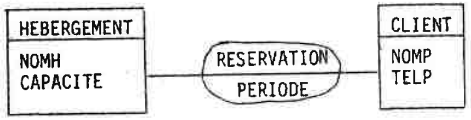
Cas 2 : PERIODE n'est pas représentée par une A-entité

Dans ce cas, quelque soit la solution adoptée pour RESERVATION, (ie association ou entité), un lien attribut le reliera à PERIODE ; soit schématiquement :

(a)



(b)



Ces exemples ont voulu illustrer les choix à faire pour transformer une représentation en termes d'agrégats en une représentation en termes de A-entités et d'associations.

La présentation de la démarche a été délibérément faite de façon informelle. Dans le paragraphe qui suit, nous allons revenir à un niveau plus formel pour exprimer cette transformation de représentation.

III.2.2.- Spécification formelle du schéma

III.2.2.1.- Les types AGREGAT et ASSOCIATION

Les concepts de A-ENTITE et d'association présentés au cours du paragraphe précédent permettent d'exprimer une abstraction d'un monde observé. Ils peuvent être formalisés par les types suivants :

type AGREGAT(ATTR1,...,ATTRn)
  sortes AGREGAT, ATTR1,...,ATTRn,BOOL
  opérations
    faire-AGR : ATTR1×...×ATTRn → AGREGAT
    f ATTR1 : AGREGAT → ATTR1
    f ATTR2 : AGREGAT → ATTR2
      ⋮
    f ATTRn : AGREGAT → ATTRn
    Egal-AGR : AGREGAT×AGREGAT → Bool
  axiomes
    soit C_1: ATTR1,...,c_n : ATTRn
         a: AGREGAT
    - $fATTR_i$ (faire-AGR(C_1,...,C_n)) = C_i
    - Egal-AGR(faire-AGR(C_1,...,C_n),a) =
        si $Egal_{ATTR1}$(C_1,fATTR1(a))
        ∧ ...
        ∧ $Egal_{ATTRn}$(C_n,fATTRn(a))
        alors vrai sinon faux
ftype

Ce type AGREGAT spécifie la notion de A-entité. Nous utiliserons désormais le terme AGREGAT au lieu de A-entité, ce dernier ayant été introduit au paragraphe précédent pour ne pas créer de confusion avec le type ENTITE ou avec l'agrégat résultant d'une agrégation.

Pour les associations, on ne s'intéressera qu'à celle d'ordre deux.

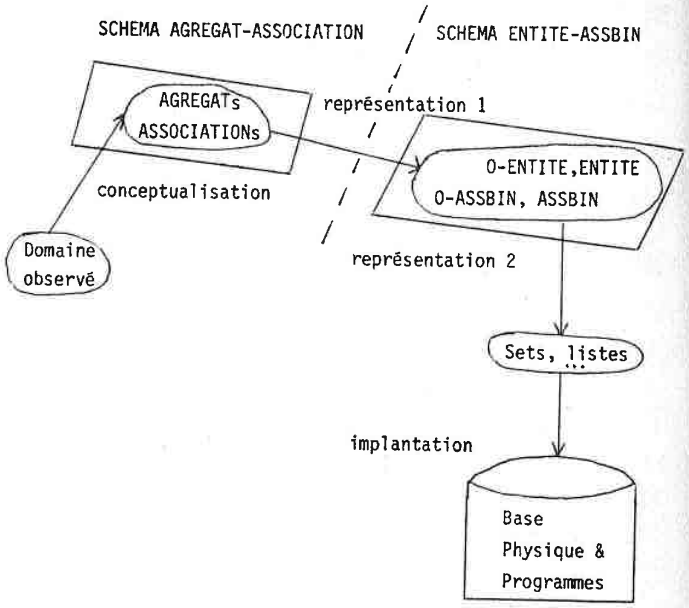
type ASSOCIATION(AGREGAT, AGREGAT,ATTR1,...,ATTRm)
  sortes ASSOCIATION, AGREGAT, BOOL
  opérations
    faire-assoc : AGREGAT×AGREGAT×ATTR1×...×ATTRm → ASSOCIATION
    proj1 : ASSOCIATION → AGREGAT
    proj2 : ASSOCIATION → AGREGAT
    Egal-assoc : ASSOCIATION×ASSOCIATION → BOOL
    f-ATTR1 : ASSOCIATION → ATTR1
      ⋮
    f-ATTRm : ASSOCIATION → ATTRm
  axiomes
    soit a_1,a_2:AGREGAT ; r:ASSOCIATION
         c_1:ATTR1,...,c_m:ATTRm
    - proj1(faire-assoc(a_1,a_2,c_1,c_2,...,c_m)) = a_1
    - proj2(faire-assoc(a_1,a_2,c_1,c_2,...,c_m)) = a_m
    - Egal-assoc(faire-assoc(a_1,a_2,c_1,...,c_m),r) =
        si Egal-AGR(a_1,proj1(r))∧Egal-AGR(a_2,proj2(r))
        ∧ Egal-ATTR1(c_1,f-ATTR1(r))∧...∧Egal-ATTR$_m$(c_m,f-ATTRm(r))
        alors vrai sinon faux
    - f-ATTR1(faire-assoc(a_1,a_2,c_1,...,c_m)) = c_1
      ⋮
    - f-ATTRm(faire-assoc(a_1,a_2,c_1,...,c_m)) = c_m
ftype

III.2.2.2.- Hiérarchie d'abstraction

L'idée que nous avons poursuivie jusqu'ici, est de définir des concepts à usage externe afin de "cacher" l'aspect formel des types ENTITE, ASSBIN, SCHEMA.

Nous proposons, alors, de ne rendre disponibles à un utilisateur, que les concepts d'AGREGAT et d'ASSOCIATION. Les ENTITE et ASSBIN constitueront des concepts qui représenteront AGREGAT et ASSOCIATION respectivement.

Ainsi schématiquement, nous pouvons voir cela de la façon suivante :



Cette approche présente selon nous un double avantage

(1) Celui de "cacher" certains aspects formels (rebutants pour nombre d'utilisateurs).

(2) et celui de retarder certains choix de représentation.

Par exemple, le fait qu'une DATE soit exprimée par un triplet d'entiers est pour nous une décision de représentation. De même, le fait qu'un NOM "soit de type" STRING, ou que l'égalité entre deux entités s'exprime en termes d'égalité de leur CLE... l'est aussi.

La notion d'agrégat permet ainsi de s'abstraire de ces considérations de représentation. Nous en tiendrons compte lors de l'expression du schéma AGREGAT-ASSOCIATION en SCHEMA ENTITE-ASSBIN.

Formellement, on représentera un AGREGAT par une O-ENTITE (puis une ENTITE), et une ASSOCIATION par une O-ASSBIN (puis une ASSBIN). La fonction de représentation utilisée est définie ainsi :

a) Représentation d'AGREGAT par O-ENTITE

Rappelons qu'une O-ENTITE est spécifiée par :

type O-ENTITE(C_1:T_1,...,C_n:T_n)
  sortes O-ENTITE,T_1,...,T_n,BOOL
  opérations
    C_1:O-ENTITE → T_1
    C_2:O-ENTITE → T_2
    ⋮
    C_n:O-ENTITE → T_n
    Faire-OENT:T_1×T_2×...×T_n → O-ENTITE
    Egal-OENT:O-ENTITE×O-ENTITE → BOOL
  axiomes
    {cf § I de ce chapitre}
ftype

La fonction de représentation (rep) d'AGREGAT par O-ENTITE est définie par :

- rep (ATTR1) = T_1
  ⋮
- rep (ATTRn) = T_n
- rep(fATTR1) = C_1
  ⋮
- rep(fATTRn) = C_n
- rep(faire-AGR) = Faire-OENT
- rep(Egal-AGR) = Egal-OENT

Formellement, on devrait prouver la correction de la représentation et démontrer que les axiomes de AGREGAT sont des théorèmes de O-ENTITE ([DER & FI 79], chapitre II, § 2.3).

Nous nous contenterons ici d'être "intimement convaincu" qu'elle est correcte.

b) Représentation de ASSOCIATION par O-ASSBIN

Rappelons également la spécification de O-ASSBIN

type O-ASSBIN(f_1:O-ENTITE,f_2:O-ENTITE,C_1:T_1,...,C_m:T_m)
sortes O-ASSBIN,O-ENTITE,BOOL,T_1,...,T_m
opérations
Faire-OASSBIN : O-ENTITE×O-ENTITE×T_1×...×T_m → O-ASSBIN
Egal-Orel : O-ASSBIN×O-ASSBIN → BOOL
f_1 : O-ASSBIN → O-ENTITE
f_2 : O-ASSBIN → O-ENTITE
C_1 : O-ASSBIN → T_1
⋮
C_n : O-ASSBIN → T_m
axiomes
{se référer à la spécification de O-ASSBIN}
ftype

De façon analogue à la représentation de AGREGAT, on définit la représentation de ASSOCIATION par :

- rep(AGREGAT) = O-ENTITE
- rep(ATTR1) = T_1
- ...
- rep(ATTRm) = T_m
- rep(faire-assoc) = Faire-OASSBIN
- rep(Egal-assoc) = Egal-orel
- rep(proj1) = f_1
- rep(proj2) = f_2
- rep(fATTR1) = C_1
- ...
- rep(fATTRm) = C_m

Encore une fois, en toute rigueur, on devrait prouver la correction de la représentation. Le problème dans le cas de ASSOCIATION est plus ardu dans la mesure où ASSOCIATION est un type paramétré par un type paramétré (AGREGAT).

Encore une fois, nous demandons au lecteur de se laisser persuader par notre intime conviction de la correction de la représentation.

On trouvera dans [DER & FI 79], chap. II, § 2.3, une présentation détaillée du problème de la représentation d'un type abstrait spécifié algébriquement .

III.3.- Conclusion

Nous achevons ainsi cette large parenthèse relative à des aspects méthodologiques.

Nous avons essayé de montrer comment l'on pouvait progressivement mener une démarche de conceptualisation, en se détachant des concepts formels qui aideront à exprimer le résultat de la démarche. Nous avons, également, essayé de dégager "des concepts à usage externe" dont l'intérêt est de retarder des choix de re-

présentation.

En outre, nous avons montré comment ces concepts pouvaient être représentés en termes de nos concepts de base (O-ENTITE et O-ASSBIN), l'idée étant d'autoriser l'utilisateur à faire usage de ces concepts pour exprimer ses besoins (requêtes).

[Enfin, dans ce qui suivra, il nous arrivera de confondre AGREGAT, O-ENTITE et ENTITE, ASSOCIATION, O-ASSBIN et ASSBIN quand cela ne peut donner lieu à aucune ambiguïté dans leur interprétation].

IV - INSTANCE et "VARIABLE BASE DE DONNEES"

La démarche, précédemment préconisée, permet la mise en évidence progressive des agrégats, attributs et associations qui définissent une abstraction d'un monde observé. Les concepts d'AGREGAT, d'ASSOCIATION et d'ATTRIBUT sont "à usage externe" au sens où ils permettent à un utilisateur du système de s'abstraire de nombre de considérations de représentation d'une part, et de certains aspects formels, d'autre part.

Afin de montrer la différence entre notre approche et une approche traditionnelle, nous allons dans un premier temps, présenter deux schémas (qui "valent mieux qu'un long discours") où nous faisons figurer ce que voit et ce qu'utilise un utilisateur d'une base de données.

Dans un deuxième temps, nous définirons les notions d'instance et de "variable base de données". Elles nous permettront alors, de mieux exprimer la puissance de l'utilisation des mécanismes d'abstraction pour spécifier et réaliser des bases de données. Nous situerons en outre, le niveau d'intervention de chacun des outils logiciels que nous avons développés ou utilisés.

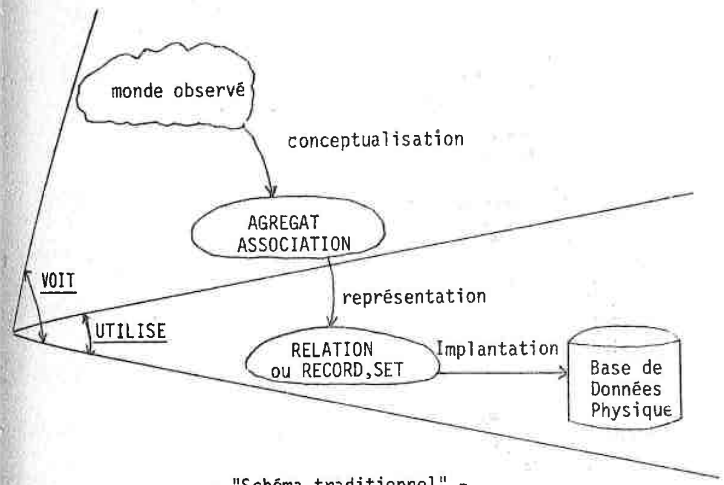
IV.1.- Notre approche vs l'approche traditionnelle

a) "Approche traditionnelle"

Le schéma conceptuel (ou l'abstraction du domaine observé) est exprimé dans un formalisme qui peut être différent de celui du SGBD qui va servir à réaliser une base de données. De ce fait, il sera traduit en termes du SGBD avant que celle-ci soit concrétisée.

En d'autres termes, si le schéma est exprimé en termes d'AGREGAT et d'ASSOCIATION et que le SGBD est de type relationnel (ou CODASYL), on devra exprimer les AGREGATS et les ASSOCIATIONS en termes de RELATIONs (ou de RECORDs et SET).

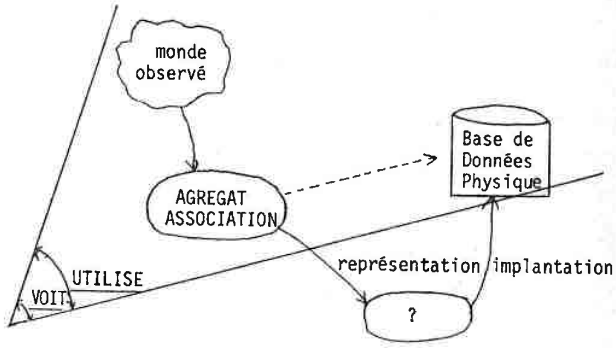
Les utilisateurs de la base voient alors le schéma conceptuel, sa description dans le formalisme du SGBD et la base de données, mais ils n'utilisent que les termes du SGBD pour opérer sur leur base.



- "Schéma traditionnel" -

b) Notre approche

L'objectif poursuivi par l'usage des mécanismes d'abstraction est de cacher le niveau représentation à un utilisateur de la base, en lui donnant les moyens (langages et outils logiciels) d'exprimer ses besoins au niveau le plus abstrait qu'est le schéma conceptuel.



Nous verrons, à l'issue du paragraphe qui suit que ce schéma sera modifié pour expliciter davantage la puissance des mécanismes d'abstraction.

IV.2.- Instance et "variable base de données"

La spécification d'une base résulte "de la fusion" des spécifications de ses constituants (agrégats/entités, associations, attributs...). Elle constitue une caractérisation statique d'une classe "d'objets base de données".

Si nous reprenons les termes de la spécification du méta-système de données faite au § II du chapitre II de la partie B, la spécification d'une base caractérise la sorte ETAT ; Un objet de cette sorte correspond à une instance de base de données dont les

images (états) successives sont obtenues par l'application des opérations de modification (de la sorte Opmod) ou examinées par l'application des opérations d'observation (de la sorte Opsor) que comporte la spécification.

Conceptuellement,

- les objets de la sorte C-SCHEMA sont des spécifications abstraites de bases de données (ou SCHEMAs conceptuels, au sens classique du terme, mais "typés").
- une instance de base de données est alors un objet du type SCHEMA que l'on peut désigner par un identificateur ou variable du type.

Une spécification abstraite $\mathcal{S}$ d'une base peut ainsi être vue comme la donnée de (S,σ,A) où

- S est l'ensemble des symboles de $\mathcal{S}$ (symboles d'attributs, d'entités, d'associations).
- σ est l'ensemble des opérations et de leur profil [σ peut être partitionné en deux sous-ensembles : celui des opérations qui examine les états d'une base (opsor) et celui des opérations qui modifient un état (opmod)].
- A est un ensemble d'axiomes caractérisant
  - les attributs,
  - les entités,
  - les associations,
  - et les opérations de $\mathcal{S}$.

type $\mathcal{S}$
  opérations
    initbd : $\rightarrow \mathcal{S}$
    opmod : $\mathcal{S} \times$ Modif $\rightarrow \mathcal{S}$
    opsor : $\mathcal{S} \times$ SORTIE $\rightarrow$ sortie
  axiomes
    {A}
ftype

Dans une approche algébrique, on considèrerait une instance d'une base de données (un objet de la sorte $\mathcal{S}$) comme le résultat de l'application d'une succession d'opérations sur une instance initiale (produite par initbd).

Par exemple,

AJTSITUATION(AJTSITUATION(AJTPAYS(AJTPAYS(AJTHEB(AJTHEB (INITBD( ),H1),H2),P1),P2),(P1,H1)),(P2,H2))

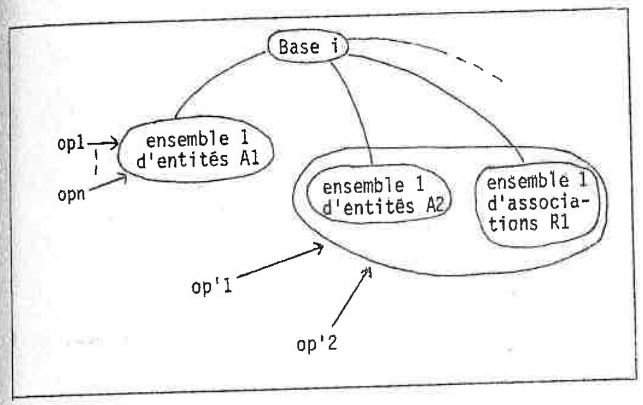
est la base constituée par les ensembles

- {H1,H2} d'HEBERGEMENTS
- {P1,P2} de PAYS
- {(P1,H1),(P2,H2)} d'associationsentre P1 et H1,et P2 et H2, obtenue par une succession d'ajouts d'occurrences à la base vide obtenue par INITBD().

Dans une approche axiomatique (pré/post), on associe à $\mathcal{S}$ une structure de représentation support et à ses opérations des procédures qui agiront sur cette structure [FIN 79]. L'intérêt d'un système de types abstraits est alors de "cacher" cette structure à l'utilisateur afin de lui offrir des possibilités pour opérer sur la structure abstraite (celle déterminée par $\mathcal{S}$).

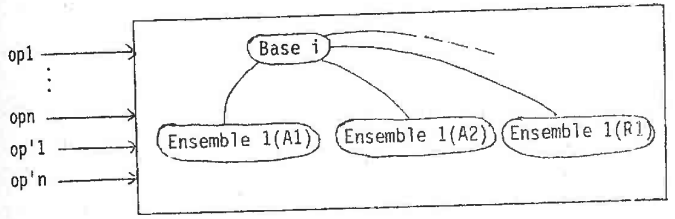
Schématiquement on aurait :

- une utilisation "interne" de la base en considérant l'application de chaque opération sur le sous-ensemble de la base sur lequel elle opère (ce sous-ensemble est déterminé par le profil de l'opération).



[op1,...,opn opèrent sur l'ensemble 1 d'entités A1
op'1,op'2 opèrent sur l'ensemble 1 d'entités A2 et les associations R1]

- une utilisation "externe" en considérent la base comme une boîte noire sur laquelle s'appliquent les opérations



Le niveau "interne" devient dans la terminologie des types abstraits un niveau de réalisation où :

- les constituants de la base (agrégats et associations) sont exprimés en termes de relations dans un système relationnel, de records et sets dans un système CODASYL...).

- les opérations sont exprimées à l'aide du LMD du logiciel en question.

Exemple : Considérons l'agrégat HEBERGEMENT = AGREGAT(NOMH, CAPACITE, CATEGORIE) et les opérations CAPH et AJOUTH qui permettent respectivement d'accéder à la capacité d'un hébergement et d'adjoindre un hébergement à une collection d'hébergements.

La représentation relationnelle de cet agrégat serait :

HEBERGEMENT = RELATION(NOMH, CAPACITE, CATEGORIE)

et ses opérations s'exprimeraient par

SELECT CAPACITE FROM HEBERGEMENT WHERE NOMH = ⟶ , pour CAPH

et INSERT (n,C_1,C_2) INTO HEBERGEMENT pour AJOUTH (et où n,C_1,C_2 désignent respectivement un nom d'hébergement, une capacité et une catégorie).

L'inclusion de l'agrégat HEBERGEMENT dans la spécification d'une base aura pour effet d'y introduire également les opérations qui lui sont associées.

L'application de celles-ci se fera alors sur une instance de la base, c'est-à-dire sur un ensemble d'occurrences présentes à un moment donné.

D'un point de vue formel, cela amène à reconsidérer les profils des opérations de façon à ce qu'elles soient définies sur la base. On procèdera également à un renommage de ces opérations.

Exemple : CAPH et AJOUTH qui avaient comme profil :

CAPH : HEBERGEMENT → CAPACITE

AJOUTH : ENS(HEBERGEMENT)×HEBERGEMENT → ENS(HEBERGEMENT)

deviennent respectivement :

CAPHEB : $\mathcal{S}$ ×HEBERGEMENT → CAPACITE

AJOUTHEB : $\mathcal{S}$ ×ENS(HEBERGEMENT)×HEBERGEMENT → $\mathcal{S}$

La mise en oeuvre de ce mécanisme (renommage et modification de profils) permet donc de rendre disponibles, en tant qu'opérations de $\mathcal{S}$, des opérations définies sur des types inclus dans $\mathcal{S}$.

Dans ce qui suit, nous nous attachons à détailler le processus de représentation d'une base, étant donnée sa spécification $\mathcal{S}$.

REMARQUES

1 - Rien ne s'oppose a priori à ce que l'on ait à tout moment plusieurs instances (repérées par des variables différentes) de base de données spécifiée par $\mathcal{S}$. Sur ces instances pourraient être définies des opérations qui auraient ainsi comme arguments des bases de données. On pourrait définir ainsi :

- l'union de deux bases :

  UNION : $\mathcal{S} \times \mathcal{S} \rightarrow \mathcal{S}$

- l'intersection :

  INTER : $\mathcal{S} \times \mathcal{S} \rightarrow \mathcal{S}$

- l'égalité :

  EGAL : $\mathcal{S} \times \mathcal{S} \rightarrow$ BOOL

- ...

2 - D'autre part, pour un même SCHEMA $\mathcal{S}$, on pourrait définir diverses représentations abstraites, puis associer à chacune d'elles une représentation concrète qui caractérise les bases de données physiques. Celles-ci auront comme structure, la structure décrite par la représentation concrète.

Le schéma suivant permet de mieux exprimer ces remarques. Il résume les différentes étapes nécessaires pour aboutir à des bases de données physiques. Il situe également les niveaux d'utilisation

- des concepts (formels et informels) que nous avons définis.
- et des outils logiciels que nous avons développés ou dont nous avons fait usage.

(*) Le sous-système de compréhension de requêtes agit à ces niveaux.
(**) Le traducteur des requêtes complétées intervient à ce niveau.

V - CONCLUSION

Ce schéma illustre bien la puissance de "l'outil abstraction". Pour une spécification $\mathcal{S}$ on peut choisir plusieurs structures supports de la représentation, et pour une représentation donnée, on peut concrétiser plusieurs bases de données conformes à cette représentation (ie qui en sont des modèles).

Cette approche permet donc de faire des "économies" de spécification. Mais est-elle intégralement applicable dans un contexte réel d'application ? Est-il souvent nécessaire de disposer de plusieurs bases physiques représentant un même domaine ?

Généralement non, mais on peut imaginer des utilisations réalistes, tant en phase de conception qu'en phase d'utilisation.

En phase d'utilisation, pour des contraintes de sécurité de fonctionnement (reprise en cas d'incident), on mémorise les états de la base à des instants donnés de son existence. (On peut dire que l'on conserve des valeurs de la variable base de données).
Dans certains cas, une même spécification peut être une abstraction de plusieurs domaines "similaires". Des bases de données physiques conformes à cette spécification représenteront chacun des domaines...

En phase de conception, avant la mise en place effective d'une base, la possibilité que l'on a de définir plusieurs structures de représentation, sans que cela ait d'impact sur les requêtes (puisqu'elles sont exprimées sur la structure abstraite) pourrait aider à choisir la "meilleure" structure de représentation. En effet, on pourrait "faire varier" la structure de représentation et mesurer les performances de la base en fonction de chacune de ces structures en considérant un "sous-ensemble significatif" de données du domaine de l'application. La représentation "la plus performante" serait alors choisie pour réaliser la base de données physique du domaine ...

Par ailleurs, si l'on définit l'évolutivité d'un système de données comme étant son aptitude à s'accommoder de changements dans les structures qu'il gère (modification de représentations, ajout de nouvelles structures,...) sans incidence sur les programmes existants, il nous semble alors, que l'indépendance des requêtes vis-à-vis des choix de réalisation de la base est un facteur qui devrait favoriser l'évolutivité de notre système.

Dans les chapitres qui suivent, nous revenons à un point de vue plus concret. Nous nous intéresserons plus précisément à la définition d'une structure de représentation, à la concrétisation d'une base physique et à son utilisation.

CHAPITRE II : APPORTS DE SINDBAD POUR LA SPÉCIFICATION D'UNE BASE ABSTRAITE DE DONNÉES

Afin d'aider l'utilisateur lors de la spécification d'une base, nous avons réalisé deux outils fonctionnant de manière interactive .

L'un, ANALTYP (cf. § I), permet d'analyser et de mettre en bibliothèque des spécifications d'agrégats, d'entités, de types de base ...

L'autre, DEFDB (cf. § II), aide l'utilisateur à "construire" une spécification de base abstraite en utilisant la/les bibliothèques créées par ANALTYP et des informations supplémentaires que lui fournit l'utilisateur.

I - L'ANALYSEUR DE SPECIFICATIONS ANALTYP

Le but de ANALTYP est de constituer un environnement de spécification contenant les spécifications des agrégats, attributs et types de base nécessitées par l'application.

On notera qu'il ne se charge pas de vérifier des propriétés des spécifications qui lui sont fournies (complétude, consistance...). Nous supposons que cela a été préalablement fait à l'aide de systèmes tels que VEGA [CHAB 82] ou REVE [LESC 83]...

I.1.- Principe de fonctionnement

Dans la version actuelle, les spécifications en entrée de ANALTYP sont supposées être dans un fichier (source) créé à l'aide d'un éditeur de textes. (La possibilité de les introduire de façon interactive est prévue mais n'est pas entièrement implantée).

Dans le fichier source, chaque spécification a la structure suivante :

```
TYPE <nom type>
   SORTES <liste symboles de sortes>
  [OPERATIONS
        <symbole d'opérations> : <profil>
   AXIOMES
        <partie gauche> = <partie droite>]
FTYPE
```

[La syntaxe du langage de spécification est donnée dans l'ANNEXE III].

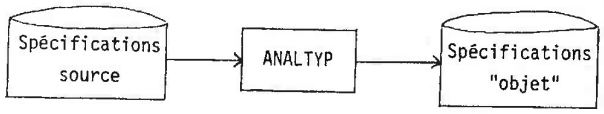
ANALTYP offre la possibilité :

- de créer une bibliothèque de spécification
- d'en étendre une déjà existante.

Il permet également d'effectuer des analyses sélectives de spécification. En effet, il autorise l'utilisateur à ne demander l'analyse que d'un seul ou de quelques spécifications d'un fichier source. Evidemment, la demande d'analyse de la totalité du fichier est également permise.

Le résultat de ANALTYP est une "bibliothèque-objet". Elle comporte, sous une forme de représentation interne (cf. ANNEXE III) :

- les symboles d'attributs, de types de base, d'agrégats
- les symboles de leurs opérations respectives
- la définition des profils de ces dernières
- les axiomes associés.



Le schéma du dialogue qui s'instaure entre un utilisateur et ANALTYP est présenté au paragraphe suivant.

I.2.- Schéma de fonctionnement

ANALTYP opère selon le schéma ci-dessous, où ← symbolise l'émission d'un message par ANALTYP, et → la réponse utilisateur :

```
← NOM DE LA BIBLIOTHEQUE DE TYPE ?
→ <nom>
← ENTREE DES SPECIFICATIONS A PARTIR
        - DE LA CONSOLE (non implémenté)(C)
        - D'UN FICHIER CONTENANT LES SPECIFICATIONS SOURCE (D)
→ D
← NOM DU FICHIER ?
→ <NOMFICH>
← ANALYSE DE TOUTES LES SPECIFICATIONS DE <NOMFICH> OU DE
        QUELQUES SPECIFICATIONS ?
→ TOUTES|UNE
```

Si l'on désire analyser toutes les spécifications source (option TOUTES) ANALTYP fera "défiler" tout le fichier les contenant, sinon il opèrera sous la conduite de l'utilisateur pour faire une analyse sélective (option UNE) :

```
   ANALYSE DE LA SPECIFICATION DE <nom type>
← Entrez un nom de type ou "FIN"
→ <nom type>
```

Le processus est arrêté par la commande "FIN".

[On trouvera en ANNEXE III, un exemple de fonctionnement inspiré du projet MONT-LOZERE ([BOUD 82], [CHABJ 82])].

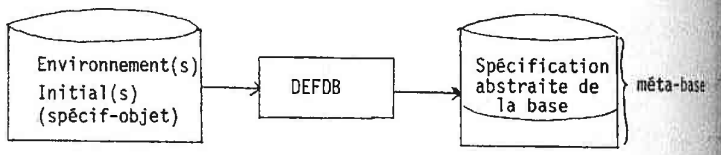
II - DEFDB, UN OUTIL D'AIDE A LA SPECIFICATION DE BASES ABSTRAITES

DEFDB est destiné à aider l'utilisateur à construire une spécification de base abstraite. Il est une réalisation du type C-SCHEMA (chapitre I/C), conçue de telle sorte à dissimuler à son

utilisateur les aspects formels des O-ENTITE, ENTITE, O-ASSBIN et ASSBIN.

Son utilisation suppose que les liens attributs ont été définis (c'est ce qui a pu donné lieu aux agrégats "compilés" par ANALTYP) et que les liens associations sont connus.

II.1.- Principe et fonctionnalités de DEFDB



L'utilisateur de DEFDB peut construire, de façon interactive, une base abstraite à partir d'un environnement initial préalablement créé par ANALTYP.

DEFDB guide l'utilisateur dans la spécification de la base

- pour définir les entités
  - importation de l'agrégat que l'ENTITE doit représenter
  - indication de la CLE
  - restriction éventuelle de ses opérations
  - renommage éventuel de ses opérations
  - redéfinition de l'agrégat par restriction de ses attributs, si nécessaire.
- pour décrire les associations
  - nom de l'association
  - indication de sa collection
  - nom de ses fonctions rôles
  - cardinalités.
- pour introduire des opérations mettant en jeu plus d'un type d'ENTITE et/ou d'ASSOCIATION (les opbds - cf. chapitre I/C)

  - symbole de l'opération
  - profil.
- pour exprimer des axiomes

DEFDB effectue quelques contrôles de cohérence :

- unicité des symboles utilisés (pas de synonymie)
- présence de toutes les sortes impliquées par les diverses spécifications
- "légalité" des associations (les agrégats de la collection doivent être présents)
- ...

L'intérêt d'un outil tel que DEFDB est donc d'alléger la tâche d'un utilisateur pour spécifier une base de données.

II.2.- Un exemple de fonctionnement

Voir ANNEXE IV.

CHAPITRE III : CONCRÉTISATION ET UTILISATION D'UNE BASE DE DONNÉES DÉDUCTIVE

I - INTRODUCTION

I.1.- Généralités

La démarche préconisée pour définir une base abstraite de données permet de considérer et de spécifier, progressivement :

- des entités (ou agrégats de base)
- leurs associations
- les abstractions relationnelles nécessaires
- les opérations associées à l'un ou l'autre des trois constituants précédents du schéma de la base.

La spécification obtenue résulte de la "fusion" de spécifications des constituants de bases (entités et associations) et des abstractions relationnelles. Elle constitue ainsi, une caractérisation statique d'une classe "d'objets bases de données".

Dans les approches traditionnelles, la mise en place d'une base de données consiste à décrire son schéma en termes du langage de définition de données (LDD) et à y ranger des données (chargement initial) puis, évidemment, à les utiliser (via le langage de requêtes et/ou les programmes qui auront pu être réalisés).

La description dans le LDD est "compilée" par le SGBD et détermine une base de données physique (initialement vide). Nous avons vu (au § III du chapitre I de cette partie) que l'approche que nous préconisons permet d'aboutir, si on le désire, à plusieurs bases de même schéma abstrait et de schémas de représentation identiques ou différents.

Dans ce chapitre, nous adopterons une démarche "plus opérationnelle". Nous y montrons comment déterminer une représentation

de la base abstraite puis comment utiliser cette représentation pour obtenir et utiliser une base de données physique.

I.2.- Etapes pour la concrétisation d'une base

La concrétisation d'une base passe par deux phases :

- une phase, préliminaire, concerne le choix du logiciel qui servira à son implémentation. Celui-ci doit comporter, outre les fonctions de gestion de données (offrir des structures de représentation et des fonctions d'accès), des facilités de prise en compte de la notion de type abstrait (qui permettraient d'étendre l'éventail des structures de représentation et des procédures d'accès offerts par le dit logiciel). Il devra également comporter des mécanismes de déduction.
- la seconde concerne la production à proprement parler de la base, en procédant en trois étapes :

  1 - Définition des constituants (attributs, entités, associations..) dans les termes du logiciel choisi : on dira que l'on définit une représentation de la base. Cette étape serait, en termes traditionnels, celle de la description du schéma de la base dans le LDD du SGBD. La différence essentielle est dans le fait que l'on souhaiterait ne pas être astreint, pour la définition des constituants, aux seuls types de données offerts par le SGBD mais que l'on puisse en introduire de nouveaux selon les besoins.

  2 - L'implémentation d'un type abstrait vise à définir le corps des procédures associées à ses opérations. De façon analogue, il faut prendre soin de définir les "programmes" qui opèreront sur la base. Le "plus" nécessité par l'approche types abstraits, par rapport à la démarche traditionnelle, est que l'on puisse définir puis utiliser, au sein de ces programmes, les procédures associées aux opérations des types, de la même

façon que les instructions du LMD.

3 - Enfin, la dernière étape est celle de la mise en place et de l'utilisation qui consiste à créer un ensemble d'occurrences d'objets de la base et à l'utiliser (interrogation, modification) en mettant en oeuvre les programmes et procédures réalisés lors de l'étape précédente.

Dans ce chapitre, nous allons détailler chacun de ces points en présentant simultanément :

- d'une part, une "solution-papier" en termes d'un système relationnel fictif que nous avons appelé ABSTRACT-R.
- d'autre part, une solution en VEGA, système que nous avons utilisé pour quelques unes de nos expérimentations.

Nous allons maintenant présenter les outils de VEGA que nous utiliserons et les fonctionnalités que nous attendons de ABSTRACT-R.

I.3.- VEGA et son interface avec PROLOG

VEGA fournit un environnement initial de spécification de types qui permettent de représenter des constituants d'une base (cf. § III.2/I/B). Cet environnement contient également des constructeurs de types. Nous reproduisons ci-dessous le constructeur TUPLE dont nous ferons souvent usage par la suite.

Nous donnons également, quelques précisions sur des aspects syntaxiques du "langage de manipulation de données" de VEGA et de son utilisation via l'interface VEGA-PROLOG.

a) Le constructeur TUPLE

SYSTEM (TUPLE(s_1:S_1 ;s_2:S_2;...;s_n:S_n))/
SORTS (T)/IMPORTS (S_1;S_2;...;S_n;BOOL;INT)/

BUILD/
T((S_1;S_2;...;S_n) ← T)/
OKOPNS/
T(s_i(T) ← S_i)/ %accès au $i^{ème}$ constituant%
%pour tout i, i ∈ [1,n]%
T(T=T ← BOOL)/ %égalité de Tuples%
T(SIZE(T) ← INT)/ %nombre de constituants%
EROPNS/
T($ERROR_i$ ← S_i)/
OKAXS/
S_i(s_i(x_1;x_2;...;x_n) = x_i)/
BOOL((x_1;x_2;...;x_n) = (y_1;...;y_n) =
ITE(S_1(x_1=y_1)...S_n(x_n=y_n),TRUE,FALSE))/
INT(SIZE(x_1;...;x_n) = ITE(INT(0<n)),n,0)/
ERAXS
S_i(s_i(x_1;...;x_n) = IT(S_i(x_i) = TRUE,x_i,$ERROR_i$))/
END (TUPLE(<s_1:S_1;s_2:S_2;...;s_n:S_n>))/

Outre ce système TUPLE, l'environnement initial de VEGA comporte également, les types (SYSTEMes) :

- Booléen (BOOL)
- caractère (CHAR)
- entier (INT)
- chaine
- ensemble de... (SET(-))
- séquence de... (SEQ(-))

b) Le langage de manipulation de VEGA et l'interface avec PROLOG

Le jeu d'instructionsde VEGA est constitué par :

1) RUN (<expression>) ?

ex :RUN(TSETHEB(SH ← ADD(SH,LENID;4;2)))? permet d'ajouter (ADD) le tuple <LENID,4,2> à l'ensemble SH(de type TSETHEB).

2) MRUN (<instruction>[;<suite-d'instructions>]*) où les instructions peuvent être
   - des RUN
   - des PRINT (impression)
   - des LINE (saut de ligne)
   - des EACH et/ou FIRST

3) EACH et FIRST sont des instructions permettant d'appliquer une action à tout (EACH) ou au premier (FIRST) élément d'une collection nommée. Leurs syntaxes respectives sont les suivantes :

   EACH(<variable de sorte s> IN <variable collection de sortes s>
   [:(<expr. logique>)]

   FIRST(<variable de sorte s> IN <variable collection de sortes s>[:(<expr. logique >)])

Exemple : • sortir tous les hébergements situés en MARGERIDE

```
MRUN(EACH(*H IN SETSITUATION(ST):STRING(SITUATION
        (NOMPAYS(*H))=MARGERIDE));PRINT(*H);LINE);
```

• sortir le premier hébergement de plus de 4 places

```
MRUN(FIRST(*H IN TSETHEB(SH):INT(THEB(CAPH(*H))>4));
        PRINT(*H))?
```

Ainsi, un programme VEGA est constitué d'une instruction RUN ou MRUN. L'interface de VEGA avec PROLOG permet d'autres formes d'expression, utiles pour se dégager de la syntaxe quelque peu ardue de VEGA, et aussi pour exprimer des requêtes plus complexes.

Elles auront l'une des trois formes syntaxiques suivantes :

(1) +P(*X,*Y,...,*W) - <expression VEGA>
(2) +P1(*X,...,*Z) - P2(-,-,..)-P3(-,...,-)-...-Pn(-,...,-)
(3) - Pj(-,...,-)[-Pk(-,...,-)-...].

Les formes (1) et (2) sont utilisées pour définir des schémas de requêtes où la partie gauche désigne "l'en-tête" de la requête avec ses arguments, et la partie droite le "corps" de la requête.

La forme (3) est une demande d'exécution.

Nous présenterons le mécanisme d'évaluation des requêtes au paragraphe III.2.1. de ce chapitre.

I.4.- ABSTRACT-R : un SGBD relationnel "papier"

ABSTRACT-R est un système relationnel imaginaire que nous allons utiliser pour montrer les exigences de notre approche. Il nous permettra de situer les systèmes actuels par rapport à un système déductif supportant la notion de type abstrait.

Nous supposons qu'il dispose d'un langage de manipulation de données "à la" SEQUEL. Nous supposons également qu'il offre la possibilité de définir de nouveaux types et opérations associées.

Dans ce chapitre, la définition de nouveaux types sera introduite par l'expression syntaxique :

DEFINE-TYPE T AS ...

L'expression DEFINE-OPN <symbole opérations>(<argop>) ON <type>
RETURNS <vartype>
AS <corps>

introduit
- une opération <symbole-opération>
- ses arguments <argop>
- le type sur lequel elle porte (ON <type>)
- son/ses résultats éventuels (<vartype>)
- sa définition explicite (<corps>)

Enfin, DEFINE-ABSTRACTION introduira la définition des abstractions relationnelles.

II - DE LA SPECIFICATION D'UNE BASE A SA REPRESENTATION

Outre l'association de "types de données informatiques" à

des types de la spécification et la définition des corps de leurs opérations, la représentation d'une base s'intéresse également à choisir les collections de données qui feront l'objet d'une mémorisation explicite et celles dont les données seront implicites.

En effet, c'est à ce niveau que l'on devra "fixer" les collections à définir en extension et celles qui le seront en intention.

II.1.- Définition en intention et définition en extension

Reprenons brièvement la définition de ces deux notions, en liaison avec les concepts d'entité, d'association et d'abstraction relationnelle.

II.1.1.- Définition en extension

La définition d'un ensemble en extension consiste à en énumérer les éléments.

Pour les collections d'objets qui nous intéressent et qui seraient définies de cette façon, le problème est assez simple :

- la constitution des collections est faite de façon explicite par l'émission d'opérations d'adjonction d'éléments à la collection (initialement vide).
- les collections peuvent être utilisées à l'aide des opérations de recherche (de sous-ensembles vérifiant éventuellement une propriété donnée), de modification/suppression d'éléments.

En termes informatiques, ces opérations s'appliqueront sur une structure de données associée à la collection.

Exemple : Soit HEBERGEMENT = AGREGAT (NOMH, CAPACITE, CATEGORIE)
On pourra le représenter par un tuple THEB (string,entier, entier) et l'ensemble des hébergements par SET(THEB) sur lequel on pourra appliquer les opérations du type SET (insertion, estdans,...)

- insertion (SHEB,("LION D'OR",80,2))
- estdans?(SHEB,("LION D'OR",80,2))
- ...

[Les objets informatiques associés seraient alors, par exemple, un fichier contenant des enregistrements ayant la structure du tuple].

Le problème est tout autre dès qu'il s'agit des abstractions construites sur les agrégats de base.

Leur définition en extension impose de décider de la façon dont sera construite et utilisée l'extension, c'est-à-dire, est-ce à l'utilisateur ou au système de s'en charger ?

Dans le premier cas, l'utilisateur devra expliciter les opérations sur l'extension alors que dans le second, le système détermine les actions à entreprendre sur les objets de base qui participent à la définition de l'abstraction, en fonction de l'opération émise sur cette dernière.

Exemple : Considérons E1 comme l'ensemble des hébergements,
E2 l'ensemble des hébergements de type CAMPING ($E2 \subset E1$)
et E3 celui des HOTELs ($E3 \subset E1$).

Si E2 et E3 sont définis en extension, l'ajout d'un hébergement h à E1 devra s'accompagner d'un ajout de h à E2 ou E3 (en supposant que CAMPING et HOTEL soient les seules natures d'hébergement possibles), de même qu'un ajout de h' à E2 ou E3 devra entraîner l'ajout de h' à E1.

Cet exemple sommaire montre suffisamment la redondance des actions et des informations mémorisées. Il a trait au problème des opérations de mise à jour au sein d'une base de données déjà évoqué au paragraphe II.5/III/A.

Pour notre part, dans les expérimentations que nous avons menées, nous avons délibérément ignoré ce problème. Il devra faire l'objet de recherches ultérieures. Nous avons considéré que tout ensemble défini en extension devait être manipulé (constitué et modifié) par le biais d'opérations explicites émises par l'utilisateur.

II.1.2.- Définition en intention

Elle consiste à caractériser un ensemble par des propriétés que doivent posséder ses éléments. Ces propriétés expriment ce que signifie ETRE MEMBRE de l'ensemble.

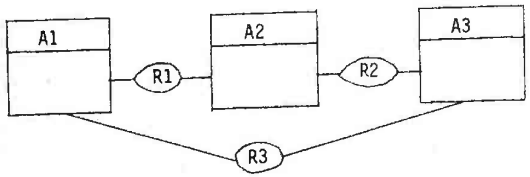
Il s'agira pour nous, dans ce cas, d'exprimer les propriétés définissant l'intention dans le langage du système logiciel d'implémentation.

En termes procéduraux, il s'agit de définir le corps de la procédure qui permet de "calculer" des instances de l'intention à partir d'instances d'extension et selon un processus logico-mathématique que définissent les propriétés de la dite intention.

Nous noterons enfin que ces modes de définition de collections d'objets concernent aussi bien les agrégats que les associations du schéma de la base.

En effet, en ce qui concerne ces dernières, le fait que nous ayions souvent à considérer des ensembles d'objets reliés par une association nous impose de choisir une représentation pour ces ensembles. Pour certains, seule la définition en extension est possible. (Exemple : l'association PERSONNE-ENFANT). Pour d'autres le choix est possible.

Exemple : Soit le schéma suivant où A1,A2,A3 désignent des agrégats et R1,R2,R3 des associations.



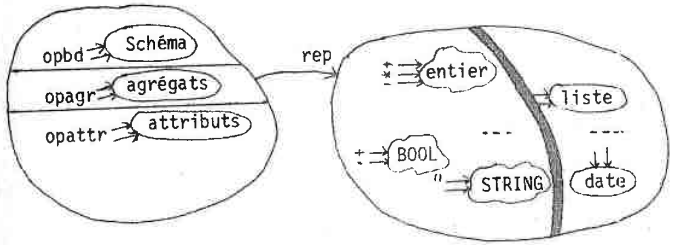
Moyennant certaines conditions, on pourra définir une des associations "sur" les extensions des deux autres.

Ces rappels étant faits, revenons à la représentation d'une base de données.

II.2.- La représentation d'une base

Pour représenter des données en termes de types informatiques, les systèmes traditionnels offrent un ensemble fixe constitué par les types les plus courants (entier, chaîne de caractères, record, ensemble de records...). Dans notre vocabulaire, ces types sont les types et les constructeurs de base définis dans l'environnement initial de spécification.

En ce qui nous concerne, nous avons à déterminer l'ensemble des types informatiques nécessaires à la représentation de ceux de la spécification de la base en utilisant ceux de l'environnement initial ou en en introduisant d'autres, si nécessaire. (Ces derniers peuvent être totalement nouveaux ou être le résultat d'un des mécanismes de construction de types (restriction, enrichissement...) que nous avons présentés).



Le but de ce paragraphe est d'expliciter la fonction de représentation (rep) qui permet d'associer à chaque constituant de la spécification abstraite (attributs, agrégats, associations et opérations) une représentation (structure de données et/ou procédure).

II.2.1.- Les attributs

La représentation des attributs consiste à leur associer un domaine.

Dans ce sens, certains ont déjà pu faire l'objet d'une représentation, d'autres nécessitent un effort supplémentaire de spécification.

Concrètement, la représentation des attributs leur associera :

- un type de base présent dans l'environnement (ou type de données connu du système)

ex : NOMH : STRING
CAPACITE : INTEGER

- un type "nouvellement" spécifié ou construit

ex : Si DATE, attribut composé de JOUR, MOIS, AN n'est pas exprimable en termes de types de base, on devrait avoir la possibilité de le spécifier (ou de le construire) et de faire prendre en compte sa spécification par le système : il deviendrait ainsi un type connu du système.

II.2.2.1.- Cas de ABSTRACT-R

ABSTRACT-R est supposé disposer des "types informatiques" de base (entier, caractère, chaîne de caractères,...). Ils seront utilisés pour représenter certains attributs.

ex : NOMH : CHAINE
CAPACITE : ENTIER

Il devra également être capable de prendre en compte de nouvelles spécifications de type, comme celle de DATE donnée ci-dessous.


```
DEFINE-TYPE TDATE
        AS J :INTEGER IN [1..31]
           M : INTEGER IN [1...12]
           A : INTEGER IN [1900...2002]
```

En outre, on devrait également avoir la possibilité de définir les opérations rattachées au type introduit.

Ainsi, outre l'accès aux domaines de J,M,A (que l'on exprimera par une notation pointée), on pourrait également définir MEMEDATE et PLUSTOT ainsi :

```
DEFINE-OPN MEMDATE (D1:TDATE,D2:TDATE) ON TDATE RETURNS BOOL
        AS TRUE IF D1.J = D2.J AND D1.M = D2.M AND D1.A = D2.A

DEFINE-OPN PLUSTOT(D1:TDATE) ON TDATE RETURNS D2:TDATE
        AS(D2.J<D1.J AND D2.M = D1.M AND D2.A = D1.A)
          OR (D2.M<D1.M AND D2.A + D1.A)
          OR (D2.A<D1.A)
```

Les types ainsi définis pourront alors être utilisés pour représenter des attributs ; leurs opérations seront disponibles pour les utilisateurs du système.

II.2.1.2.- Expression en VEGA

Selon le même principe que dans ABSTRACT-R, aux attributs pourront être associés des MODELs de l'environnement initial ou des MODELs nouvellement construits.

Reprenons l'exemple de la date. Son type sera construit à l'aide de trois types, élaborés par restriction du type entier (INT), et du type TUPLE.

```
SYSTEM INTJOUR/SORTS(INTJOUR)/
    MODEL (INT RESTRICTO(*J:1 TO 31))/
END (INTJOUR)/

SYSTEM INTMOIS/SORTS(INTMOIS)/
    MODEL(INT RESTRICTO(*M:1 TO 12))/
END (INTMOIS)/

SYSTEM INTAN/SORTS(INTAN)/
    MODEL(INT RESTRICTO(*J:1900 TO 2002))/
END (INTAN)/

SYSTEM (TDATE)/SORTS(TDATE)/
    IMPORTS (INTJOUR;INTMOIS;INTAN;BOOL)/
    MODEL(TUPLE(<J:INTJOUR;M:INTMOIS;A:INTAN>))/

    OKOPNS
        TDATE(MEMEDATE(TDATE,TDATE) ← BOOL)/
        TDATE(PLUSTOT(TDATE) ← TDATE)/
      ⋮
    OKSEMS/
        BOOL((MEMEDATE(REP(*D1),REP(*D2)) = (*D1 = *D2))/
        ⋮
END (TDATE)
```

REMARQUE :

Cette spécification, selon la technique dite du modèle abstrait [CHAB 82], nous amène à faire quelques remarques :

- la clause MODEL introduit le modèle abstrait
- la clause RESTRICTO permet d'exprimer des contraintes sur le type. Ainsi INTJOUR, INTMOIS et INTAN sont définis à partir du type entier (INT) avec les intervalles de valeurs respectifs

[1..31], [1..12] et [1900..2002].

- les nouveaux types (SYSTEMes) INTJOUR, INTMOIS et INTAN sont utilisés ensuite pour définir le type TDATE comme un TUPLE dont on étend la liste des opérations par PLUSTOT et où on renomme l'égalité de TUPLEs en MEMEDATE.
- la clause OKSEMS introduit les axiomes définissant les opérations du type spécifié TDATE en termes d'opérations de son MODELe. Ainsi (*D1 = *D2) est la représentation (en terme d'égalité de tuples) de MEMEDATE.

II.2.2.- Cas des entités de base

Leur représentation concerne

- la classe d'objets (que détermine le type ENTITE)
- un membre de la classe (type O-ENTITE).

Dans les systèmes relationnels traditionnels, la définition des RELATIONs englobe les deux niveaux de représentation. Pour notre part, nous les dissocierons pour des raisons de clareté et de progression dans la démarche, et aussi, pour bien montrer sur quel objet s'applique telle ou telle opération définie.

La présentation qui suit s'appuiera sur l'exemple d'agrégat suivant :

```
type HEBERGEMENT = AGREGAT(NOMH,CAPACITE,CATEGORIE)
    opérations
       fnomh : HEBERGEMENT → NOMH
       fcap :  HEBERGEMENT → CAPACITE
       fcat :  HEBERGEMENT → CATEGORIE
       plusgrand : HEBERGEMENT×HEBERGEMENT → BOOL
    axiomes
       h1,h2 : HEBERGEMENT
     • plus grand(h1,h2)=si sup_cap(fcap(h1),fcap(h2))alors VRAI
                                                  sinon FAUX
     • ...
ftype
```

On suppose que sur CAPACITE est définie une opération sup_{cap} qui permet de comparer des capacités d'hébergement.

II.2.2.1.- Représentation des agrégats dans ABSTRACT-R

Dans les systèmes relationnels, la notion de type d'une relation n'existe pas en tant que telle. Ainsi la déclaration

RELATION HEBERGEMENT(NOMH:CHAINE,CAPACITE:ENTIER,CATEGORIE:ENTIER)

décrit un ensemble de tuples <NOMH, CAPACITE, CATEGORIE> où HEBERGEMENT est le nom de l'ensemble.

On considère donc qu'implicitement la collection de tuples est de type ensemble et que le nom de la relation désigne un objet de ce type. De plus, implicitement également, il ne peut exister dans la base qu'une seule instance du type.

La différenciation des niveaux de représentation pemettra
- de dissocier les identificateurs des types, des variables déclarées de ces types.
- de distinguer l'identificateur du type représentant un tuple d'une relation donnée, (ce type définirait en somme le schéma de la relation [ANNEXE II]), de celui du type représentant l'ensemble des tuples (l'occurrence de la relation).

a) La représentation d'une instance d'agrégat

Elle sera généralement faite par le biais d'un type tuple. Les fonctions d'accès à ses constituants seront exprimées sous forme de notation pointée (qui elle-même représente la projection du tuple sur un de ses constituants).

Les autres fonctions ou opérations éventuellement définies sur l'agrégat seront introduites par DEFINE-OPN.

Exemple :
```
DEFINE-TYPE THEB
   AS NOMH : STRING
      CAPACITE : INTEGER
      CATEGORIE : INTEGER
```



```
DEFINE-OPN PLUS-GRAND(H1:THEB,H2:THEB)ON THEB
           RETURNS BOOL
   AS TRUE IF H1.CAPACITE>H2.CAPACITE
```

STRING, INTEGER et BOOL sont considérés comme étant des types connus de ABSTRACT-R. H1 et H2 sont des paramètres formels de la fonction PLUSGRAND. Des variables de tuples THEB devront leur être substituées lors de chaque utilisation de la fonction.

Par ailleurs, dans le "corps" de PLUS-GRAND, la relation entre les capacités est exprimée par '>' qui opère sur les entiers (et qui est donc la représentation de sup_{cap}).

Nous verrons plus loin, (§ II.2.3. de ce chapitre), que la structure de représentation des agrégats peut être modifiée par la représentation des associations.

b) Représentation d'une collection d'agrégats

Elle s'exprime à l'aide d'un type "collection de tuples". Ce type, comme déjà dit, était implicitement un ensemble (SET), avec les opérations de recherche, d'ajout, de suppression et de modification de tuples.

Dans ABSTRACT-R, la définition de la collection en tant qu'ensemble devra être explicitée (exemple 1). Si nécessaire, ses opérations devront être exprimées en termes d'opérations sur l'ensemble (exemple 2).

Exemple 1 :
```
DEFINE TYPE TSETHEB AS SET (ou RELATION) OF THEB
```

Exemple 2 :
```
DEFINE-OPN AJOUTHEB(SH:TSETHEB,H:THEB) ON TSETHEB
           RETURNS SH:TSETHEB
   AS INSERT H INTO SH
```

Le mécanisme de définition des opérations (avec renommage) permet de rapprocher le vocabulaire de la représentation de celui

en vigueur dans le domaine que la base de données représente, et de donner un corps aux opérations. En d'autres termes, l'utilisateur pourra faire usage de AJOUTHEB dans ses requêtes ou ses programmes, laissant au système le soin de traduire AJOUTHEB(SH,H) en "INSERT H INTO SH".

II.2.2.2.- La représentation des agrégats dans VEGA

Le principe étant le même que pour ABSTRACT-R, nous nous contenterons donc d'exprimer le même exemple dans le langage de VEGA

a) Représentation d'un agrégat

```
SYSTEM (THEB)/SORTS(THEB)/
   MODEL (TUPLE(<NOMH:STRING;CAPACITE:INT;CATEGORIE:INT>))/
   OKOPNS
         THEB(PLUSGRAND(THEB,THEB) ←BOOL)/
          ⋮
   OKSEMS
         BOOL((PLUSGRAND(REP(*H1))=ITE(INT(CAPACITE(*H1)>
               CAPACITE(*H2)),TRUE,FALSE))/
     ⋮
END (THEB)/
```

b) Représentation de la collection d'agrégats

```
SYSTEM (TSETHEB)/SORTS(TSETHEB)/
   MODEL (SET(THEB))/
   OKOPNS
         TSETHEB(AJOUTHEB(TSETHEB,THEB) ← TSETHEB)/
           ⋮
   OKSEMS
         TSETHEB(AJOUTHEB(REP(*S),*H) = ADD(*S,*H))/
     ⋮
END (TSETHEB)/
```


[ADD est l'opération d'ajout d'élément à un ensemble, définie sur le type SET].

REMARQUE :

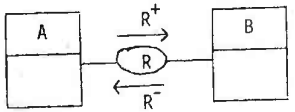
SET peut ne pas être le seul type utilisé pour représenter les collections. Selon les besoins, on pourrait en choisir d'autres (comme par exemple un ensemble ordonné, une file ...). On devra dans ce cas, les spécifier dans un premier temps (en tant que SYSTEMe) avant de pouvoir les utiliser (en tant que MODELe).

La remarque est également applicable à ABSTRACT-R. Exprimée en termes de SGBD traditionnels, cela voudrait dire que l'on "type" les relations utilisées que l'on voudrait organiser autrement qu'en "vrac". Ceci concerne essentiellement les relations sur lesquelles on veut définir un ordre ou des index. On devra, de ce fait, leur associer un type (et ses opérations) adéquat.

II.2.3.- La représentation des associations

Comme pour la représentation d'un agrégat, on va s'intéresser pour l'association à sa représentation proprement dite (ie en tant que membre d'une collection) puis à la collection à laquelle elle appartient.

De façon générale, une association R définie entre deux agrégats A et B peut être représentée par le produit cartésien des attributs clés de A et de B et des attributs propres à l'association, si elle en comporte.



$rep(R) = <clé_A, clé_B, attr_R>$

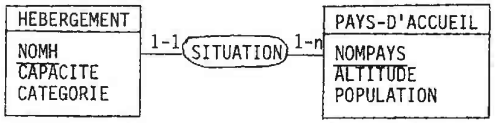
Une variante existe cependant. Elle consiste, au cas où R^+

(ou son inverse) est totale et monovaluée, à introduire la clé de B (resp. celle de A) et les attributs de R dans la représentation de A (resp. de B).

La fonction R^+ (ou son inverse) doit être totale et monovaluée car sinon, les valeurs des attributs introduits dans la représentation de l'agrégat (A ou B) seraient indéfinies pour certaines occurrences de l'agrégat. A la fois une perte de place et une particularisation des traitements qui porteraient sur R, sont alors occasionnés par ces valeurs non définies. (Ce point a rapport avec le problème du traitement de la négation en bases de données dont nous n'avons pas parlé jusqu'à maintenant. On en trouvera un exposé dans [GALL 78]).

En termes relationnels, on est donc amené à représenter une association dans le premier cas, par une relation, et dans le second cas, par l'introduction de la clé d'un des agrégats dans la définition de la relation associée à l'autre agrégat. (cf. décomposition des relations en ANNEXE II et § IV.2.3/I/A).

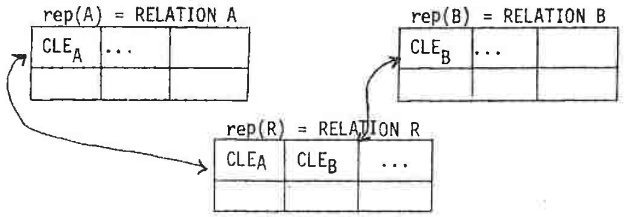
Exemple



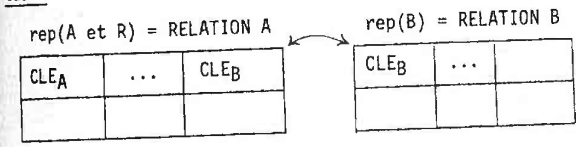
L'association SITUATION peut être représentée
- soit par <NOMPAYS,NOMH> (cas 1)
- soit par <NOMH,CAPACITE,CATEGORIE,NOMPAYS> (cas 2)

En termes de tableaux de relations, on aurait :

- Cas 1



- Cas 2



On établit ainsi des liaisons entre relations à l'aide de leurs attributs clés.

Une fois la représentation d'une occurrence d'association définie, on devra choisir le type représentant sa collection (ensemble, ensemble ordonné...). On s'attachera, en outre, à exprimer la représentation des fonctions définies sur l'association.

On notera cependant, que cette démarche est applicable uniquement dans le cas où l'on choisit la définition en extension. La définition en intention se fera de façon sensiblement différente, comme nous le verrons au paragraphe II.2.3.2. de ce chapitre.

II.2.3.1.- La représentation en extension des associations

Considérons l'association SITUATION entre PAYS et HEBERGEMENT et analysons par son biais, le processus de représentation en extension.

```
type TSITUATION = ASSOCIATION (PAYS, HEBERGEMENT)
ftype
```

L'utilisation du constructeur ASSOCIATION fait hériter SITUATION des opérations faire-assoc, proj1, proj2... (cf. § III.2.2/I/C).

Nous avons montré que O-ASSBIN est une représentation abstraite de ASSOCIATION. Appliqué à SITUATION, nous obtenons :

```
type TSITUATION = O-ASSBIN(1PAYS DE HEBGT:1PAYS,1HEBGT DU PAYS:
                  1HEBERGEMENT)
   sortes TSITUATION,O-ASSBIN,1PAYS,1HEBERGEMENT,BOOL
   opérations
         1SITUATION:PAYS×HEBERGEMENT → TSITUATION
         MEMESIT : TSITUATION×TSITUATION → BOOL
         1PAYS DE HEBGT : TSITUATION → 1PAYS
         1HEBGT DU PAYS : TSITUATION → 1HEBERGEMENT
   axiomes
         %ceux de O-ASSBIN où l'on renomme
           Faire-OASSBIN par 1SITUATION
           Egal-Orel     par MEMESIT
           f1            par 1PAYSDEHEBGT
         %  f2            par 1HEBGTDUPAYS %
ftype
```

REMARQUE :

La représentation des associations à l'aide des clés des entités revient à choisir de désigner les instances d'entités associées en indiquant uniquement leur clé. (C'est évidemment un choix de représentation. Il peut en exister d'autres).

De ce fait, lors de la représentation des agrégats, on devra choisir l'attribut-clé de l'agrégat. La fonction d'accès qui lui est associée a la propriété d'être injective.

Cette notion de CLE nous permet par ailleurs de reformuler l'égalité entre O-ENTITE en termes d'égalités de leurs clés :

```
type O-ENTITE(C1:T1,...,CLE:TCLE,...,Cn:Tn)
   sortes ...
   opérations
         Faire-OENT : T1×...×TCLE×...×Tn → O-ENTITE
            ⋮
         CLE : O-ENTITE → TCLE
            ⋮
         Egal-OENT : O-ENTITE×O-ENTITE → BOOL
```



```
   axiomes
         Egal-OENT(Faire-OENT(t1,...,tclé,...,tn),e) =
            si EGAL_TCLE(t_clé,CLE(e)) alors Vrai sinon faux
      ⋮
ftype
```

Par ailleurs, le fait que l'on considère des collections d'association, nous a amené à spécifier un type adéquat (type ASSBIN) et ses opérations. A cette occasion, nous avions dit qu'à toute association était implicitement associée une ASSBIN paramétrée par la O-ASSBIN représentant l'association. On associera ainsi TSETSITUATION à SITUATION :

```
type TSETSITUATION = ASSBIN(TSITUATION,(1,n),(1,1))
   sortes TSETSITUATION, TSITUATION, ASSBIN, Entier, BOOL
   opérations
      SETSIT :                          → TSETSITUATION
      AJOUTSIT : TSETSITUATION×TSITUATION → TSETSITUATION
      ENLEVSIT : TSERSITUATION×TSITUATION → TESETSITUATION
      DANSIT :   TSETSITUATION×TSITUATION → BOOL
      PAYSDEHEBGT : TSETSITUATION×1HEBERGEMENT → PAYS
      HEBGTDUPAYS : TSETSITUATION×1PAYS → HEBERGEMENT
   axiomes

      %ceux de ASSBIN où
            (min1,max1) = (1,n),(min2,max2) = (1,1)
       et où l'on renomme
            ASS-INIT    en  SERSIT
            AJTASS      en  AJOUTSIT
            SUPRASS     en  ENLEVSIT
            EXISTASS    en  DANSIT
            VALF        en  HEBGTDUPAYS
            INVERSF     en  PAYSDEHEBGT %

ftype
```

La représentation d'une association concernera alors celle de l'O-ASSBIN et de l'ASSBIN associés.

L'expression des opérations d'une O-ASSBIN différera selon que celle-ci est représentée par un tuple indépendant <CLE_A,CLE_B, ...>, ou par l'inclusion d'une des clés dans la représentation d'un des deux agrégats associés.

Nous allons examiner successivement chacun de cas appliqué à ABSTRACT-R et à VEGA.

a) Cas 1 : Représentation par <$clé_A$, $clé_B$,...>

α) Représentation dans ABSTRACT-R

```
- DEFINE TYPE TSITUATION        %définit une association%
  AS NOMPAYS : STRING
     NOMH : STRING
- DEFINE-OPN 1PAYSDEHEBGT(T:TSITUATION) ON TSITUATION
                              RETURNS NP:STRING
  AS NP : T.NOMPAYS
- DEFINE-OPN 1HEBGTDUPAYS(T:TSITUATION) ON TSITUATION
                              RETURNS NH : STRING
  AS NH : T.NOMH
- DEFINE-TYPE TSETSITUATION   %définit la collection d'association%
  AS SET(ou RELATION) OF TSITUATION
- DEFINE-OPN PAYSDEHEBGT(S:TSETSITUATION,H:STRING)
              ON TSETSITUATION RETURNS NP : STRING
  AS SELECT NOMPAYS FROM S
                    WHERE NOMH = H
- DEFINE-OPN HEBGTDUPAYS(S:TSETSITUATION,P:STRING)
              ON TSETSITUATION RETURNS SET(STRING)
  AS SELECT NOMH FROM S WHERE NOMPAYS = P
- DEFINE-OPN AJOUTSIT(S:TSETSITUATION,T:TSITUATION)
              ON TSETSITUATION RETURNS S:TSETSITUATION
  AS INSERT T INTO S
```


- ... On exprimerait ainsi chacune des opérations définies sur TSITUATION et TSETSITUATION à l'aide du langage du système d'implémentation.

β) Représentation en VEGA

Nous pouvons adopter deux types d'approche :

- l'une purement algébrique : on associe des MODELes à O-ASSBIN et ASSBIN et on spécifie algébriquement leurs opérations.
- l'autre plus "programmatoire" : on associe des MODELes aux objets et on exprime leurs opérations à l'aide de l'interface VEGA-PROLOG.

En illustration, la représentation de SITUATION mêlera les deux approches.

```
SYSTEM (TSITUATION)/MODEL TUPLE( NOMPAYS:STRING;NOMH:STRING )/

    OKOPNS/
       TSITUATION(1PAYSDEHEB(TSITUATION) ← STRING)/
       TSITUATION(1HEBGTDUPAYS(TSITUATION) ← STRING)/
       :
    OKAXS/
       STRING(1PAYSDEHEB(rep(*T)) ← NOMPAYS(*T))/
       STRING(1HEBGTDUPAYS(rep(*T)) ← NOMH(*T))/
       :
END(TSITUATION)/


SYSTEM (TSETSITUATION)/MODEL SET(TSITUATION)/
   :
    OKOPNS
       TSETSITUATION(DANSIT(TSETSITUATION,STRING,STRING) ← BOOL)/
       TSETSITUATION(AJOUTSIT(TSETSITUATION,TSITUATION) ← TSETSITUATION)/
        :
```

```
    OKAXS
      TSETSITUATION(AJOUTSIT(REP(*S),*T) = ADD(*S,*T))/
      BOOL(DANSIT(REP(*S),*T) = ITE((*T IN *S),TRUE,FALSE))/
       :
END (TSETSITUATION)/
```

Les opérations PAYS DE HEBGT et HEBGT DU PAYS sont décrites sous la forme dite "plus programmatoire" :

```
+HEBGT DU PAYS(*ST,*P)-EACH(*S IN TSETSITUATION(*ST):
                        STRING(1PAYS DE HEBGT(*S) = *P);
                        PRINT(1HEBGT DU PAYS(*S)))
+PAYS DE HEBGT(*ST,*H) - FIRST(*S IN TSETSITUATION(*ST) :
                        STRING(1HEBGT DU PAYS(*S) = *H);
                        PRINT(1PAYS DE HEBGT(*S)))
```

REMARQUE : Cette forme d'expression permet de définir des "schémas de programmes" dont la demande d'activation pourra être faite en invoquant la partie gauche (assimilable à l'en-tête d'une procédure) dans laquelle on aura instancié des variables.
Ainsi - HEBGT DU PAYS (ST,MARGERIDE)? aura pour effet de rechercher dans l'ensemble de SITUATION repéré par la variable ST, les hébergements du pays MARGERIDE.
Nous décrivons, au § III.2 de ce chapitre, le processus mis en oeuvre pour évaluer de telles expressions.

b) Cas 2 : Représentation par l'introduction de la clé d'un agrégat dans la représentation de l'autre agrégat

Dans ce cas, l'association SITUATION est représentée par l'inclusion de NOMPAYS dans la représentation de HEBERGEMENT. L'expression des opérations se trouve modifiée en conséquence.

α) Expression dans ABSTRACT-R

```
DEFINE-TYPE THEB
  AS NOMH : STRING
     CAPACITE : INTEGER
     CATEGORIE : INTEGER
     NOMPAYS : STRING
DEFINE-TYPE TSETHEB AS SET (OU RELATION) OF THEB

DEFINE-OPN 1PAYSDEHEBGT(T:THEB) ON THEB RETURNS NP : STRING
  AS NP = T.NOMPAYS

DEFINE-OPN 1HEBGTDUPAYS(T:THEB) ON THEB RETURNS NH : STRING
  AS NH = T.NOMH

DEFINE-OPN PAYSDEHEBGT(SH:TSETHEB,H:STRING) ON TSETHEB
                       RETURNS NP : STRING
  AS SELECT NOMPAYS FROM SH WHERE NOMH = H

DEFINE-OPN HEBGTDUPAYS(SH:TSETHEB,P:STRING) ON TSETHEB
                       RETURNS SNH : SET(STRING)
  AS SELECT NOMH FROM SH WHERE NOMPAYS = P.
```

Les opérations d'adjonction, de suppression, de modification d'éléments de la collection d'associations sont dans ce cas, "incluses" dans celles de la collection de l'agrégat qui "contient" la représentation de l'association.

(Ainsi, l'ajout de 1SITUATION(P1,H1) est exprimé "dans" l'ajout, dans l'ensemble des Hébergements, du tuple HEBERGEMENT représentant H1, soit <H1,C1,C2,P1>).

β) Expression en VEGA

```
SYSTEM (THEB)/MODEL TUPLE(<NOMH:STRING;CAPACITE:INT;CATEGORIE:INT;
                           NOMPAYS:STRING)/
```

```
      OKOPNS
         - {opérations sur THEB}
         - THEB(1PAYSDEHEB(THEB) ← STRING)/
         -THEB(1HEBGTDUPAYS(THEB) ← STRING)/
      OKAXS
         STRING(1PAYSDEHEB(REP(*H)) = NOMPAYS(*H))/
         STRING(1HEBGTDUPAYS(REP(*H)) = NOMH(*H))/
           ⋮
END (THEB)/

SYSTEM (TSETHEB)/MODEL SET(THEB)/
      OKOPNS
         TSETHEB(PAYSDEHEBGT(TSETHEB,STRING) ← STRING)/
         TSETHEB(HEBGTDUPAYS(TSETHEB,STRING) ← SETPAYS)/
         +{opérations de la collection d'hébergements}
END (TSETHEB)/
```

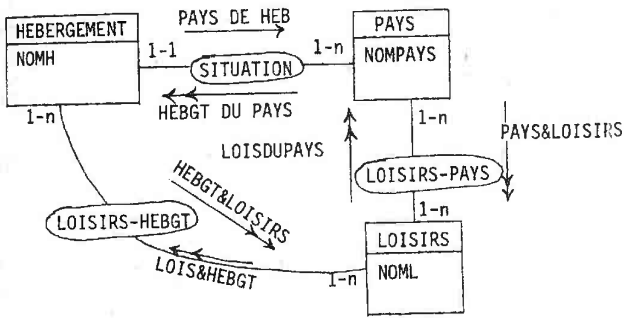
Comme dans le cas 1, on exprime PAYSDEHEBGT et HEBGTDUPAYS sous forme de "schémas de programmes" de l'interface VEGA-PROLOG.

```
+ PAYSDEHEBGT(*S,*H) - FIRST(*T IN TSETHEB(*S):STRING(NOMH(*T)=*H);
                       PRINT(1PAYSDEHEB(*T)))
+ HEBGTDUPAYS(*S,*P) - EACH(*T IN TSETHEB(*S):STRING(NOMPAYS(*T)=*P);
                       PRINT(1HEBGTDUPAYS(*T))
```

II.2.3.2.- Représentation intentionnelle des associations

Dans ce cas, on décide de ne pas mémoriser les occurrences d'une association. On doit alors se doter de moyens pour les "calculer" à partir des occurrences d'agrégats et/ou d'associations mémorisées.

Nous allons expliciter le mécanisme sur l'exemple suivant :

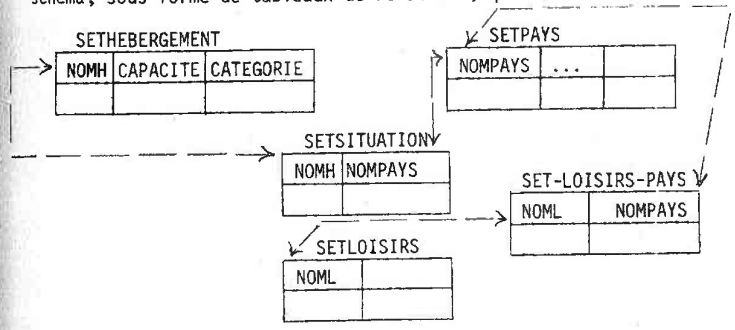


où LOISIRS-PAYS informe sur les loisirs offerts par un pays d'accueil et LOISIRS-HEBGT sur les loisirs à proximité d'un hébergement.

Nous supposons disposer d'une règle (de gestion qui va donner naissance à une règle d'inférence) qui stipule que les loisirs à proximité d'un hébergement sont ceux offerts par le pays d'accueil où se situe l'hébergement.

Moyennant cette règle, on pourra alors opter pour une définition en intention de LOISIRS-HEBGT en utilisant SITUATION et LOISIRS-PAYS.

Nous supposons pour cela que HEBERGEMENT, PAYS, LOISIRS, SITUATION et LOISIRS-PAYS ont déjà fait l'objet d'une représentation en termes de tuples et de collections de tuples, selon le schéma, sous forme de tableaux de relations, qui suit :



a) Expression en ABSTRACT-R

Dans les systèmes traditionnels, il est possible d'exprimer ces intentions sous forme de requêtes (ou de programmes).

Ainsi SELECT NOML FROM SETLOISIRS-PAYS
WHERE NOMPAYS = SELECT NOMPAYS FROM SETSITUATION
WHERE NOMH = 'LE NID'

permet d'obtenir les loisirs (NOML) à proximité de l'hébergement "LE NID".

Ce que l'on souhaiterait, c'est que ABSTRACT-R puisse générer de telles expressions à partir de la définition de leurs "atomes". Ainsi, on ne serait plus astreint à les expliciter soi-même sous forme de requêtes.

Exemple : Nous supposons avoir défini

PAYSDEHEB(H) par "SELECT NOMPAYS FROM SETSITUATION
WHERE NOMH = H"

et PAYS & LOISIRS(P) par "SELECT NOML FROM SETLOISIRS-PAYS
WHERE NOMPAYS = P"

Nous aimerions avoir la possibilité de définir HEBGT & LOISIRS en termes de PAYSDEHEB et de PAYS&LOISIRS, par une expression de la forme :

HEBGT&LOISIRS(H) = PAYS&LOISIRS(PAYSDEHEB(H)).

ABSTRACT-R devrait alors "savoir" transformer cette expression en une requête relationnelle.

Ainsi, la demande "Quels sont les loisirs à proximité de 'LE NID'" serait exprimée par HEBGT & LOISIRS ("LE NID"), puis transformée successivement en :

(1) PAYS & LOISIRS (PAYS DE HEB('LE NID'))

(2) PAYS & LOISIRS (SELECT NOMPAYS FROM SETSITUATION
WHERE NOMH = 'LE NID')

(3) SELECT NOML FROM SETLOISIRSPAYS
WHERE NOMPAYS = SELECT NOMPAYS FROM SETSITUATION
WHERE NOMH = 'LE NID'

Les requêtes ainsi obtenues sont "terminales". Elles pourraient alors, être évaluées par un système relationnel.

Ce mécanisme de transformation n'est pas applicable uniquement aux définitions en intention. Il concerne également les fonctions de base (les "atomes"), comme par exemple, PAYSDEHEB ou LOISIRSDANS-PAYS, si elles sont utilisées seules.

Ce processus peut s'interpréter (et être implanté) de diverses façons :

- en termes de procédures

Nous définissons en fait des procédures de base, comme PAYSDEHEB & PAYS & LOISIRS, et des "méta-procédures" qui peuvent avoir des procédures comme paramètres.

Le mécanisme utilisé dans ce cas, est celui d'unification et de substitution (avec tous les problèmes que risquent de poser la substitution de "procédures effectives" à des "procédures formelles").

- en termes de réécriture ([HUET 80], [JOUA 83])

La définition des opérations peut encore être interprétée comme une règle de réécriture (orientée de la gauche vers la droite). Dans ce cas, le mécanisme à utiliser procèdera par unification et réécriture (remplacement de parties gauches par des parties droites).

[Dans la présentation de l'exemple ci-dessous, nous prenons de larges libertés dans l'écriture des règles, afin d'illustrer brièvement cette seconde interprétation].

Exemple :

Soit les règles (R1), (R2), (R3) définissant respectivement PAYS DE HEB, PAYS & LOISIRS, et HEBGT & LOISIRS

(R1) PAYS DE HEB(X) → SELECT NOMPAYS ... WHERE NOMH = X

(R2) PAYS & LOISIRS(Y) → SELECT NOML ... WHERE NOMPAYS = Y

(R3) HEBGT & LOISIRS(Z) → PAYS & LOISIRS(PAYS DE HEB(Z))

L'expression HEBGT & LOISIRS ('LE NID') serait alors successivement réécrite en :

(1) PAYS & LOISIRS (PAYS DE HEB ('LE NID'))

par (R3) où l'on substitue 'LE NID' à Z

(2) PAYS & LOISIRS (SELECT NOMPAYS...WHERE NOMH = 'LE NID')

par (R1) et la substitution X/'LE NID'

(3) SELECT NOML ... WHERE NOMPAYS = SELECT NOMPAYS...
WHERE NOMH = 'LE NID'

par (R2) et la substitution Y/SELECT NOMPAYS...WHERE
NOMH = 'LE NID'

b) Expression en VEGA

Le principe est similaire à celui qui vient d'être exposé.

L'expression terminale est une phrase du langage de VEGA ou de son interface avec PROLOG. Le mécanisme de transformation et d'évaluation est basé sur la résolution de PROLOG.

Nous détaillerons ce point, au paragraphe III de ce chapitre, en liaison avec la mise en place et l'utilisation d'une base de données.

II.2.4.- Représentation des abstractions définies sur les objets de base

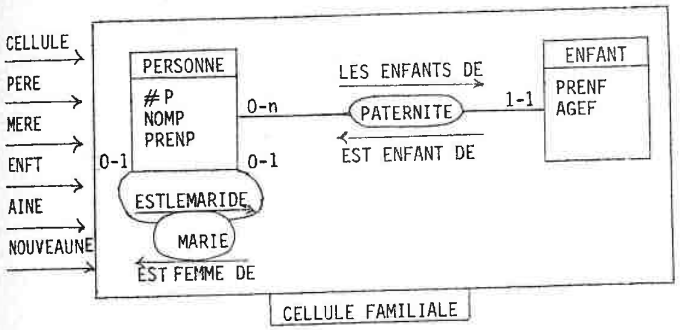
Jusqu'à présent, on ne s'était intéressé qu'à la représentation des agrégats et associations "de base". Comment peut-on représenter les abstractions que l'on a pu en faire lors de la phase de conception ?

Nous avons vu qu'une abstraction relationnelle pouvait s'assimiler à une vue sur laquelle pouvaient être définies et appliquées des opérations. Elle sera généralement définie en intention, sa définition en extension posant le problème dit de propagation de mise à jour dont nous ne nous sommes pas préoccupés dans le cadre de ce travail. Ses opérations seront exprimées (représentées) en termes d'opérations des agrégats et/ou associations qui ont servi à la définir.

Ainsi, la représentation d'une abstraction relationnelle consiste à :

- définir une "structure" des instances de l'abstraction
- caractériser ces instances à partir des instances d'objets de base
- définir les opérations qui leur sont applicables.

Pour expliciter le mécanisme de représentation de ces abstractions dans ABSTRACT-R et VEGA, nous utiliserons l'exemple de CELLULE FAMILIALE (cf. § I.4 du chapitre précédent).



a) Représentation dans ABSTRACT-R

Une forme possible de représentation de l'exemple CELLULE FAMILIALE est, en ABSTRACT-R :

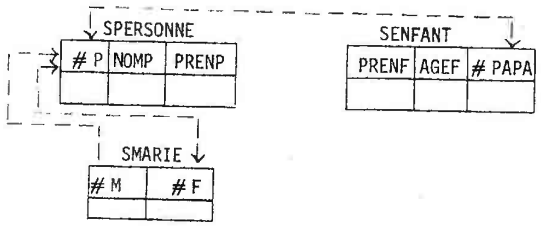
1) Définition de la structure d'une instance

```
DEFINE ABSTRACTION CELFAM
  AS #P1 : INTEGER
     #P2 : INTEGER
     ENFANT : LIST(STRING)
  WHERE #P1 IS SELECT # M FROM SMARIE WHERE #M = #P1
        AND #P2 IS SELECT #F FROM SMARIE WHERE #F = #P2
        AND ENFANT IS SELECT PRENF FROM SENFANT WHERE #PAPA=#P1
                                                  = #
```

Dans cette définition nous supposons que

- l'association MARIE est représentée par une collection SMARIE de tuples <#M, #F>, où #M désigne le numéro du mari et #F celui de l'épouse.
- l'association PATERNITE est représentée par l'insertion du numéro du père #PAPA dans la représentation de ENFANT

soit, schématiquement



- la clause AS de DEFINE ABSTRACTION définit la structure de CELFAM.
- la clause WHERE définit une correspondance entre les composants de la structure, et des constituants d'agrégats et/ou d'associations de base.

(Ainsi, #P1 est l'attribut #M dans la représentation de l'association MARIE, #P2 est l'attribut #F...).

La correspondance est décrite par des expressions du langage du système d'implémentation.

2) Représentation des opérations qui portent sur l'abstraction

On définira les opérations portant sur les abstractions, en termes d'opérations sur les représentations des agrégats et/ou associations de base.

Exemples :

(1)
```
DEFINE OPN NOUVEAU-NE(N:STRING,P:ENTEGER) ON CELFAM
  AS INSERT(N,0,P) INTO SENFANT
```

%On définit NOUVEAU-NE comme une insertion de l'enfant de PRENOM N, d'AGE 0 (zéro) et de PERE P%

On remarquera que cette définition permet de cacher certains effets à l'utilisateur (mise à zéro de l'AGE).

(2)
```
DEFINE OPN AINE(N:INTEGER) ON CELFAM RETURNS P:STRING
  AS SELECT PRENF FROM SENFANT
            WHERE #PAPA = N
            AND AGE = (SELECT MAX(AGE) FROM SENFANT)
```

[Cette définition fait appel à l'opération MAX (supposée appartenir à un type ENTIER) qui permet de rechercher le plus grand entier parmi un ensemble d'entiers].

Encore une fois, ce mode de représentation des opérations permet de :

- décrire des "en-tête" de procédures qui seront les seules parties visibles par l'utilisateur,
- et de définir les corps de ces procédures "cachant" ainsi à l'utilisateur la façon dont l'opération est implémentée.

Des mécanismes "ad hoc" du système doivent prendre en charge les transformations des expressions des utilisateurs en des expressions "terminales".

b) Représentation des abstractions dans VEGA

Dans sa version actuelle, VEGA offre des possibilités limitées pour représenter des abstractions. Ainsi, le fait que l'on ne puisse pas définir des TUPLES ayant des tuples comme constituants, fait que certaines abstractions ne peuvent pas être spécifiées en tant que SYSTEMes.

Néanmoins, leurs opérations peuvent être définies à l'aide d'expressions de l'interface VEGA-PROLOG.

Exemple

```
(1)  +MARIE(*H,*F) - FIRST((*P,*E) IN SMARIE : INT(*P =*H
                                  AND *E = *F) ;
                     PRINT(*H,*F))

(2)  +ENFT(*F) - EACH((*N,*A,*P) IN SENFANT:INT(*P =*F);
                     PRINT(*N)).
```

Ces deux expressions permettent respectivement de savoir si deux personnes (repérées par leurs numéros d'identification *H, *F) sont mariées (ie sont dans l'ensemble SMARIE) et d'obtenir [(2)], s'il en existe, la liste des enfants d'une personne de numéro *F.

Ces expressions sont alors utilisables pour exprimer les opérations de CELFAM.

- Ainsi la composition d'une cellule familiale pourra être obtenue par :

  +CELLULE(*F) - MARIE(*F,*X) - PRINT(*F,*X) - ENFT(*F)

- NOUVEAUNE sera exprimée en termes d'adjonction d'un élément dans l'ensemble SENFANT :

  +NOUVEAUNE(*N,*P) - TSENFANT(SENFANT ← ADD(SENFANT,(*N,0,*P)))

- La mère d'une famille est obtenue par l'opération MERE

  +MERE(*P) - MARIE(*P,*F) - PRINT(*F).

- ...

L'idée, pour la représentation des abstractions relationnelles, consiste donc, à exprimer les opérations sur les représentations des agrégats et/ou des associations que l'abstraction considérée invoque. Ces opérations s'utilisent alors, comme si elles agissaient sur des instances effectives des abstractions.

En somme, la représentation des abstractions relationnelles (comme les définitions en intention) permet de définir des règles pour inférer ces abstractions à partir d'objets de base.

II.2.5.- Cas des axiomes de la spécification

Nous avons déjà évoqué (partie B - chapitre III) le fait que les axiomes d'une spécification pouvaient être utilisés

- en tant que contraintes (au sens contraintes d'intégrité), pour certains,
- en tant que règles d'inférence en interrogation, ou
- en tant que règles de propagation en mise à jour, pour d'autres.

Nous nous sommes plus particulièrement intéressés au second usage. En ce qui concerne le premier, nous supposons que le système utilisé (ABSTRACT-R ou VEGA) offre des moyens pour exprimer

et préserver au moins les contraintes les plus simples (cf. clauses INVARIANT et RESTRICTO de VEGA), quant au troisième, nous ne nous y sommes pas intéressés ; il devrait faire l'objet d'approfondissements ultérieurs.

Ainsi, l'utilisation de certains axiomes en tant que règles d'inférence est un moyen d'expliciter le calcul logico-mathématique qui permet d'obtenir des informations non explicites dans la base à partir de celles qui y sont explicitement rangées.

L'exemple des loisirs à proximité des hébergements (HEBGT & LOISIRS) traité au § II;2.3.2 de ce chapitre en a été une illustration.

En effet l'axiome HEBGT & LOISIRS(H) = PAYS & LOISIRS(PAYS DE HEB(H)) exprime le fait que les loisirs près d'un hébergement sont ceux du pays où est situé l'hébergement.

Son utilisation en tant que règle d'inférence, consiste à définir la façon dont sera évaluée sa partie droite (par évaluation de "SELECT NOML FROM ... WHERE NOMH = H" par un système relationnel ou par l'exécution de la séquence VEGA correspondante).

II.2.6.- Conclusion

Dans le cadre de ce qui vient d'être présenté, il nous semble intéressant de rappeler que :

- la démarche suivie permet d'aboutir de façon progressive à l'expression d'une représentation d'une base abstraite à l'aide des outils (de représentation) offerts par le système logiciel d'implémentation. La caractéristique essentielle requise de ce dernier, est de permettre l'extension de ses structures de représentation par de nouvelles.
- En outre, les choix de représentation des collections d'objets (en intention, en extension) s'accompagnent de choix d'implémentation de leurs opérations. D'une façon assez simple, la démarche suivie (et les techniques sous-jacentes) permet d'obtenir les éléments nécessaires à la conduite de déductions (représentation des objets, de leurs collections, de leurs opérations, des compositions d'opérations et règles d'inférence), sous réserve que le système d'implémentation soit doté de mécanismes sachant exploiter ces éléments.
- enfin, cette approche par abstraction puis représentation permet de rendre "transparents" à l'utilisateur toutes les considérations de représentation et d'implantation. En effet, l'utilisation "presse-bouton" du système n'exigera de ses auteurs que la seule connaissance des aspects externes (en-tête des procédures) des opérations définies.

[Dans l'optique d'une programmation plus "sophistiquée", il sera nécessaire de disposer d'un langage "plus évolué" où les opérations définies peuvent être utilisées autant qu'instructions de ce langage].

Le paragraphe qui suit traite de l'utilisation de ces opérations pour mettre en place (créer) et utiliser une base de données.

III - MISE EN PLACE et UTILISATION D'UNE BASE DE DONNEES

La représentation d'une base étant définie, nous allons maintenant nous préoccuper de sa création puis de son utilisation.

Par analogie avec les langages de programmation supportant le concept de type abstrait de données, la phase de représentation est essentiellement une phase de réalisation de types. Les corps des programmes utiliseront des instances de ces types par le biais des opérations de ces types (création, modification, observation). Ces instances sont désignées par des identificateurs déclarés du type.

La mise en place d'une base de données physique sera effectuée selon un procédé similaire :

- déclaration de variables associées à des collections d'objets (agrégats, associations)
- création des collections.

L'utilisation de la base consistera alors à exécuter des opérations d'observation ou de modification de ces collections.

Le caractère déductif de la base est déterminé par le fait que les résultats des opérations (essentiellement d'observation) ne proviennent pas exclusivement de l'observation des collections. Ils peuvent être issus de l'application de règles d'inférence (définissant les intentions, les abstractions relationnelles...).

III.1.- La mise en place d'une base de données

Dans une approche relationnelle, la définition suivante :

RELATION HEBERGEMENT(NOMH : STRING,CAPACITE:INTEGER,CATEGORIE: INTEGER)

couvre à la fois les notions :

(a) - de type tuple THEB(NOMH:STRING,CAPACITE:INTEGER, CATEGORIE:INTEGER)
(b) - de collection de tuples TSETHEBERGEMENT = SET(THEB)
(c) - et la déclaration d'une variable HEBERGEMENT de type TSETHEBERGEMENT.

Dans le cadre de notre démarche, ces aspects sont différenciés. Après la définition des types faite au § II [qui regroupe (a) et (b)], l'instanciation d'une base de données physique consistera à déclarer des variables (notion (c)) d'entités, d'associations,...,avant de pouvoir les utiliser.

III.1.1.- Application à ABSTRACT-R

ABSTRACT-R doit prendre en compte le type de déclaration : DECLARE HEBERGEMENT : TSETHEBERGEMENT, qui équivaut à la déclaration d'une RELATION HEBERGEMENT au sens traditionnel du terme (ie comme une collection de tuples).

Dans notre système, les définitions de TSETHEBERGEMENT, en tant que SET (ou RELATION) OF THEB, et de THEB en tant que TUPLE (NOMH : STRING, CAPACITE : ENTIER, CATEGORIE : ENTIER) suffisent à définir HEBERGEMENT en tant que RELATION.

L'instanciation de la relation sera produite par l'application d'une opération d'initialisation (S-INIT dans la spécification du type ENTITE) suivie d'une succession d'opérations d'insertion.

Exemple :
```
HEBERGEMENT ← { }
INSERT ('LE NID',4,2) INTO HEBERGEMENT
INSERT ('LES PINGOUINS',3,3) INTO HEBERGEMENT
  :
```

L'opération INSERT utilisée est supposée être une opération du système, ou plus exactement du type représentant la RELATION. Si nous avions défini une opération AJOUTHEB(H) AS INSERT H INTO TSETHEBERGEMENT, nous aurions pu l'utiliser. Sa "traduction" en

INSERT H INTO HEBERGEMENT,

serait du ressort de ABSTRACT-R.

REMARQUE :

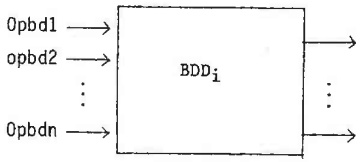
1 - Les mécanismes que doit comporter le système pour supporter ce type de démarche peuvent être complexes :
- instanciation d'objets d'un type donné

- mise en oeuvre de processus de contrôle des occurrences
- assurer la protection des objets, en n'autorisant que l'application des seules opérations définies sur leur type.

2 - Sur un plan des principes, rien ne s'oppose à ce que l'on déclare plusieurs variables d'un même type. On aurait ainsi plusieurs instances d'un même type de relation.

3 - Selon le même principe, le système devrait autoriser la déclaration de variables bases de données. Cette variable est du type du schéma de la base.

On aboutit ainsi à la concrétisation de nos objectifs en matière d'abstraction : la base de données, identifiée par sa variable, serait alors, observée et/ou modifiée par l'utilisateur. Elle se comporterait ainsi comme "la boite noire" sur laquelle il appliquerait les opérations définies.



III.1.2.- Application à VEGA

Reprenons les types TSETHEBERGEMENT et TSETSITUATION des § II. 2.2 et II.2.3 de ce chapitre.

* La déclaration d'une variable SH du type TSETHEBERGEMENT, et son initialisation s'expriment par

  - RUN(TSETHEBERGEMENT(SH ← 0))?

    SH est initialisé à vide par "← 0", où ← symbolise l'affectation et "0" le vide.

* L'adjonction de tuples s'exprime à l'aide de l'opération de modification ADD.
  Ainsi -RUN(TSETHEBERGEMENT(SH ← ADD(SH,(LE NID;4;2))))?
  a pour effet d'ajouter à SH le tuple (LE NID,4,2).

* De la même façon,

  - RUN(TSETSITUATION(ST ← 0))? initialise à vide un ensemble ST de SITUATIONs
  - RUN(TSETSITUATION(ST ← ADD(ST,(MARGERIDE;LE NID))))? y introduit la situation de "LE NID" en "MARGERIDE".

REMARQUES :

1 - On peut, ici aussi, utiliser plusieurs variables d'un même type. Ainsi,
  - RUN(TSETHEBERGEMENT(SH2 ← 0)) définirait un ensemble SH2 d'hébergements.

2 - Les mécanismes de contrôle de VEGA permettent, lors de l'évaluation de telles expressions, de vérifier les contraintes éventuellement apposées sur des types par le biais des clauses INVARIANT et RESTRICTO.
  Ainsi, si une clause RESTRICTO (*C : 1 TO 5) est mentionnée dans le SYSTEMe TCAT qui définit la CATEGORIE d'un hébergement,
  - RUN(TSETHEBERGEMENT(SH ← ADD(SH,(LES PINGOUINS,4,10))))? produit une erreur sur TCAT.

Donc, le processus de mise en place d'une base de données physique peut se résumer en
- la déclaration des collections d'objets à mémoriser (SH,ST,SH2..)
- la création effective des collections à l'aide des opérations d'adjonction d'éléments à des collections.

Cette étape, généralement appelée changement initial, consiste à mémoriser, sous forme de données, l'état du monde observé un instant précis. Cet état sera ensuite modifié en fonction des changements du monde observé, afin d'en être, à tout instant, une image "fidèle".

L'application, sur les états successifs de ce monde, des opérations d'observation et/ou de modification constitue l'utilisation (la manipulation des données) de la base.

III.2.- Manipulation de données et mécanisme de déduction

Nous avons vu, au chapitre I/A, que la manipulation de données nécessitait un langage permettant d'exprimer

- l'action à effectuer (recherche, ajout...)
- la caractérisation des éléments de la base sur lesquelles s'effectue l'action.

Dans les systèmes traditionnels, les actions possibles sont celles offertes par le LMD des systèmes en question :

- ajout d'éléments (par des commandes INSERT ou STORE ou ..)
- suppression (DELETE)
- modification (UPDATE ou MODIFY ou ...)
- recherche (SELECT ou RETRIEVE ou FIND ou ...).

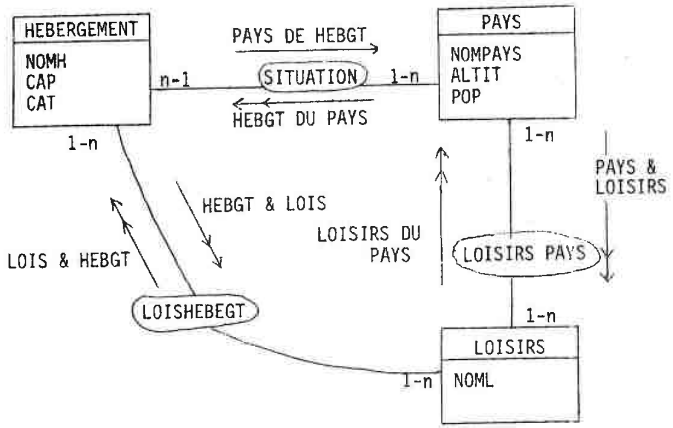
Dans une approche par les types abstraits, les actions sont les opérations de la spécification. Ceci nous semble présenter un double intérêt :

- d'une part, on dispose d'un ensemble d'actions plus riche, puisqu'on ne se limite pas aux seules actions offertes par le système d'implémentation.
- d'autre part, on dispose d'un vocabulaire plus "expressif" pour l'utilisateur. En effet, un choix judicieux des symboles de la spécification permet de déterminer un langage de l'application proche de celui utilisé couramment dans le domaine. (En d'autres termes, les symboles sont choisis de telle sorte à exprimer la sémantique intuitive de ce qu'ils désignent ; la spécification définit leur sémantique de façon plus formelle).

Dans ce paragraphe, nous allons brièvement exposer le principe de résolution de PROLOG, puis nous présenterons des utilisations ("naïves" et "moins naïves") d'une base de données déductive à l'aide de VEGA ou de son interface avec PROLOG.

III.2.1.- Le principe de la résolution

Sans vouloir entrer dans les détails de la réalisation de VEGA, nous allons essayer de montrer le principe de fonctionnement du système, en nous aidant de l'exemple de schéma ci-dessous. Nous nous baserons sur son expression sous forme de clauses PROLOG.



III.2.1.1.- Une description de l'exemple

Les clauses ci-dessous décrivent, en PROLOG, l'ensemble des occurrences de

- HEBERGEMENT [clauses (1) à (3)]

- PAYS [(4) à (6)]
- LOISIRS [(14) à (17)]
- SITUATION [(7) à (9)]
- LOISIRS PAYS [(10) à (13)]

[Nous supposons que SORTIR fait partie des "prédicats évaluables" de PROLOG].

(1) + HEBERGEMENT(LENID,4,2)
(2) + HEBERGEMENT(FAUCONS,5,3)
(3) +HEBERGEMENT(PINGOUIN,10,2)
(4) +PAYS(MARGERIDE,1000,2500)
(5) +PAYS(VALLEEDULOT, 1800,6000)
(6) +PAYS(MARJEVOLS,1500,3900)
(7) +SITUATION(MARGERIDE,PINGOUINS)
(8) +SITUATION(MARGERIDE,LENID)
(9) +SITUATION(MARJEVOLS,FAUCONS)
(10) +LOISIRSPAYS(MARGERIDE,SKI DE FOND)
(11) +LOISIRSPAYS(MARGERIDE,PECHE)
(12) +LOISIRSPAYS(MARGERIDE,RANDONNEE)
(13) +LOISIRSPAYS(MARJEVOLS,CANOE KAYAK)
(14) +LOISIRS(CANOE KAYAK)
(15) +LOISIRS(RANDONNEE)
(16) +LOISIRS(PECHE)
(17) +LOISIRS(SKI DE FOND)
(18) +NOMH(*X)-HEBERGEMENT(*X,*1,*2)-SORTIR(*X)
(19) +CAPACITE(*X)-HEBERGEMENT(*X,*Y,*1)-SORTIR(*Y)
(20) +CATEGORIE(*X)-HEBERGEMENT(*X,*1,*Y)-SORTIR(*Y)
(21) +PAYSDEHEBGT(*H)-SITUATION(*P,*H)-SORTIR(*P)
(22) +HEBGTDUPAYS(*P)-SITUATION(*P,*H)-SORTIR(*H)
(23) +LOISIRSDUPAYS(*P)-LOISIRSPAYS(*P,*L)-SORTIR(*L)
(24) +PAYS&LOISIRS(*L)-LOISIRSPAYS(*P,*L)-SORTIR(*P)
(25) +LOISIRSHEBGT(*H,*L)-SITUATION(*P,*H)-LOISIRSPAYS(*P,*L)
(26) +HEBGT&LOIS(*H)-LOISIRSHEBGT(*H,*L)-SORTIR(*L)
(27) +LOIS&HEBGT(*L)-LOISIRSHEBGT(*H,*L)-SORTIR(*H)

Les clauses (18) à (20) décrivent respectivement :

NOMH	: HEBERGEMENT	→	STRING
CAPACITE	: HEBERGEMENT	→	ENTIER
CATEGORIE	: HEBERGEMENT	→	ENTIER
PAYSDEHEBGT	: HEBERGEMENT	→	PAYS
HEBGTDUPAYS	: PAYS	↠	HEBERGEMENT
LOISIRSDUPAYS	: PAYS	↠	LOISIRS
PAYS&LOISIRS	: LOISIRS	↠	PAYS

La clause (25) décrit l'association LOISHEBGT par le biais des associations SITUATION et LOISIRSPAYS (c'est-à-dire en intention).

Les clauses (26) et (27) décrivent les deux "fonctions rôles" associées à LOISHEBGT.

III.2.1.2.- L'expression de requêtes

Cet ensemble de clauses peut être utilisé selon deux modes :

a) le mode implicite

Il correspond au mode "presse-bouton".
Dans ce cas, une requête est de la forme

- P(*X,...,a,...,*Z,...,b)

où P est un littéral qui apparaît en partie gauche d'une des clauses qui décrivent le schéma et (certaines de) ses opérations. Certaines variables, arguments de P, peuvent être instanciées par des constantes (a,b...), d'autres pas (*X,*Z,...)

Exemples

(1) - "Quels sont les hébergements de la base" s'exprime par :

- HEBERGEMENT (*X,*Y,*Z)

(2) - "Quels sont les hébergements de catégorie 2*" s'exprime par

- HEBERGEMENT (*X,*1,2)

(3) - "Quels sont les loisirs à proximité de l'hébergement LE NID" s'écrit :

- HEBGT & LOIS (LE NID)

b) Le mode explicite

Il sert pour des utilisations "moins naïves" du système. Dans ce mode, une requête a la forme

- P1(<liste de variables et/ou constantes>)-...-Pn(<liste var. et/ou cstes>).

où les Pi sont des littéraux qui apparaissent en partie gauche des clauses initiales.

Exemples

(1) L'exemple (3) écrit "-HEBGT&LOIS(LENID)" en mode implicite, s'exprime, en mode explicite, par

- SITUATION(*X,LENID)-LOISIRSPAYS(*X,*L)-SORTIR(*L)

(2) "Quels sont les hébergements de catégorie 2* à proximité desquels on peut faire du SKI DE FOND" s'exprime sous la forme :

- HEBERGEMENT(*X,*1,2)-SITUATION(*Y,*X)
  -LOISIRSPAYS(*Y,SKIDEFOND)-SORTIR(*X)

ou encore

(2') - HEBERGEMENT(*X,*1,2)-LOIRSIRSHEBGT(*X,SKIDEFOND)
  -SORTIR(*X)

NOTE : On fera remarquer que certaines clauses de l'ensemble de départ [comme celles de (21) à (24)], sont superflues. En effet, "l'expression relationnelle" qu'autorise PROLOG permet d'éviter l'explosion combinatoire des expressions fonctionnelles. Ainsi,

- SITUATION(*P,a) "remplace" PAYSDEHEBGT(a), où a est un nom d'hébergement. De la même façon,
- SITUATION(b,*H) "remplace" HEBGTDUPAYS(b), où b est un nom de pays ...

III.2.1.3.- L'évaluation des requêtes

L'évaluation des requêtes se fera par une succession d'étapes de résolution (cf. § V.3.3/II/A) comme le montrent les exemples ci-après.

Exemple 1 :

(0) - HEBERGEMENT(*H,*C,*T)-SORTIR(*H)-LIGNE
PROLOG va successivement unifier (0)

- avec (1), à l'aide de la substitution {*H/LENID,*C/4,*T/2}
- avec (2) par {*H/FAUCONS,*C/5,*T/3}
- et avec (3) par {*H/PINGOUINS,*C/10,*T/2}.

Il produira alors les réponses :

LE NID
FAUCONS
PINGOUINS

Exemple 2 :

(0) - HEBEGT&LOISIRS(LE NID)

[1] Il y a unification de (0) et (26) par {*H/LENID}

Le littéral HEBGT&LOISIRS est "remplacé" par la partie droite de la clause (26) dans laquelle on opère également la substitution {*H/LE NID }

[2] - LOISHEBGT(LENID,*L)-SORTIR(*L)
Le même mécanisme que [1] est alors répété :

[3] - SITUATION(*P,LENID)-LOISIRSPAYS(*P,*L)-SORTIR(*L)
en utilisant (25) et {*H/LENID}

[4] - LOISIRSPAYS(MARGERIDE,*L)-SORTIR(*L)
en utilisant (8) et {*P/MARGERIDE}

[5] - SORTIR(SKI DE FOND) en utilisant (10) et {*L/SKI DE FOND}
[SORTIR provoque l'impression de SKI DE FOND]

[5'] PROLOG explorant toutes les solutions possibles, on obtient également :

[5'] - SORTIR(PECHE) en utilisant (11) et {*L/PECHE}

[5"] - SORTIR(RANDONNEE) en utilisant (12) et {*L/RANDONNEE}

Ces deux exemples permettent d'illustrer nos propos en matière de déduction.

Le premier est un exemple d'évaluation sur une entité définie en extension, alors que le second opère sur une association (LOIS-HEBGT) définie en intention, sur les extensions de LOISIRS PAYS et SITUATION.

D'un autre côté, une rapide comparaison entre PROLOG et VEGA permet de montrer que ce dernier autorise :

- le regroupement de clauses dans une collection
  Ainsi, sur l'exemple, les clauses (1) à (3) constitueraient l'ensemble SH de type SET(THEB), les clauses (4) à (6) l'ensemble PA de type SET(PAYS)...
- l'application d'opérations sur ces collections

- la définition et l'utilisation de collections définies en intention
- l'association de types aux arguments des littéraux
- l'expression de requêtes tant dans le mode implicite que dans le mode explicite.

Tous ces points font de VEGA et de son interface avec PROLOG, un outil très adapté pour l'élaboration de maquettes de base de données déductives.

III.2.2.- Manipulation de données en VEGA : quelques exemples

1) Le mode implicite

```
(1)  + HEBERGEMENT-EACH((*H) IN TSETHEB(SH));PRINT(*H))

(2)  + HEBEGT&LOISIRS(*X)-FIRST((*P,*H) IN TSETSITUATION(ST):
                            STRING(*H = *X));
                            EACH((*P2,*L) IN TSETLOISPAYS(LP):
                               STRING(*P = *P2));
                            PRINT(*L);LINE)
```

(1) permet d'obtenir tous les hébergements de l'ensemble SH (du type TSETHEB)
(2) sort tous les loisirs à proximité d'un hébergemert *X.

L'utilisation de ces opérations consistera, tout simplement, en l'indication de l'opération et des valeurs des paramètres éventuels.

Ainsi - HEBGT&LOISIRS (LE NID) permet d'obtenir tous les loisirs à proximité de l'hébergement LE NID.

Ce procédé de mise en oeuvre est dédié, essentiellement, aux utilisateurs naïfs du système.

2) Le mode explicite

(1) Trouver un hébergement pour 4 personnes de catégorie 2 étoiles

```
- MRUN(FIRST(*H IN TSETHEB(SH):INT(THEB(CAPACITE(*H)=4);
      INT(THEB(CATEGORIE(*H)=2));PRINT(*H)))?
```

(2) Trouver tous les hébergements de 3 personnes, à plus de 1000 m d'altitude et tels que l'on puisse faire du SKI DEFOND à proximité d'eux.

```
- MRUN(EACH((*P,*A,*G) IN TSETPAYS(SP):INT(*A>1000));
      EACH((*P2,*L2) IN TSETLOISPAYS(LP):STRING((*P=*P2)
      &(*L2=SKI DE FOND));
      EACH((*P1,*H1) IN TSETSITUATION(ST):(STRING(*P=*P1));
      EACH(*H IN TSETHEB(SH):INT(THEB(CAPACITE(*H)=3);
                  PRINT(*H)))?
```

IV - CONCLUSIONS

Dans ce chapitre, nous avons montré comment passer, progressivement d'une spécification de base abstraite à son implantation. Nous avons insisté sur les capacités attendues du système logiciel d'implantation, à savoir, permettre :

- l'introduction de nouveaux types de données
- l'utilisation de ces types et de leurs opérations dans les requêtes ou programmes, au même titre que les types de base du système.
- l'expression intentionnelle de collection d'agrégats.
- et enfin posséder des mécanismes de déduction sachant exploiter des axiomes de la spécification (ou des définitions en intention) comme règles d'inférence.

Nous avons essayé de montrer que ces deux derniers points conféraient au système un caractère déductif "transparent" pour ses

utilisateurs. En d'autres termes, un "utilisateur naïf" du système pourra tout ignorer de la représentation de la base et de l'implantation des opérations. Il ne "percevra" que le comportement externe du système.

Il va sans dire que le choix des structures de représentation et la transformation des expressions abstraites (ie celles en termes du vocabulaire de la spécification) en des expressions concrètes (ie des expressions sur la structure de représentation) nécessitent l'intervention d'un utilisateur moins naïf (pourquoi pas un outil automatisé ?...) qui, lui, devra avoir connaissance du langage du système d'implémentation.

ABSTRACT-R et VEGA nous ont permis d'illustrer nos propos.

ABSTRACT-R est un système relationnel "imaginaire". Nous avons essayé de montrer, par son biais, quels doivent être les outils supplémentaires à offrir à un utilisateur (prendre en compte des DEFINE TYPE, OPN, ABSTRACTION...) et les mécanismes internes que devait posséder un SGBD traditionnel pour satisfaire nos aspirations.

VEGA, système expérimental de spécification et de vérification de types, a déjà été utilisé pour réaliser des maquettes de bases de données ([CHAB 82], [BOUD 82]). Nous avons essayé de montrer l'usage qui pouvait en être fait pour la réalisation de bases de données dites déductives. Les facilités qu'il offre pour utiliser ou construire des types, ainsi que les possibilités de son interface avec PROLOG, facilitent grandement la tâche pour réaliser des maquettes [CHAB 83] de systèmes de données déductifs.

Mais, dans le cas d'applications "en grandeur réelle", il faudra se préoccuper (plus que nous ne l'avons fait jusqu'à maintenant)

- des problèmes de performances

Le moins que l'on puisse dire, c'est qu'elles sont assez basses actuellement, compte tenu de la "gourmandise" en temps et en

espace de la version de PROLOG utilisée.

- de ceux de mise à jour de la base, que nous n'avons pas abordé dans cette phase expérimentale.
- enfin, de l'amélioration du "confort d'utilisation" du système (syntaxe du langage moins ardue, possibilités de formatage des résultats, aides automatisées (ou semi-automatisées) à la mise en place et à l'utilisation...).

Le dernier volet de notre travail est en rapport avec ce confort d'utilisation. Il fait l'objet du chapitre qui suit.

CHAPITRE IV : SINDBAD ET LA COMPRÉHENSION DE REQUÊTES

I - INTRODUCTION

Nous avons voulu privilégier deux traits parmi ceux qui caracrisent un système dit "intelligent" :

- le premier que nous avons traité dans les deux chapitres précédents concerne la capacité du système à fournir aux utilisateurs une masse d'informations plus grande que celle mémorisée : c'est l'aspect déductif du système.
- le second, que nous abordons dans ce chapitre, concerne les facilités d'utilisation d'un tel système et ses capacités de communication avec un utilisateur qui ne serait pas un spécialiste de l'informatique.

Le but visé, dans cette partie, est triple :

- dégager l'utilisateur des détails de représentation en lui donnant la possibilité d'exprimer ses besoins à l'aide du vocabulaire de la spécification abstraite.
- autoriser des expressions incomplètes de requêtes.
- faire dialoguer l'utilisateur avec le système selon le procédé de communication décrit ci-dessous.

Soit deux interlocuteurs A (l'utilisateur) et B (le système).

(1) - A exprime ses besoins en informations sous forme d'une suite de propositions (ou bribes)

ex : "Je voudrais un hébergement pour 15 jours ; nous sommes 4 ; on aimerait pouvoir faire du ski de fond"

(2) - B accuse réception de la demande puis essaie de relier les bribes. Pour cela, il tente d'expliciter des sous-entendus, en utilisant ses connaissances sur le domaine.

ex : (a) A voudrait louer un hébergement.
(b) La capacité de l'hébergement doit être égale à 4.
(c) La durée de la location est de 15 jours.
(d) A voudrait faire du ski de fond à proximité de son lieu d'hébergement.

(3) - Après cela, B soumet ces différentes interprétations à A pour s'assurer de la correction de la compréhension.

(4) - Si c'est le cas, B va essayer de trouver la/les réponses qui pourraient satisfaire A.

(5) - Sinon, B va essayer, seul ou avec l'aide de A, de "mieux comprendre" le problème qui lui est posé.

Le principal avantage attendu de la concrétisation de tels mécanismes, est de faire en sorte que l'utilisation naïve d'une base devienne plus qu'une utilisation "presse-bouton". Nous montrerons par ailleurs, que l'on pourrait également en tirer profit durant la phase de conception.

II - PRINCIPES GENERAUX DU SYSTEME

II.1.- Principes de fonctionnement

L'expression d'une requête de manipulation de données comporte généralement :

- la nature de l'action à effectuer (interrogation ou modification)
- la désignation des données auxquelles s'applique cette action

- et le chemin d'accès aux données de la base.

Exemple :

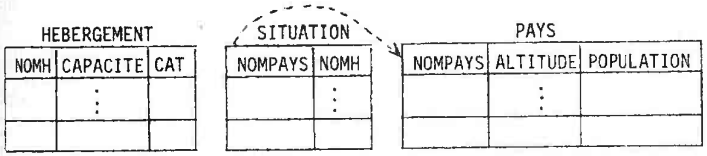
En langage relationnel, "Quelle est l'altitude du pays de l'hébergement H" s'exprimerait de la manière suivante :

(1) SELECT ALTITUDE FROM PAYS
WHERE NOMPAYS = SELECT NOMPAYS FROM SITUATION
WHERE NOMH = 'H'

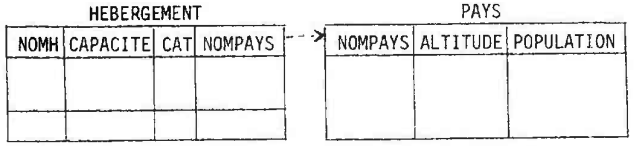
ou (2) SELECT ALTITUDE FROM PAYS
WHERE NOMPAYS = SELECT NOMPAYS FROM HEBERGEMENT
WHERE NOMH = 'H'

selon que l'association SITUATION a été représentée par une relation SITUATION(NOMPAYS, NOMH) ou par l'adjonction de NOMPAYS à la relation HEBERGEMENT.

Dans un cas comme dans l'autre, la requête utilise la représentation de l'association SITUATION pour exprimer le "chemin" (symbolisé par des pointillés sur la schématisation ci-dessous).



- CAS 1 -



- CAS 2 -

Sur un autre plan, l'expression de la requête mentionne explitement la localisation des attributs concernés au sein des relations [nompays et nomh dans SITUATION, ALTITUDE dans PAYS et jointure de SITUATION et PAYS par NOMPAYS dans l'expression (1) ou NOMH et nompays dans HEBERGEMENT et jointure de HEBERGEMENT et PAYS par NOMPAYS dans (2)].

Le problème que nous étudions dans ce chapitre est celui de la conception d'un système qui, étant donné un ensemble de bribes d'informations exprimées dans une requête, soit capable de déterminer automatiquement ou avec l'aide de l'utilisateur, à la fois la localisation des données et le chemin pour y accéder.

En bref, nous voudrions disposer d'un système qui, étant donné, par exemple, ALTITUDE et 'H' soit capable de générer l'expression (1) ou (2) selon le cas et, ce, en limitant au maximum l'intervention humaine durant tout le processus.

Exemple : Si l'on fournit au système les indications suivantes :

SORTIR ALTITUDE : HEBERGEMENT 'H',

il devrait être capable d'abord de reconnaître les bribes d'informations du genre :

- ALTITUDE est un attribut de PAYS
- 'H' est un NOM D'HEBERGEMENT

puis de les relier dans une "interprétation plausible" :

SORTIR ALTITUDE DU PAYS OU SE SITUE L'HEBERGEMENT 'H'

L'interprétation est construite en utilisant la connaissance qu'a le système sur le domaine. Elle s'exprimera en termes du vocabulaire connu sur ce domaine.

Si l'interprétation obtenue convient à l'utilisateur, elle est alors évaluée par le logiciel de gestion de la base.

Le système que nous avons réalisé se limite, pour le moment, à

la production de l'interprétation (ou compréhension) de la requête. La conception d'un "système complet" consisterait à développer un traducteur de l'expression de la requête "interprétée" en une expression du langage de manipulation de données du SGBD.

[Nous avons néanmoins réalisé un traducteur qui produit des expressions PROLOG associées aux requêtes interprétées. Ce traducteur n'est pas général, au sens où il est lié à une représentation particulière de la base. Nous donnerons au paragraphe IV de ce chapitre des indications sur la façon dont un traducteur général pourrait être réalisé].

La partie de SINDBAD qui traite la résolution de ce problème (le sous-système de COMPREHENSION) suppose que l'on ait préalablement acquis :

- la spécification abstraite (à l'aide du sous-système DEFDB)
- la description de la base concrète, résultant de la phase d'élaboration de la représentation de la base
- l'implantation des diverses opérations (simples ou dérivées) de la spécification.

II.2.- Caractéristiques du langage d'expression des requêtes

II.2.1.- Caractéristiques

Le langage d'expression de requêtes que nous avons implémenté, présente les caractéristiques suivantes :

a) Les bribes peuvent être énoncées dans n'importe quel ordre.

Le seul point qui est impératif est d'indiquer les données recherchées en tête de la requête.

Exemple : Les requêtes suivantes :

SORTIR HEBERGEMENT : ALTITUDE "1200", LOISIRS "SKI DE FOND", CAPACITE "4".
SORTIR HEBERGEMENT:ALTITUDE "1200","CAPACITE 4",LOISIRS"SKI DE FOND".
SORTIR HEBERGEMENT:LOISIRS "SKI DE FOND",CAPACITE "4", ALTITUDE "1200"

seront interprétées de la même façon.

b) Des "tournures de phrases" courantes et certaines synonymies sont autorisées.

Exemple :

- L'HEBERGEMENT "H" ou l'HEBERGEMENT de NOM "H" sont considérées comme synonymes.
- ALTITUDE ("MARGERIDE") et ALTITUDE(PAYS "MARGERIDE") également.

c) Tout élément du vocabulaire produit par la spécification est utilisable. Ce dernier point nous semble important car il fournit un moyen pour exprimer des requêtes sous une forme proche du "jargon" de l'utilisateur.

Nous avons en effet vu qu'à l'aide de mécanismes simples, il

était possible de rendre disponible un vocabulaire propre au domaine de l'application.

Exemple : Dans la spécification abstraite, apparaissent les opérations AJOUTER HEBERGEMENT et RESERVER.
La représentation les a définit dans le langage du logiciel d'implantation

. rep(AJOUTER HEBERGEMENT(h)) = INSERT h INTO SHEB
. rep(RESERVER(h,c,d)) = INSERT c INTO SCLIENT
INSERT(h,c,d) INTO SRESERVATION

où SHEB, SCLIENT, SRESERVATION sont les relations qui représentent HEBERGEMENT, CLIENT et l'association RESERVATION.

Les opérations AJOUTERHEBERGEMENT et RESERVER sont utilisables en tant que telles dans l'expression des requêtes.

d) Il permet de confondre le nom de la fonction d'accès et le nom de l'attribut auquel est associée la fonction.

Exemple : Soit FNOMH : HEBERGEMENT → NOMH

Les expressions FNOMH(HEBERGEMENT) et NOMH(HEBERGEMENT) sont équivalentes.

e) Il ne fait pas obligation d'indiquer la localisation d'un attribut qui apparaît dans la requête.

Exemple : ALTITUDE sera reconnu comme ALTITUDE (PAYS)
NOMH comme NOMH (HEBERGEMENT)

[Ce dernier point suppose que les symboles utilisés sont uniques. Par exemple, un symbole d'attribut ne peut pas être utilisé dans des agrégats différents. Un petit effort de programmation suffirait, néanmoins, à lever cette contrainte].

Exemple : Si NOM apparaît en tant qu'identificateur d'attribut dans CLIENT et dans PAYS et qu'il est rencontré dans une requête, un dialogue qui s'établirait entre l'utilisateur et le système permettrait de lever l'ambiguïté.

II.2.2.- Structure d'une requête

Comme nous ne nous intéressons, pour le moment, qu'à l'interrogation de la base, nous ne considèrerons que la commande SORTIR (seule commande actuellement disponible dans SINDBAD). Sa syntaxe est la suivante :

SORTIR <liste de "cibles"> : <liste de sélecteurs>.

où
- la liste de cibles désigne la liste des éléments dont on recherche les valeurs.
- la liste des sélecteurs est la donnée "en vrac" des propriétés que doivent satisfaire ces éléments. Pour valider l'idée d'un système de compréhension de requêtes, il ne nous a pas paru primordial, dans un premier temps, de considérer des expressions complexes de sélecteurs. Au paragraphe IV.3 de ce chapitre, nous essayons de montrer que l'extension du langage à de telles expressions ne change pas fondamentalement ce qui nous paraît être l'essentiel du sous-système de compréhension de requêtes.

Exemples :

(1) SORTIR HEBERGEMENT, PAYS : NATUREHEBERGEMENT "HOTEL"

(2) SORTIR NOML, NOMH : HEBGT&LOISIRS(HEBERGEMENT),SITUATION (HEBERGEMENT,"MAGERIDE")

(3) SORTIR LOISIRS : PAYS "MARGERIDE".

Ces requêtes expriment respectivement :

- Trouver les hébergements de type HOTEL et les pays où ils se situent.

- Trouver les loisirs et les hébergements tels que l'on puisse s'adonner à ces loisirs à proximité des hébergements recherchés (sémantique de HEBGT&LOISIRS) et que ces hébergements soient situés en "MARGERIDE".
- Trouver les loisirs auxquels on peut s'adonner en "MARGERIDE".

[La syntaxe complète du langage est donnée en ANNEXE V].

II.3.- Compréhension de requêtes vs dérivation relationnelle

II.3.1.- La compréhension

L'objet de la compréhension est d'aboutir à une interprétation plausible de la requête, en éliminant les ambiguïtés et les synonymies et en explicitant les "sous-entendus".

Concrètement, le système de compréhension essaie d'établir une association (directe ou à l'aide d'une succession d'associations) entre les agrégats cités ou tout simplement invoqués dans une requête. Ceci revient, en quelque sorte à expliciter la "sémantique de la requête".

On dira que le système complète la requête. Dans la suite, on parlera indifféremment de COMPLETION ou de COMPREHENSION. D'autre part, nous utiliserons une opération de composition d'associations définie comme suit :

Soit R1 une association entre A et B, et R2 une association entre B et C, la composition de R1 et de R2 est une association R entre A et C telle que :

$R^+ : A \rightarrow C$

$R^+(a) = R2^+(R1^+(a))$

avec $R1^+ : A \rightarrow B$ ($R1^+$ est une des fonctions rôles définies sur R1).

$R2^+$: B → C ($R2^+$ est aussi une des fonctions rôles sur R2).

On notera la composition de R1 et de R2 R1.R2 ou encore

A.R1.B et B.R2.C

La complétion d'une requête vise alors à déterminer le sous-schéma de la base dans lequel figurent les agrégats et les associations utilisés dans la requête.

Autrement dit, si l'on note $\mathcal{R}(C,S)$ une requête utilisateur où C désigne les cibles et S les sélecteurs, le processus de compréhension consiste alors, à expliciter la sémantique de $\mathcal{R}$ à partir de la connaissance qu'il a sur le domaine sur lequel s'applique $\mathcal{R}$ et de règles d'utilisation de cette connaissance.

Les règles d'utilisation concernent essentiellement les modalités de composition des associations.

Les mécanismes que met en oeuvre le processus de complétion seront peut être mieux explicités grâce à la notion de dérivation relationnelle que nous allons définir maintenant.

II.3.2.- Notion de dérivation relationnelle

Elle est définie en termes de dépendances de données et de règles qui lient ces dépendances.

Nous avons vu (chap. I/A) que les diverses dépendances de données (cf. ANNEXE II) utilisées dans une approche relationnelle, permettent d'exprimer à la fois des contraintes et des propriétés sémantiques.

Dans notre approche, nous pouvons caractériser une association en termes de deux dépendances réciproques l'une de l'autre et auxquelles on associe un nom.

Exemple : L'association SITUATION qui lie HEBERGEMENT et PAYS peut être exprimée par une RELATION (HEBERGEMENT, PAYS) sur laquelle portent :

- une dépendance fonctionnelle : HEBERGEMENT → PAYS dénommée PAYSDEHEBGT
- et une dépendance multivaluée : PAYS →→ HEBERGEMENT dénommée HEBGTDUPAYS.

(HEBERGEMENT désigne la liste d'attributs {NOMH,CAPACITE, CATEGORIE} et PAYS la liste {NOMPAYS, ALTITUDE, POPULATION}).

Dans une approche relationnelle, les dépendances servent essentiellement à la décomposition et à la normalisation des relations. (Précisons que dans notre approche, l'expression de la représentation de la base correspond à l'expression d'un schéma relationnel après décomposition où les relations sont "au moins" en 3NF).

Par ailleurs, les dépendances peuvent être utilisées pour découvrir de nouvelles dépendances. Dans ce cas, on utilise les règles qui lient les dépendances (cf. ANNEXE II) comme des règles de dérivation.

Exemple de règle : Transitivité des dépendances fonctionnelles

si X → Y et Y → Z alors X → Z
où X,Y,Z sont des listes de constituants d'une relation R.

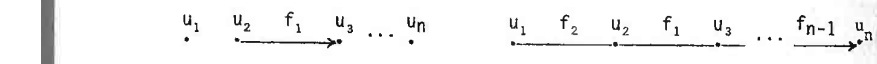
Si nous considérons le schéma $\mathcal{S}$ de la base comme une relation $\mathcal{S}(\mathcal{A},\mathcal{F})$ (la relation universelle) où $\mathcal{A}$ est l'ensemble des agrégats et $\mathcal{F}$ l'ensemble des dépendances exprimées sur les associations, nous pouvons alors exprimer l'analogie qui existe entre la complétion telle que nous l'avons décrite et la dérivation relationnelle.

Considérons une requête $\mathcal{R}$ comme une relation R(U,F) où U désigne l'ensemble des agrégats $u_1,u_2,\ldots,u_n$ invoqués par la requête et F l'ensemble des dépendances $f_1,f_2,\ldots,f_m$ qu'expriment

les associations invoquées dans $\mathcal{R}$.

La complétion est un processus qui vise à trouver une composition de dépendances qui exprimerait la sémantique de la requête. Cette composition de dépendances s'établit en utilisant les règles de dérivation de dépendances et les dépendances initiales (celles de $\mathcal{F}$).

Schématiquement, le processus de complétion vise à transformer la figure 1 ci-dessous en la figure 2 (avec $U = \{u_1, u_2, \ldots, u_n\}$ et $F = \{f_1\}$)



- Figure 1 -

- Figure 2 -

Exemple : Considérons la requête SORTIR HEBERGEMENT : NOML"SKI DE FOND"
Elle invoque directement HEBERGEMENT et "indirectement" "LOISIRS".

U = {HEBERGEMENT, LOISIRS} et $F = \{\}$

Or, nous disposons des dépendances :

(DF1)	PAYSDEHEBGT : HEBERGEMENT → PAYS	association
(DF2)	HEBGTDUPAYS : PAYS ↠ HEBERGEMENT	SITUATION
(DF3)	LOISIRSDANSPAYS : PAYS ↠ LOISIRS	association
(DF4)	PAYSDELOISIRS : LOISIRS ↠ PAYS	LOISIRSPAYS
(DF5)	HEBGT&LOISIRS:HEBERGEMENT ↠LOISIRS	association
(DF6)	LOISIRS&HEBGT:LOISIRS ↠ HEBERGEMENT	LOISIRS HEBGT

- L'utilisation de (DF5) produit directement une complétion :

(a) HEBERGEMENT $\xrightarrow{\text{HEBGT\&LOISIRS}}$ LOISIRS

- Il est encore possible d'en produire une autre en utilisant (DF1) et (DF3) et les règles de dérivation suivantes :

(R1) - transitivité des dépendances multivaluées
si $X \twoheadrightarrow Y$ et $Y \twoheadrightarrow Z$ dans $R(X,Y,Z,W)$, avec X,Y,Z disjoints deux à deux et W éventuellement vide alors
$X \twoheadrightarrow Z$ dans R.

(R2) - règle entre dépendance fonctionnelle et dépendance multivaluée
si $X \rightarrow Y$ alors $X \twoheadrightarrow Y$ (avec X et Y disjoints)

La chaîne de dérivation s'établit comme suit :

(1) HEBERGEMENT → PAYS par (DF1)

(1') HEBERGEMENT ↠ PAYS par (R2)

(2) HEBERGEMENT ↠ LOISIRS par (R1) appliquée à (1') et (DF5)

La requête "complétée" est alors :

(b) HEBERGEMENT $\xrightarrow{\text{PAYSDEHEBGT}}$ PAYS $\xrightarrow{\text{LOISIRSDANSPAYS}}$ LOISIRS

Enonçons quelques remarques avant de passer à la présentation détaillée du système de complétion.

Remarque 1 :

Le processus décrit est essentiellement syntaxique. La "conformité sémantique" du résultat de la complétion et de la requête initiale est du domaine de décision de l'utilisateur. (Le système peut produire les complétions (a) et (b) de l'exemple précédent, mais il ne peut pas décider quelle est celle qui "répond le mieux" à la requête initiale).

Remarque 2 :

Les problèmes que soulève ce processus rejoignent ceux que nous avons évoqués lors de la présentation des systèmes experts (chap. II/A) : méthodes et stratégies de résolution...

Nous détaillerons au paragraphe IV.3 de ce chapitre quelques réflexions à ce sujet.

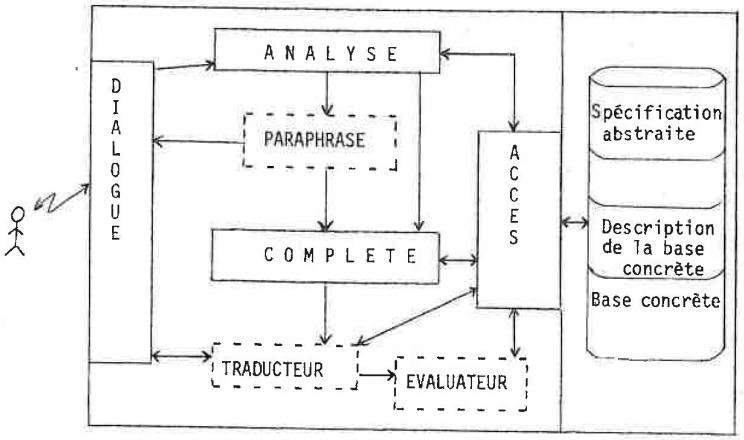
Remarque 3 :

Enfin, le fait de nommer les dépendances (et d'utiliser leurs noms) améliore, selon nous, la lisibilité des requêtes.

III - ARCHITECTURE GLOBALE DU SYSTEME

L'architecture fonctionnelle du système de compréhension de requêtes est décrite ci-dessous.

III.1.- Architecture et composants du système



NOTE : Les modules en traits discontinus ne sont pas actuellement réalisés.

- Le module de DIALOGUE assure la communication avec l'extérieur :
  - lecture des requêtes
  - demande d'informations pour :
    - corriger certaines erreurs (ex : symboles inconnus du système)
    - poursuivre la complétion
    - avoir l'avis de l'utilisateur sur des états (partiels ou finals) de la complétion.
- ANALYSE se charge de l'analyse de la requête. Il effectue
  - la reconnaissance des bribes
  - la constitution de l'arbre de la requête
  - l'élimination de certaines ambiguïtés
  - l'identification des agrégats et associations invoqués par la requête.
- COMPLETE est le module chargé d'effectuer la recherche du sous-schéma, associé à la requête, qui contient les agrégats et les associations identifiés par l'analyseur. Il utilise la méta-base et les résultats de l'ANALYSEur.
  La solution est recherchée progressivement. Chaque élément de solution est soumis à l'approbation de l'utilisateur. Celui-ci a la possibilité de :
  - l'accepter
    Dans ce cas, le système poursuit sa recherche jusqu'à obtenir une solution globale constituée par les éléments de solution acceptées.
  - la refuser
    Dans ce cas, le système peut être amené à demander des informations complémentaires concernant les raisons du refus.
    L'utilisateur agit donc sur le processus de recherche en indiquant les "voies erronnées" qu'a pu suivre COMPLETE, ou en indiquant des voies plus sûres.

Néanmoins, en cas d'absence d'indications complémentaires, le système poursuit quand même sa recherche jusqu'à aboutir à une solution agréée par l'utilisateur ou à un échec de la complétion, (si toutes les solutions qu'il a pu trouver sont refutées par l'utilisateur).

- TRADUCTEUR et EVALUATEUR sont des modules qui interviennent après COMPLETE, lorsque celui-ci s'achève avec succès.

  EVALUATEUR est le sous-système de résolution qui va rechercher dans la base concrète, les données qui répondent à la requête.

  Il comportera les deux mécanismes de recherche évoqués précédemment : extraction de données à partir des ensembles mémorisés et/ou mise en oeuvre d'un processus de déduction.

  TRADUCTEUR a pour but de traduire le résultat de COMPLETE dans le langage cible de l'EVALUATEUR. Cette approche permet de diversifier les "systèmes cibles". Ainsi, le résultat de TRADUCTEUR peut être un programme VEGA, une requête relationnelle ou une clause, selon que l'EVALUATEUR est VEGA, un SGBD relationnel ou PROLOG, respectivement.

Nous tenons à préciser que SINDBAD ne comporte pas (encore) de TRADUCTEUR général. Pour le moment, notre système réalise la traduction de la requête complétée en une clause PROLOG. Le traducteur réalisé n'est pas général, en ce sens qu'il s'appuie sur une représentation donnée des entités et des associations. (Si cette représentation change, le traducteur actuel devient caduc).

Nous donnerons, au dernier paragraphe de ce chapitre, quelques indications sur la façon dont pourrait être concrétisé un "traducteur général".

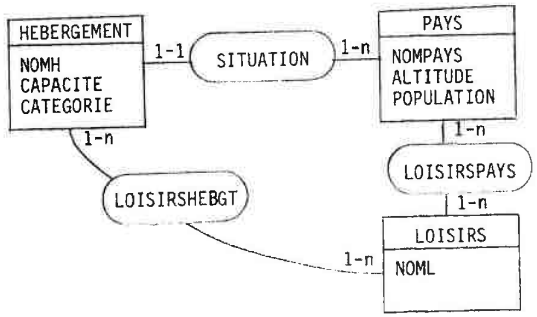
- PARAPHRASE est le module chargé de constituer, à partir des représentations internes des solutions et sous-solutions, des formes externes intelligibles par l'utilisateur du système.

- ACCES permet de rendre disponible à l'intention de l'ANALYSEur et du module de COMPLETion de requêtes les informations de la méta-base nécessaires à leur fonctionnement.

Avant d'entrer dans le détail du système, nous allons illustrer son fonctionnement à l'aide de quelques exemples.

III.2.- Exemples de fonctionnement

Les exemples que nous traiterons utiliseront le schéma ci-dessous.



On suppose qu'une méta-base a été constituée et qu'elle contient notamment

- les symboles d'agrégats, d'attributs, d'associations, d'opérations.
- la description structurelle des agrégats (liste des attributs de chacun d'eux).
- la description des associations (collection, cardinalité, symboles des fonctions la définissant).
- les profils des opérations
- ...

On suppose par ailleurs que la représentation de la base et l'implémentation de ses opérations ont été réalisées selon les principes définis au chapitre précédent.

Pour la clarté de l'exposé, supposons la représentation faite en termes de PROLOG comme suit :

```
rep(HEBERGEMENT)      :  + HEBERGEMENT(*NOMH,*CAP,*CAT)
rep(PAYS)             :  + PAYS(*NOMPAYS,*ALT,*POP)
rep(LOISIRS)          :  + LOISIRS(*NOML)
rep(SITUATION)        :  + SITUATION(*NOMH,*NOMPAYS)
rep(LOISIRSPAYS)      :  + LOISIRSPAYS(*NOML,*NOMPAYS)
rep(LOISIRSHEBGT)     :  + LOISIRSHEBGT(*NOML,*NOMH)-
                           - SITUATION(*NOMH,*NOMPAYS)
                           - LOISIRSPAYS(*NOML,*NOMPAYS).
```

1) Exemple 1 :

- Demande du client : Trouver un hébergement et faire du SKI de FOND
- Expression de la requête : SORTIR HEBERGEMENT : LOISIRS "SKI DE FOND".
- Phase 1 : L'analyse de la requête produit comme premier résultat :

```
SORTIR X1 TEL QUE
       X1 = NOMH(HEBERGEMENT)
   et X2 = NOML(LOISIRS)
   AVEC X2 = SKI DE FOND
```

Phase 2 : a) COMPLETE essaie de joindre les agrégats HEBERGEMENT et LOISIRS à l'aide des associations définies. Il produit l'élément de solution suivant :

- "HEBERGEMENT et LOISIRS sont liés par le fait que :
  (1) - HEBERGEMENT-LOISIRSHEBGT-LOISIRS"
- "Y-A-T-IL DES INCOHERENCES DANS MON INTERPRETATION (O/N) ?".

b)• Cas 1 : L'utilisateur accepte la proposition (1)

COMPLETE s'arrête. L'interprétation de la requête est alors :

```
SORTIR X1 TEL QUE
       X1 = NOMH (HEBREGEMENT)
   ET X2 = NOML(LOISIRS)
   AVEC X2 = SKI DE FOND
   ET   LOISIRSHEBGT(LOISIRS,HEBERGEMENT)
```

• Cas 2 : L'utilisateur rejette la proposition :
L'association LOISIRSHEBGT ne lui convient pas et il ne peut pas indiquer à COMPLETE l'association qui l'intéresse.

COMPLETE va essayer de trouver un autre "chemin" entre HEBERGEMENT et LOISIRS, puis le proposer :

- HEBERGEMENT et LOISIRS sont liés par le fait que :
  (1) HEBERGEMENT.SITUATION.PAYS
  (2) PAYS.LOISIRS PAYS.LOISIRS
- Y-A-T-IL DES INCOHERENCES ?

L'utilisateur est d'accord avec ces propositions.
La complétion s'arrête avec la forme de requête :

```
SORTIR X1 TEL QUE
       X1 = NOMH(HEBERGEMENT)
   et X2 = NOML(LOISIRS)
   AVEC X2 =  SKI DE FOND
   et   SITUATION(HEBERGEMENT,PAYS)
   et   LOISIRSPAYS(LOISIRS,PAYS)
```

c) Le TRADUCTEUR produit alors les requêtes-résultats correspondantes :

cas 1 : - HEBERGEMENT(*X1,*1,*2)-LOISIRS(SKIDEFOND)
-LOISIRSHEBGT(SKIDEFOND,*X1)-SORTIR(*X1)

cas 2 : - HEBERGEMENT(*X1,*1,*2)-LOISIRS(SKIDEFOND)
-SITUATION(*X1,*P)-LOISIRSPAYS(SKIDEFOND,*P)
-SORTIR(*X1)

2) Exemple 2 :

- Demande du client : Hébergement pour trois personnes à 1200 mètres d'altitude.
- Expression de la requête : SORTIR HEBERGEMENT : CAPACITE "3", ALTITUDE "1200".
- Résultat de l'analyseur :

SORTIR X1 TEL QUE
X1 = NOMH(HEBERGEMENT)
et X2 = CAPACITE(HEBERGEMENT)
et X3 = ALTITUDE(PAYS)
AVEC X2 = 3
X3 = 1200

- Résultat de la complétion

HEBERGEMENT et PAYS sont liés par le fait que
HEBERGEMENT.SITUATION.PAYS

L'utilisateur est d'accord. Le TRADUCTEUR produit alors l'expression PROLOG :

- HEBERGEMENT(*X1,3,*1)-SITUATION(*X1,*P)-PAYS(*P,1200,*2)
-SORTIR(*X1).

REMARQUE : Dans la version actuelle du système, PARAPHRASE produit des expressions de la forme <agrégat>.<association>.<agrégat> ; il pourrait produire des formes plus expressives si on lui en donne les moyens (comme, par exemple, un ensemble de règles définissant les structures des paraphrases).

ex : REGLE RELATIVE à SITUATION (HEBERGEMENT, PAYS) :

L'exprimer en (1) HEBERGEMENT×1 SOIT SITUE DANS LE PAYS P
ou (2) la SITUATION DE L'HEBERGEMENT×1 SOIT LE PAYS P

La règle associée à (1) serait de la forme :
"<agrégat 1> SOIT SITUE DANS LE <agrégat 2>",

et celle associée à (2) :
"LA SITUATION DE l'<agrégat 1> SOIT LE <agrégat2>"

En somme, PARAPHRASE disposerait de règles qui définissent des "expressions naturelles" des associations inter-agrégats. Il les utiliserait pour engendrer des formes externes plus expressives que la notation pointée produite par le système actuel.

IV - ANALYSE ET COMPLETION DE REQUETES

Le processus de compréhension se déroule en deux phases :
- une phase d'analyse de la requête
- une phase de complétion.

IV.1.- Reconnaissance de la requête

Elle a pour but de :
- effectuer l'analyse syntaxique de la requête
- lever certaines ambiguïtés
- identifier les agrégats et les associations invoqués
- produire une structure interne de représentation de la requête
- élaborer une première interprétation.

1.- L'analyse

L'analyseur de la requête a pour rôle de reconnaître les éléments qui constituent la cible et les sélecteurs. Il opère en con-

sultant le dictionnaire des symboles de la base (cf. ANNEXE V) pour déterminer la nature des symboles qu'il reconnaît (symbole d'agrégat, d'attribut, d'association, de fonction rôle ou autre). Il produit une liste des cibles et des sélecteurs présents dans la requête.

2.- Ambiguïtés et synonymies

Des "libertés d'expression" sont autorisées par le langage. L'étape suivante de l'analyseur vise à obtenir des expressions plus rigoureuses. Ces libertés concernent :

a) La possibilité de désigner un agrégat par son identificateur ou par celui de sa clé.

On ramène ce type d'expression à une expression de la forme

X_i = <identificateur de la clé> (<identificateur d'agrégat>).

On introduit également une variable X_i.

Exemple : NOMH ou HEBERGEMENT ou NOMH (HEBERGEMENT) seront exprimés sous la forme X1 = NOMH(HEBERGEMENT).

b) La possibilité de ne pas indiquer la localisation d'un attribut dans l'agrégat qu'il caractérise.

Le système identifie l'agrégat. Selon que l'attribut est clé ou pas de l'agrégat, il produit l'expression

X_i = <identif.clé>(<identif.agrégat>)
ou X_i = <identif.clé>(<identif.agrégat>)
et X_{i+1} = <identif.attribut> (X_i)

Exemple : La présence de ALTITUDE dans une requête donnera lieu aux expressions :

X2 = NOMPAYS(PAYS)
et X3 = ALTITUDE(X2)

En d'autres termes, on fait le choix de repérer un agrégat par sa clé. De ce fait, on utilise la même variable pour désigner la valeur de la fonction d'accès associée à la clé et l'agrégat lui-même.

3.- Identification des agrégats et des associations invoqués par la requête

a) Identification des agrégats

L'ensemble des agrégats, que nous noterons $\mathcal{P}$, qu'invoque une requête, est constitué par :

- les identificateurs d'agrégats mentionnés explicitement dans la requête

ex : SORTIR HEBERGEMENT ... ⇒ HEBERGEMENT ∈ $\mathcal{P}$

- les identificateurs d'agrégats introduits par les transformations effectuées par l'analyseur

ex : SORTIR NOMH : ALTITUDE "1200" ⇒ HEBERGEMENT et PAYS ∈ $\mathcal{P}$

- les identificateurs des agrégats invoqués par une association ou des fonctions, comme va le montrer le paragraphe qui suit.

b) Identification des associations

L'ensemble $\mathcal{S}$ des associations invoquées dans une requête est constitué par celles

- invoquées explicitement

ex : SORTIR HEBERGEMENT : SITUATION(HEBERGEMENT,"MARGERIDE")
⇒ SITUATION ∈ $\mathcal{S}$

- invoquées de façon indirecte

C'est le cas où un symbole de fonction rôle est utilisé dans une requête. L'association sur laquelle est défini le rôle est incluse dans $\mathcal{S}$.

Les agrégats de sa collection sont inclus dans $\mathcal{P}$.

ex : SORTIR LOISIRS : HEBGT&LOISIRS("LE NID")

⇒ $\mathcal{P}$ = {HEBERGEMENT,LOISIRS}

$\mathcal{S}$ = {LOISIRSHEBGT}

L'objectif de ce processus est de reconnaître l'ensemble $\mathcal{P}$ des agrégats (ou noeuds) et celui $\mathcal{S}$ des associations (ou segments) qui feront nécessairement partie du sous-schéma de la base que détermine la requête.

4.- Elaboration d'une première interprétation

Elle est réalisée à l'issue des différentes transformations qui ont précédé. Elle est une transcription en clair de la représentation interne de la requête (décompilation). Y figurent :

- les cibles
- les variables introduites par le système
- les littéraux éventuellement rencontrés dans la requête.

IV.2.- La complétion de la requête

C'est un processus interactif qui vise à trouver une interprétation plausible de la requête.

Concrètement, il s'agit de trouver un sous-graphe du graphe de la base, qui contienne $\mathcal{P}$ et $\mathcal{S}$, c'est-à-dire dont l'ensemble des noeuds contienne $\mathcal{P}$ et l'ensemble des arcs contienne $\mathcal{S}$.

Pour y parvenir, le sous-système de complétion s'appuie sur la définition de la base, et plus particulièrement sur la définition des associations.

IV.2.1.- Schéma global de fonctionnement

Le principe de l'algorithme de complétion est le suivant :

(1) - Examen de $\mathcal{P}$ et $\mathcal{S}$ pour déterminer si la requête est initialement complète.

(a) Si c'est le cas, proposer l'interprétation à l'utilisateur et s'arrêter si elle lui convient.

(b) Si la requête n'est pas initialement complète ou que l'utilisateur n'est pas d'accord avec l'interprétation du (a), poursuivre la complétion en (2).

(2) - Choisir un point O dans $\mathcal{P}$.

(2')- Trouver tous les agrégats auxquels il est associé.
Trouver, en même temps, pour chaque p associé à O, la liste A_p des associations qui lient p et O.
Soit ADJ(O) = $\{<p,A_p>\}$ le résultat de cette étape.

(3) - Soit ADJ'(O) = ADJ(O) ∩ $\mathcal{P}$

si ADJ'(O) n'est pas vide (il existe dans les points p, adjacents à O, un point figurant dans la requête).

alors - proposer à l'utilisateur la suite d'associations qui mène de O à tout point p' de ADJ'(O).

- débuter le dialogue

- mémoriser les réponses de l'utilisateur pour
  - ne plus reconstituer les propositions acceptées
  - ou ne plus réutiliser, dans la suite du processus, les éléments des propositions qui ont été refutés.

(4) - Poursuivre le processus jusqu'à ce que $\mathcal{P}$ et $\mathcal{S}$ soient entièrement parcourus et que les suites de propositions émises

soient acceptées par l'utilisateur, sinon s'arrêter en échec.

exemple : voir page suivante.

En fait, l'algorithme ne progresse pas "au fil de l'eau" comme pourrait le laisser supposer sa description précédente. En effet, certaines règles sont utilisées à chaque étape de la complétion pour :

- éliminer des agrégats et des associations "indésirables"
- ordonner les agrégats et les associations qui subsistent après l'application des règles d'élimination.

IV.2.2.- Règles de conduite de la complétion

Les règles utilisées pour guider et rendre "moins aveugle" le processus de complétion, sont de deux types :

- les règles dites d'élimination
- les règles dites d'ordre.

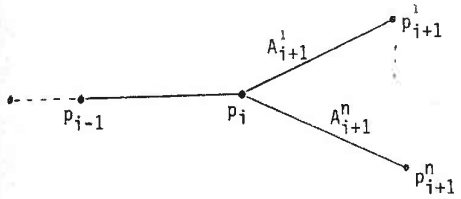
IV.2.2.1.- Les règles d'élimination

Elles permettent de rejeter des associations et des agrégats. Ces règles sont activées chaque fois que le processus est amené à rechercher les agrégats associés à un agrégat donné. Celui-ci, que l'on appellera agrégat à propager, peut être source (ie choisi dans $\mathcal{P}$) ou intermédiaire (ie est sur le chemin en cours d'élaboration).

Considérons p_i le point à propager, p_{i-1} un de ses antécédents et l'ensemble $ADJ(p_i) = \{<p^k_{i+1}, A^k_{i+1}>, k \in [1,n]\}$ des n agrégats associés à p_i avec leurs ensembles respectifs A^k_{i+1} d'associations.


```
ENTREZ VOTRE REQUETE ou "FIN" :
TIR PAYS:HEBERGE "LE NID".

   I    ANALYSE  DE  LA  REQUETE    I

% premiere interpretation de votre demande est :

SORTIR  X01     TEL QUE :
 - 01 - X01 = NOMPAYS(X02)
 - 02 - X02 = PAYS
 - 03 - X03 = NOMHEB(X04)
 - 04 - X04 = HEBERGE

 AVEC :

  NOMHEB = LE NID


   I    COMPLETION DE LA REQUETE    I

PAYS ET HEBERGE sont lies par le fait que :

  - 1 - PAYS . SITUATION . HEBERGE

Y-A-T-IL DES INCOHERENCES DANS MON INTERPRETATION (O/N) ? : O

VOULEZ-VOUS M'INDIQUER SUR QUELLE(S) LIGNE(S) ? : 1
AUTRE NUMERO DE LIGNE ou 99 (pour FIN) : 99
SAVEZ-VOUS QUELLE RELATION LIE PAYS ET HEBERGE (O/N) ? : N

JE VAIS ESSAYER AUTRE CHOSE, SI POSSIBLE.

IL N'Y A PAS D'AUTRE RELATION ENTRE PAYS
  ET  HEBERGE .

PAYS ET HEBERGE sont lies par le fait que :

  - 1 - PAYS . LOISIRSPAYS . LOISIRS
  - 2 - LOISIRS . LOISIRSHEBGT . HEBERGE

Y-A-T-IL DES INCOHERENCES DANS MON INTERPRETATION (O/N) ? : N

NOUS EN SOMMES A LA SUITE DE PROPOSITIONS :

   - 1 -  PAYS . LOISIRSPAYS . LOISIRS
   - 2 -  LOISIRS . LOISIRSHEBGT . HEBERGE
```

```
- Mon interpretation finale de votre demande est :
  ------------------------------------------------

    SORTIR  X01     TEL QUE :
     - 01 - X01 = NOMPAYS(X02)
     - 02 - X02 = PAYS
     - 03 - X03 = NOMHEB(X04)
     - 04 - X04 = HEBERGE

    AVEC :

      NOMHEB = LE NID

      ET :

          1)  LOISIRSPAYS(PAYS,LOISIRS)
          2)  LOISIRSHEBGT(LOISIRS,HEBERGE)


Q- Voulez-vous TRADUIRE la requete dans un langage particulier ? (O/N) :

Q- Dans le langage de quel SYSTEME CIBLE ? : PROLOG

        ---------------------------------------
        I    TRADUCTION DE LA REQUETE EN PROLOG    I
        ---------------------------------------


*-------------------------*
I REMARQUE IMPORTANTE: I-----------------------------------------*
*-------------------------*CE TRADUCTEUR de requete suppose que ù
ù les choix de representation suivants ont ete faits :         ù
ù                                                              ù
ù-  rep(AGREGAT(C1,...,Cn)) = +AGREAGT(*C1,...,*Cn)            ù
ù-  rep(ASSOCIATION(AGR1,AGR2)) = +ASSOCIATION(*CLE1,*CLE2)    ù
ù     ou CLE1 est la cle de l'agregat1 (AGR1) et               ù
ù        CLE2 celle de l'agregat2 (AGR2)                       ù
*--------------------------------------------------------------*


+RQTE(*X01)-PAYS(*X01,*01,*02)-HEBERGE(LE NID,*03,*04)
           -LOISIRSPAYS(*X01,*X10)-LOISIRSHEBGT(*X10,LE NID)
           -SOR(*X01)
```

Les règles d'élimination actuellement utilisées sont au nombre de trois :

(R1) REGLE 1 :

Elle consiste à éliminer dans tout A_{i+1}^k, toute association déjà proposée et refusée par l'utilisateur.

Pour cela, le système doit gérer un ensemble REFUS qui contient les associations déjà refusées.

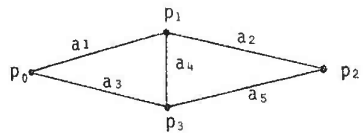
Cette règle peut s'exprimer plus formellement ainsi :

$\forall a$, a : ASSBIN ; $\forall k$, $k \in [1,n]$

DANS $(A_{i+1}^k, a) \wedge$ DANS (REFUS,a) $\rightarrow$ RETIRER(A_{i+1}^k, a)

[DANS et RETIRER permettent respectivement de tester la présence d'un élément dans un ensemble et d'enlever un élément d'un ensemble].

Exemple de cas de figure où (R1) est applicable



Supposons que pour relier p_0 et p_2, on aboutisse aux propositions :

(1) $p_0 \cdot a_1 \cdot p_1$

(2) $p_1 \cdot a_2 \cdot p_2$

et que l'utilisateur rejette (2).

Le processus de complétion ne réutilisera plus l'association a_2.

(R2) REGLE 2 :

Elle concerne les conditions d'application des associations.

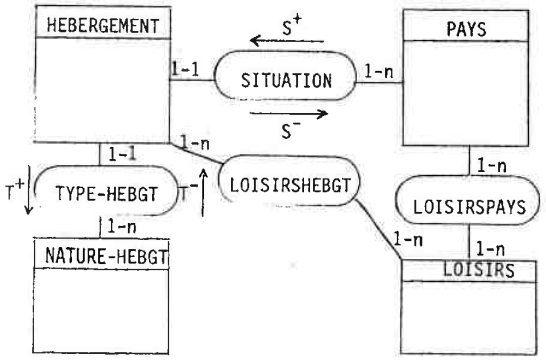
Si une requête invoque deux agrégats A et C entre lesquels il n'existe pas d'association, mais qu'il existe un agrégat B qui soit lié à A par une association R1 et à C par une association R2, on considère alors que l'on veut relier A à C "selon la "sémantique" d'une relation R englobant la "sémantique" de R1 et R2".

La règle 2 permet de préciser les conditions d'acceptation de la conjonction de R1 et R2 dans l'interprétation d'une requête.

Voyons l'utilisation de cette règle sur un exemple.

Considérons le schéma du § III.2 de ce chapitre avec

- les associations SITUATION, LOISIRSPAYS, LOISIRSHEBGT
- et une nouvelle association TYPE-HEBGT qui lie un hébergement à sa NATURE (camping, hotel,...) et inversement.



CAS 1 : On cherche à relier NATURE-HEBGT et PAYS par les associations TYPE-HEBGT et SITUATION

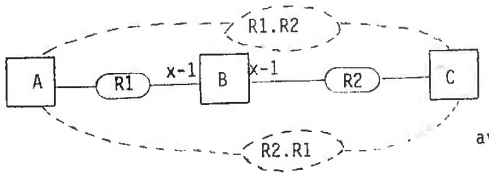
La relation R, résultant de la composition de TYPE-HEBGT et de SITUATION permet d'associer NATURE-HEBGT et PAYS. Elle détermine les TYPES D'HEBERGEMENT qui se trouvent dans un PAYS d'accueil ($S^+.T^+$). A l'inverse, ($T^-.S^-$), on obtient les PAYS avec des HEBERGEMENTs de TYPE(s) donné(s).

Donc, dans un sens comme dans l'autre (de NATURE-HEBGT vers PAYS ou inversement), l'association (virtuelle) R définie par la composition de SITUATION et de TYPE-HEBGT peut constituer une interprétation plausible d'une requête où seraient invoqués PAYS et NATURE-HEBGT.

De façon générale, de telles compositions d'associations sont engendrées par le processus de complétion et soumises à l'approbation (ou au refus) de l'utilisateur en temps utile.

Donc, si, entre deux agrégats A et C, il existe un agrégat B lié à A par l'association R1 et lié à C par R2 et que les cardinalités de B dans R1 et dans R2 sont 0-1 ou 1-1

alors R1 et R2 sont "composables" pour associer A et C, indépendamment des cardinalités de A et de C dans R1 et R2 respectivement.



avec $x \in \{0,1\}$

CAS 2 : Considérons maintenant la composition de LOISIRS-HEBGT et SITUATION

Supposons que LOISIRS-PAYS n'existe pas, et que le processus de propagation ait permis d'aboutir à LOISIRS à partir de PAYS (ou inversement).

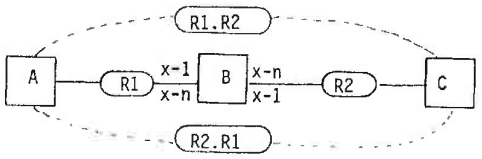
La question que l'on est amené à se poser est de savoir si une telle composition est possible. Dans l'affirmative, que peut-elle signifier ?

Dans le "sens" PAYS vers LOISIRS, l'utilisation de SITUATION permet de déterminer les hébergements d'un pays puis, les loisirs à proximité de ceux-ci (à l'aide de l'association LOISIRS HEBGT).

Dans le "sens" LOISIRS vers PAYS, l'utilisation de LOISIRS HEBGT détermine les hébergements associés à un LOISIRS et SITUATION détermine le PAYS de ces hébergements.

Ainsi, la conjonction de ces associations semble exprimer la sémantique de l'association LOISIRS-PAYS. Elle pourra être engendrée par le processus de complétion et soumise à l'utilisateur.

Plus généralement, si R1 et R2 associent un agrégat B, respectivement à A et C, et que B a une cardinalité de (x,1) dans R1 et de (x,n) dans R2 ($x \in [0,1]$) alors R1 et R2 sont composables pour associer A et C (indépendamment des cardinalités de ces derniers).



CAS 3 : Considérons enfin la composition LOISIRS PAYS et LOISIRS HEBGT

Supposons que l'association SITUATION ne fasse pas partie du schéma.

LOISIRS PAYS permet d'associer à tout PAYS ses LOISIRS et à tout LOISIRS les PAYS où il est possible de le pratiquer.

LOISIRSHEBGT associe de la même façon à tout HEBERGEMENT les LOISIRS à proximité et à tout LOISIRS les HEBERGEMENTs près desquels il peut être exercé.

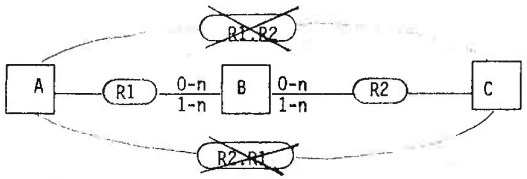
Etant donné un PAYS, l'utilisation de la composition LOISIRS PAYS.LOISIRSHEBGT détermine l'ensemble des LOISIRS du PAYS et pour chacun de ces loisirs, l'ensemble des hébergements qui lui sont associés. Ces HEBERGEMENTs sont déterminés indépendamment du PAYS concerné.

Donc une requête qui ferait appel à cette composition pourrait induire l'utilisateur en erreur car il est possible qu'il interprète la liste d'hébergements produite comme étant celle des hébergements situés dans le pays en question.

Le même scénario, appliqué de HEBERGEMENT vers PAYS pour un hébergement 'H' donné, produira l'ensemble des LOISIRS $L_1,\ldots,L_n$ à proximité de 'H' et l'ensemble des PAYS $\{\{P_{L_1},\ldots,P_{L_1}^{n_1}\},\ldots,\{P_{L_n},\ldots,P_{L_n}^{nm}\}\}$ où chacun des loisirs est praticable. Comment l'utilisateur va-t-il interpréter alors, l'association entre 'H' et les $P_{L_i}^{i}$?

Considérant donc, que la composition de telles associations est source d'erreurs d'interprétation, nous nous interdisons d'en produire lors du déroulement du processus de complétion.

Donc, si R1 et R2 associent un agrégat B à A et C respectivement
et que les cardinalités de B dans R1 et R2 sont (x,n) avec $x \in \{0,1\}$
alors R1 et R2 ne sont pas composables pour associer A et C



Le tableau ci-dessous résume ces trois cas et nous permet d'énoncer la règle 2 d'élimination.

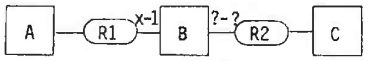
CARDINALITES DE B dans R1	dans R2	COMPOSITION R1,R2
x,1	x,1	OUI
x,1	x,n	OUI
x,n	x,1	OUI
x,n	x,n	NON

avec $x \in \{0,1\}$ et $n > 1$

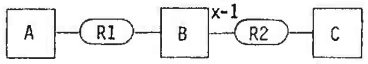
Règle 2 d'élimination

Deux associations R1 (A,B,(min1,max1),(min2,max2))
et R2 (B,C,(min'1,max'1),(min'2,max'2))
sont composables lors de la complétion d'une requête
si $\forall$ min1,max1,min'2,max'2 on a
$(min2 \in \{0,1\} \wedge max2 = 1) \vee (min'1 \in \{0,1\} \wedge max'1 = 1)$

Donc les cas de figures favorables sont les suivants :



et

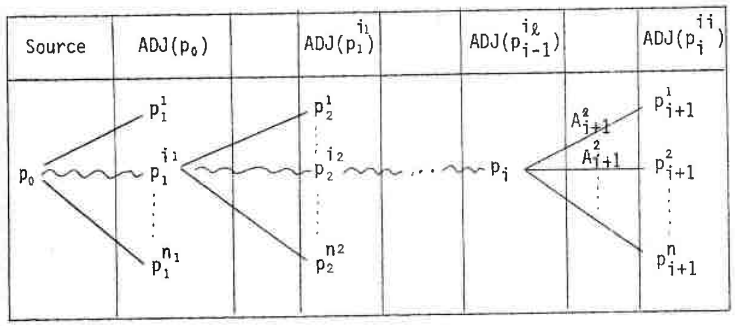


Le processus de complétion applique cette règle à chaque étape de propagation : sur le "tronçon" A-B-C puis, éventuellement sur le tronçon B-C-D...

(R3) REGLE 3 : Elle est une conséquence des règles précédentes

Si à la suite de l'application des règles 1 et 2, l'ensemble A^k_{i+1} des associations entre un agrégat p_i et un agrégat p_{i+1}, ne comporte plus d'association alors le point p_{i+1} est retiré de l'ensemble $ADJ(p_i)$ des agrégats associés à p_i.

Supposons, par exemple, que l'on aboutisse à un état j du déroulement du processus de complétion que schématise le tableau suivant :



p_ℓ —— $p_{\ell+1}$ symbolise l'ensemble des associations entre p_ℓ et $p_{\ell+1}$

p_ℓ ~~~ $p_{\ell+1}$ symbolise l'association choisie par le système de complétion (ie $p_0 \sim\sim p_1^{i_1} \sim\sim p_2^{i_2} \sim\sim \ldots \sim\sim p_i$ est susceptible de faire partie du résultat de la complétion).

Si l'application des règles 1 et 2 à A^1_{i+1} élimine toutes les associations de A^1_{i+1}, on éliminera p^1_{i+1} des successeurs possibles de p_i.

Ce mécanisme est appliqué à tout agrégat p_{i+1} adjacent à p_i.

IV.2.2.2.- L'ordonnancement des agrégats et des associations

a) L'ordre des agrégats

L'ordre établi sur les agrégats est défini comme suit :

$\forall i,k_1,k_2,\ p_{i+1}^{k_1} \in ADJ(p_i)$ et $p_{i+1}^{k_2} \in ADJ(p_i)$

- $p_{i+1}^{k_1} \in \mathcal{P} \wedge p_{i+1}^{k_2} \in \mathcal{P} \rightarrow p_{i+1}^{k_1} = p_{i+1}^{k_2}$

- $p_{i+1}^{k_1} \in \mathcal{P} \wedge p_{i+1}^{k_2} \notin \mathcal{P} \rightarrow p_{i+1}^{k_1} > p_{i+1}^{k_2}$

- $p_{i+1}^{k_1} \notin \mathcal{P} \wedge p_{i+1}^{k_2} \notin \mathcal{P} \rightarrow p_{i+1}^{k_1} = p_{i+1}^{k_2}$

Cet ordre permet de "mettre en tête", afin de les traiter en premier, les agrégats p_{i+1}, adjacents à p_i, qui sont dans l'ensemble $\mathcal{P}$ des agrégats invoqués par la requête.

b) L'ordre des associations

Tout ensemble A_{i+1}^k d'associations liant les agrégats p_i et p_{i+1} est ordonné selon un critère similaire au précédent. Les associations de A_{i+1}^k présents dans $\mathcal{S}$ sont "en tête".

IV.2.3.- Structure finale du sous-système de complétion

a) L'algorithme

Les étapes 2' et 4 de l'algorithme général du § IV.2.1 ont été modifiées pour prendre en compte les règles d'élimination et les critères d'ordre.

En 2', on appliquera ces règles et en 4, la poursuite du processus se fera en choisissant comme nouveau point à propager :

- soit un point de $ADJ'(0)$ ($ADJ'(0) = ADJ(0) \cap \mathcal{P}$), si $ADJ'(0)$ n'est pas vide,
- soit le premier point dans $ADJ(0)$.

L'algorithme de complétion qui opère selon cette stratégie, est alors le suivant :

(1) - Etant donné un point p, rechercher les points qui lui sont adjacents.

(2) - Eliminer ceux qui doivent l'être (d'après les règles d'élimination).

(3) - Ordonner ceux qui restent (selon les critères d'ordre définis).

(4) - Si aucun d'entre eux ne figure dans $\mathcal{P}$, en choisir un quelconque et relancer le processus à partir de (1).

(5) - Si, par contre, il en existe certains ($ADJ'(p) \neq \{\}$) qui sont dans $\mathcal{P}$, émettre des propositions à l'utilisateur.

[sous la forme : (1) $p_0 \cdot R1 \cdot p_1$

(2) $p_1 \cdot R2 \cdot p_2$

$\vdots$

(i+1) $p_i \cdot R_{i+1} \cdot p_{i+1}$

$\vdots$

(n) $p \cdot R_p \cdot p'$ où p_o et p' appartiennent à $\mathcal{P}$]

5.1 - L'utilisateur pourra agréer tout ou partie de ces propositions.

5.2 - Pour les propositions refusées, il pourra les remplacer par d'autres (par exemple, remplacer $p_i \cdot R_{i+1} \cdot p_{i+1}$ par $p_i \cdot R'_{i+1} \cdot p_{i+1}$) ou ne pas proposer d'éléments de remplacement.

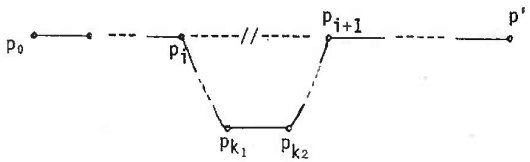
Dans ce dernier cas, le système va émettre de nouvelles propositions de remplacement, par l'exploration des ensembles des associations qui lient les agrégats figurant sur les lignes de propositions refusées.

[si $R^1_{i+1}, R^2_{i+1}, \ldots, R^\ell_{i+1}$ associent p_i et p_{i+1}, le système les proposera systématiquement à l'utilisateur].

Si aucune des propositions ne convient, il effectuera d'autres tentatives de propagation débutant en p_{i-1}.

Le fait qu'aucune association entre p_i et p_{i+1} ne satisfasse l'utilisateur signifie que la voie menant de p_0 à p' ne doit pas comprendre p_i et p_{i+1} ou, si p_i et p_{i+1} doivent figurer sur cette voie, alors celle-ci doit utiliser une composition d'associations qui mènerait de p_i à p_{i+1} et non une association définie entre eux deux.

[Ce cas de figure est symbolisé ci-dessous :
L'impasse, créée entre p_i et p_{i+1}, du fait du rejet de toutes les associations qui les lient, amènera le système à rechercher d'autres voies qui pourraient contenir p_i et p_{i+1} mais via $p_{k_1}, p_{k_2} \ldots$].



(6) - Dans le cas où une liaison de p_0 à p' a été trouvée, la propagation se poursuit à partir de p' jusqu'à aboutir à la complétion de la requête ou à une impasse. (Plus de successeur admissible en un point p_z de la propagation). Dans ce cas, on choisit dans $\mathcal{P}$, un nouvel agrégat source et on relance le processus en (1).

b) Les modules du sous-système de complétion

Le sous-système de complétion, dans sa version actuelle, comporte les modules suivants :

- CHERCHELIEN : - choisit un agrégat dans $\mathcal{P}$
  - contrôle la fin de la complétion

- PROPAGE : assure la propagation de l'agrégat p_0 choisi jusqu'à trouver, parmi ses successeurs, un agrégat p_i appartenant à $\mathcal{P}$. Dans ce cas PROPAGE fait appel à FAIRE-N-CHEMINS qui explicite la succession d'associations de p_0 à p_i avant d'entamer le dialogue avec l'utilisateur.

- ETEND : appelé par PROPAGE, il a pour rôle de calculer les agrégats p^k_j adjacents à p_{j-1} et les ensembles d'associations liant chaque p^k_j à p_{j-1}.

- ELIMINE : met en oeuvre les règles d'élimination.

- ORDONNE : établit les ordres sur les agrégats et les associations.

- DISCUTE : dirige le dialogue qui s'instaure avec l'utilisateur, suite aux propositions émises par FAIRE-N-CHEMINS. Il a également pour rôle
  - de vérifier que les informations fournies par l'utilisateur, pour modifier les propositions, sont correctes.
    Par exemple, si une proposition A.R1.B devait être, modifiée, sur décision de l'utilisateur, en A.R2.B, DISCUTE vérifiera que R2 est effectivement une association de la base définie entre A et B.
  - de constituer la liste des REFUS qui sera utilisée lors de l'application des règles d'élimination.

IV.2.4.- Conclusion

Le processus est complet au sens où, si une solution existe, elle

est exhibée. En effet, la recherche est effectuée de façon exhaustive puisque pour tout point de $\mathcal{P}$, on examine toutes les propagations possibles. Si celles-ci sont insuffisantes ou mènent à un résultat inapproprié, on essaie avec un autre point, etc...

L'utilité d'un tel système est double :

- en phase de conception, sa mise en oeuvre devrait permettre de déceler certaines insuffisances du schéma de la base. En effet, l'arrêt en échec de l'algorithme peut être dû à l'expression de la requête (les éléments qu'elle comporte sont insuffisants pour que le système puisse en établir une interprétation, et de plus, l'utilisateur n'a pas été capable d'apporter une aide suffisante au système).

  Cela pourrait également signifier qu'il n'existe pas, au sein du schéma, d'association ou de composition d'associations appropriée pour satisfaire la requête. Dans ce cas, on modifierait alors le schéma par l'adjonction de/des associations adéquates.

- en phase d'utilisation d'une base : Un outil simple d'expression de requêtes doublé d'un mécanisme de compréhension nous semble être d'un grand apport pour l'utilisation d'une base de données par des non-informaticiens. Pour peu que celui-ci sache désigner la/les informations à extraire (les cibles), le langage de requête l'autorise à juxtaposer les critères de sélection de ces informations (liste de sélecteurs) en vrac. Le sous-système de complétion intervient alors pour aider l'utilisateur à élaborer une requête complète. [Cette dernière devra être transcrite sous une forme intelligible par le système qui gère la base afin qu'il puisse l'évaluer].

  Donc de tels mécanismes semblent être une voie prometteuse pour élaborer des systèmes capables de communiquer avec des utilisateurs non spécialisés.

IV.3.- Réflexions sur les limites du système de complétion et ses possibilités d'extension

Les limites du sous-système de complétion, nous semblent se situer à deux niveaux.

IV.3.1.- Au niveau de l'expression des requêtes

Le langage implémenté permet d'exprimer des requêtes du type : "Sortir toute cible telle que la conjonction de sélecteurs soit vraie".

On considère ainsi que les cibles sont universellement quantifiées et que la liste de sélecteurs ne comporte pas de disjonction. Or, il est évident que l'expression de requêtes sur un domaine donné ne peut pas être limité à des structures aussi pauvres.

Nous pensons néanmoins que ces considérations ne doivent pas remettre fondamentalement en cause le principe de la complétion, comme le montrent les quelques réflexions qui suivent.

a) Le problème des quantificateurs

A notre avis, le problème des quantificateurs peut être résolu au niveau du sous-système de traduction des requêtes complétées. Une solution possible serait que le TRADUCTEUR entame un dialogue avec l'utilisateur afin que celui-ci puisse préciser ses quantificateurs. Ce dialogue serait alors une aide apportée au TRADUCTEUR.

Voyons cela sur un scénario possible.

Considérons la requête :

SORTIR HEBERGEMENT,PAYS : CAPACITE"3"

et son expression (externe) après complétion :

```
SORTIR X1,X3 TEL QUE
       X1 = NOMH(HEBERGEMENT)
       X2 = CAPACITE(HEBERGEMENT)
       X3 = NOMPAYS(PAYS)
avec   X2 = 3
ET     HEBERGEMENT.SITUATION.PAYS
```

(1) TRADUCTEUR : Voulez-vous tous les hébergements de capacité égale à 3

(2) UTILISATEUR : (L'usage de tout quantificateur (universel, cardinal, ordinal) est supposé possible).

TOUS ou le $N^{ième}$ ou ...

(3) TRADUCTEUR : Dans Tous les pays ?

(4) UTILISATEUR : TOUS

Ce dialogue permettrait d'engendrer une requête de la forme :

```
POUR TOUT HEBERGEMENTX1 (DANS l'ensemble SHEB d'hébergements)
    TEL QUE CAPACITE(X1) = 3
    POUR TOUT PAYS X2 (dans l'ensemble SPAYS de PAYS)
        TEL QUE SITUATION(X2,X1)
        SORTIR NOMH(X1)
               NOMPAYS(X2)
    FINPOUR
FINPOUR
```

Le TRADUCTEUR entame le dialogue en s'aidant de la requête complétée et de la méta-base. De cette façon,

(1) résulterait de l'examen de la requête

(3) résulterait de l'examen de l'association entre HEBERGEMENT et LOISIRS et de la réponse fournie en (2).

Cela suppose que le TRADUCTEUR dispose de règles de la forme :

(1) Si sur un agrégat porte une restriction de son attribut clé (ie la requête mentionne une valeur de la clé)
alors il ne peut s'agir que d'une occurrence d'agrégat.

(2) Si sur un agrégat portent des restrictions sur un ou plusieurs

de ses attributs autres que la clé,
alors l'un quelconque des quantificateurs est utilisable.

(3) Si une association R entre deux agrégats A1 et A2 doit être utilisée
et qu'il n'existe aucune restriction sur A1
et qu'il n'existe aucune restriction sur A2
et que A1 est cible
et que A2 est sélecteur
et que la cardinalité de A1 dans R est (1-n)
alors l'un quelconque des quantificateurs est utilisable de A1 vers A2.

(4) ...
⋮

Le système fonctionnerait alors tel un système expert pour la construction semi-automatique de programmes.

b) Le problème de disjonction de sélecteurs

L'introduction du connecteur logique OU permettrait d'enrichir les possibilités d'expression du langage sans, a priori, remettre fondamentalement en cause le principe du sous-système de complétion.

Une solution qui oeuvrerait dans ce sens pourrait consister en la détermination de "structures alternatives" lors de la reconnaissance initiale de la requête puis à soumettre chacune d'elles au sous-système de complétion. Dans certains cas d'utilisation des disjonctions cela n'est même pas nécessaire.

Exemple

1) Une requête comportant un sélecteur de la forme

ALTITUDE "1200" OU "1400" ne change rien au

résultat de la reconnaissance ni à celui de la complétion

puisque ce sélecteur ne sera utilisé que lors de la phase d'évaluation de la requête.

2) Par contre, une requête de la forme

SORTIR HEBERGEMENT:PAYS "MARGERIDE" OU LOISIRS"SKI DE FOND" aura une incidence sur la complétion. Si on l'interprète comme :

"SORTIR LES HEBERGEMENTS SITUES EN MARGERIDE

et s'il n'en existe pas SORTIR ceux près desquels on peut faire du SKI DE FOND", on voit bien que le processus de complétion devrait être capable d'engendrer

(1) HEBERGEMENT.SITUATION.PAYS

OU (2) HEBERGEMENT.LOISHEB.LOISIRS

Pour cela, une solution consisterait à constituer des ensembles $\mathcal{P}_1$, $\mathcal{P}_2$ et $\mathcal{S}_1$, $\mathcal{S}_2$ contenant respectivement {HEBERGEMENT,PAYS} ($\mathcal{S}_1$ vide) et {HEBERGEMENT,LOISIRS} ($\mathcal{S}_2$ vide) puis à exécuter la complétion successivement sur $(\mathcal{P}_1, \mathcal{S}_1)$ et $(\mathcal{P}_2, \mathcal{S}_2)$.

Donc, la reconnaissance de la requête devra se charger de transformer une expression du type $A \wedge (B \vee C)$ où A,B,C désignent des agrégats ou des associations en $(A \wedge B) \vee (A \wedge C)$ pour pouvoir constituer les $\mathcal{P}_i$, $\mathcal{S}_i$ qui seraient soumis à la complétion.

[Par exemple, si A est une association entre E et F, et que B et C sont des agrégats, de la requête, la complétion concernera :

$\mathcal{P}_1 = \{E,F,B\}$; $\mathcal{S}_1 = \{A\}$

et $\mathcal{P}_2 = \{E,F,C\}$; $\mathcal{S}_2 = \{A\}$].

IV.3.2.- Au niveau de son intelligence

Le sous-système de complétion est limité du point de vue intelligence. En effet, le processus est essentiellement syntaxique. Même

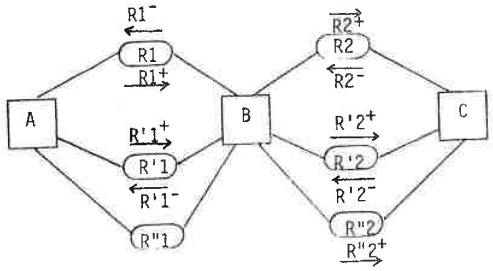
s'il parvient à retrouver des "non-dits" dans la requête qui lui est soumise, la décision finale est du ressort de l'utilisateur, dans la mesure où c'est ce dernier qui doit interpréter le résultat produit par la complétion.

Ceci tient au fait que nous n'avons pas su trouver de moyens pour indiquer au système des informations relatives à la sémantique des associations. Ces informations lui permettraient, dans une situation donnée, de pouvoir choisir les associations qui conviendraient sans avoir (ou en réduisant le) recours à l'utilisateur.

En effet, dans la version actuelle du système, si entre deux agrégats A et B existent n associations R1,...,Rn, hormis le critère trivial qui consiste à choisir celle éventuellement invoquée dans la requête, nous ne disposons pas de moyens pour choisir "celle qui convient le mieux". Ceci est vrai, tant pour une association directe que pour une succession d'associations qui mèneraient de A à B.

Dans une approche par les types abstraits, si on considère le type schéma d'une base de données, le problème serait de disposer d'un ensemble d'axiomes qui expriment les compositions d'associations (OK-AXIOMES) et celles qui ne le sont pas (AXIOMES-ERREUR).

Exemple : Sur le schéma



Le fait que R1,R2 et R'1,R'2 soient composables mais pas R"1 et R"2 serait exprimé par des axiomes de la forme :

(1) $R2^{+}(R1^{+}(A),C) = R1(A,B) \wedge R2(B,C)$

(2) $R1^{-}(R2^{-}(C),A) = R2(B,C) \wedge R1(A,B)$

(3) $R'2^{+}(R'1^{+}(A),C) = R'1(A,B) \wedge R'2(B,C)$

(4) R"2.R"1 = indéfini

(5) R"1.R"2 = indéfini

L'usage de tels axiomes lors de la complétion permettrait de réduire l'effort de recherche du système en l'empêchant d'engendrer et de proposer des compositions d'associations qu'interdisent ces axiomes.

Mais l'on sent bien que la difficulté résidera alors dans la définition exhaustive de ces axiomes. Dans notre approche, nous avons choisi de n'exprimer, en leur donnant un nom, que les compositions que l'on estimait (très subjectivement) être les plus fréquemment utilisables, les autres étant construites dynamiquement par le processus de complétion.

Mais malgré cela, ces axiomes ne permettent toujours pas d'indiquer quelle association choisir parmi un ensemble d'associations possibles. Face à ce problème, des règles d'un niveau supérieur semblent nécessaires. Elles auraient pour objectif de définir les conditions d'utilisation des règles précédentes et guideraient le système dans son choix.

Ces (méta-)règles pourraient, par exemple, être définies par contexte d'application : on établit une classification des requêtes qui peuvent porter sur la base et on définit des ordres entre les associations selon le contexte dans lequel on se trouve. L'utilisateur devra alors, outre sa requête, indiquer la classe à laquelle il estime qu'elle appartient ; le système saurait alors déterminer l'association la plus appropriée selon l'ordre établi par les méta-règles définies pour la classe en question.

Exemple : Dans le cadre d'une application sur la scolarité

- une classe de requêtes pourrait être constituée pour toutes les questions relatives à la gestion quotidienne des enseignements :
  - qui enseigne telle matière
  - en quelle(s) salle(s) se déroule tel enseignement
  - ...
- une autre classe serait constituée par des requêtes ayant trait à des problèmes d'organisation des enseignements (répartition des cours, affectation des salles...).

Dans le premier cas, une association DISPENSE qui relierait tout enseignant à la/aux matière(s) qu'il enseigne serait "supérieure" à une association PEUT-DISPENSER qui relierait tout enseignant aux matières dont il pourrait assurer l'enseignement. DISPENSE dans le contexte "gestion quotidienne" serait explorée alors avant PEUT-DISPENSER tandis que dans la contexte "organisation des enseignements" ce serait l'ordre inverse.

Mais comment déterminer ces classes ? Sont-elles forcément disjointes et si ce n'est pas le cas, comment réagir ?...

[En termes de dérivation relationnelle ce problème concerne la définition de moyens permettant de choisir, à chaque étape, la règle de dérivation la plus appropriée].

IV.4.- Conclusion

Notre objectif en réalisant un système de complétion de requêtes était d'examiner une voie qui permettrait de rapprocher l'informatique (ou plutôt son utilisation) des non-informaticiens. Cette voie, malgré ses insuffisances nous semble prometteuse.

Nous avons essayé, dans les paragraphes, qui ont précédé,

outre l'exposé fait des solutions retenues dans la version actuelle du système, de présenter quelques extensions envisageables qui, pour certaines, amélioreraient l'intelligence et les performances du système, pour d'autres contribueraient à lui ôter son caractère expérimental.

Dans le dernier paragraphe de ce chapitre, nous allons poursuivre ces réflexions par la présentation de quelques idées dont la concrétisation devrait, selon nous, compléter le système en le dotant d'un TRADUCTEUR .

V - D'UNE REQUETE APRES COMPLETION A SON EVALUATION : UN APERCU

Le but de ce paragraphe est de montrer les différentes manières d'aborder l'étape de traitement d'une requête après complétion.

Jusqu'à présent nous avons toujours fait abstraction de la représentation des bases. Depuis l'expression de la requête initiale jusqu'à sa complétion nous avons toujours utilisé la spécification.

Or, l'évaluation d'une requête doit se faire sur la structure de représentation des données de la base. Le problème est alors, de transformer la requête exprimée sur la structure abstraite (ou expression abstraite de la requête) en une requête exprimée sur la structure de représentation (ou expression concrète).

Une façon d'aborder ce problème consiste à expliciter la fonction de représentation (qui a permis d'associer une représentation de la base à sa spécification abstraite) mais aussi de fournir des moyens (automatisés ou semi-automatisés) pour exploiter cette explicitation.

V.1.- Un exemple

Nous allons maintenant décomposer tout le processus sur des

exemples. Les langages cibles sont PROLOG (a) et un langage relationnel (b).

Considérons la requête suivante :

```
SORTIR HEBERGEMENT,CATEGORIE:PAYS"MARGERIDE".
```

Après complétion nous obtenons :

```
SORTIR X1,X2
    TEL QUE X1 = NOMH(HEBERGEMENT)
            X2 = CATEGORIE(HEBERGEMENT)
            X3 = NOMPAYS(PAYS)
    AVEC    X3 = MARGERIDE
     ET     HEBERGEMENT.SITUATION.PAYS
```

a) Application à PROLOG

- Cas 1 : Supposons que la représentation suivante (REP1) ait été adoptée.

```
rep(HEBERGEMENT)::+HEBERGEMENT(*NOMH,*CATEGORIE,*CAPACITE)
rep(PAYS)       ::+PAYS(*NOMPAYS,*ALTITUDE)*POPULATION)
rep(SITUATION)  ::+SITUATION(*NOMPAYS,*NOMH)
```

L'expression de la requête en PROLOG aura la forme :

```
- HEBREGEMENT(*X1,*X2,*1)-SITUATION(MARGERIDE,*X1)-SORTIR(*X1,*X2)
```

- Cas 2 : Supposons maintenant que l'on ait choisi une représentation (REP2) qui soit :

```
rep(PAYS)        =+PAYS(*NOMPAYS,*ALTITUDE,*POPULATION)
rep(HEBERGEMENT) =+HEBERGEMENT(*NOMH,*NOMPAYS,*CATEGORIE,*CAPACITE)
rep(SITUATION)   ="NOMPAYS dans rep(HEBERGEMENT)"
```

La requête PROLOG aura, dans ce cas, la forme :

```
- HEBERGEMENT(*X1,MARGERIDE,*X2,*1)-SORTIR(*X1,*X2)
```

On constate que dans l'un et l'autre des cas (REP1 ou REP2), l'élaboration de la requête dépend de la représentation choisie. Précisons un peu le mécanisme de construction de la requête dans le cas de REP1.

(1) - NOMH et CATEGORIE sont le premier et le second attribut de HEBERGEMENT.
- HEBERGEMENT étant représenté par +HEBERGEMENT(NOMH,CATEGORIE, CAPACITE), on y introduit les variables X1 et X2 aux "emplacements" de NOMH et de CATEGORIE ; d'où :

HEBERGEMENT(*X1,*X2,*1)

On substitue la variable *1 à l'attribut CAPACITE qui ne nous intéresse pas dans cette requête.

(2) - "MARGERIDE" est un NOMPAYS (premier attribut de PAYS). Dans le schéma représentant PAYS, substituons "MARGERIDE" à NOMPAYS ; on obtient

PAYS(MARGERIDE,*2,*3)

(3) - SITUATION est représentée par SITUATION(NOMPAYS,NOMH). On substitue MARGERIDE à NOMPAYS et *X1 à NOMH :

SITUATION(MARGERIDE,*X1)

(4) - Regroupons (1), (2), (3) pour constituer la requête PROLOG (ie une conjonction de littéraux négatifs) :

- HEBERGEMENT(*X1,*X2,*1)-PAYS(MARGERIDE,*2,*3)-SITUATION (MARGERIDE,*X1) -SORTIR(*X1,*X2)

(5) - On constate une différence par rapport à l'expression de la requête qui a déjà été donnée. Cette différence tient au fait que certaines "optimisations" sont possibles, sur l'expression obtenue en (4) :

(5.1) - Il est inutile de garder -PAYS(MARGERIDE,*2,*3). En effet, aucune restriction (expression de sélection) ne porte sur ses constituants autre que la clé. En d'autres termes, on n'a pas besoin "d'accéder" à PAYS pour évaluer la requête.

(5.2) - Il serait préférable de mettre en tête de la requête les littéraux dont certains arguments sont instanciés (ie sont des constantes) car cela pourrait avoir une incidence sur les performances.
D'où la nouvelle forme de la requête :

-SITUATION(MARGERIDE,*X1)-HEBERGEMENT(*X1,*X2,*1)-SORTIR(*X1,*X2).

Dans le cas de REP2, on suivra la même démarche. La différence dans l'expression des requêtes tient au fait que SITUATION est représentée par l'inclusion de NOMPAYS dans HEBERGEMENT. Il faut donc en tenir compte.

b) Application à un système relationnel

- Cas 1 : Cas d'une représentation de type REP1

rep(HEBERGEMENT) = RELATION HEBERGEMENT(NOMH,CATEGORIE,CAPACITE)

rep(PAYS) = RELATION PAYS(NOMPAYS,ALTITUDE,POPULATION)

rep(SITUATION) = RELATION SITUATION(NOMPAYS,NOMH)

La requête SELECT NOMH,CATEGORIE FROM HEBERGEMENT
WHERE NOMH = SELECT NOMH FROM SITUATION
WHERE NOMPAYS='MARGERIDE'

peut être construite progressivement de la façon suivante :

(1) - transcription de morceaux de la requête initiale

(1.1) - SORTIR X1,X2 TELQUE X1 = NOMH(HEBERGEMENT)
et X2 = CATEGORIE(HEBERGEMENT)

en SELECT NOMH,CATEGORIE FROM HEBERGEMENT

(1.2) - X3 = NOMPAYS(PAYS) et X3 = 'MARGERIDE'

en SELECT NOMPAYS FROM PAYS WHERE NOMPAYS = 'MARGERIDE'

(1.3) - HEBERGEMENT.SITUATION.PAYS

en SELECT NOMH FROM SITUATION

(2) - regroupement des "morceaux"

```
SELECT NOMH,CATEGORIE FROM HEBERGEMENT
       WHERE NOMH = SELECT NOMH FROM SITUATION
                    WHERE NOMPAYS = SELECT NOMPAYS FROM PAYS
                                    WHERE NOMPAYS='MARGERIDE'
```

(3) - optimisations

Le dernier bloc SELECT étant inutile dans (2), on obtient, après son élimination :

```
SEEECT NOMH,CATEGORIE FROM HEBERGEMENT
       WHERE NOMH = SELECT NOMH FROM SITUATION
                    WHERE NOMPAYS = 'MARGERIDE'
```

- Cas 2 : Cas d'une représentation de type REP2

```
rep(HEBERGEMENT&SITUATION) = RELATION(NOMH,NOMPAYS,CATEGORIE,
                                      CAPACITE)
rep(PAYS)                  = RELATION(NOMPAYS,ALTITUDE,POPULATION)
```

Reprenons l'élaboration de la requête, étape par étape :

(1) - SORTIR X1,X2 TELQUE X1 = NOMH(HEBERGEMENT)
ET X2 = CATEGORIE(HEBERGEMENT)

est traduit en SELECT NOMH,CATEGORIE FROM HEBERGEMENT

- X3 = NOMPAYS(PAYS) et X3 = 'MARGERIDE' est traduit en :

SELECT NOMPAYS FROM PAYS WHERE NOMPAYS = 'MARGERIDE'

- HEBERGEMENT.SITUATION.PAYS est traduit en :

SELECT NOMH FROM HEBERGEMENT

(2) - Le regroupement des éléments de (1) donne :

```
SELECT NOMH,CATEGORIE FROM HEBERGEMENT
       WHERE NOMH = SELECT NOMH FROM HEBERGEMENT
                    WHERE NOMPAYS=SELECT NOMPAYS FROM PAYS
                                  WHERE NOMPAYS=
                                        'MARGERIDE'
```

(3) - L'expression (2) devient :

```
SELECT NOMH,CATEGORIE FROM HEBERGEMENT
       WHERE NOMPAYS='MARGERIDE'
```

après élimination de WHERE NOMH = SELECT NOMH FROM HEBERGEMENT
et substitution de 'MARGERIDE' à

```
SELECT NOMPAYS FROM PAYS
       WHERE NOMPAYS = 'MARGERIDE'
```

V.2.- Principe général de la démarche

Comme le montre cet exemple, le mécanisme de transformation d'une expression abstraite de requête en une expression concrète se décompose en :

- une transcription des constituants de l'expression abstraite en termes de composants de la représentation de la base.
- une fusion des transcriptions en une expression unique correspondant à l'expression de la requête dans le langage de manipulation

du logiciel d'implémentation.

- [Cette fusion pourra éventuellement être suivie de certaines optimisations].

V.2.1.- La transcription

Elle s'appuie sur la fonction de représentation. En effet, c'est la connaissance de REP1 et REP2 qui a aidé, dans l'exemple, à déterminer les expressions concrètes à partir de la même expression abstraite.

La transcription devra alors utiliser,

a) - d'une part, un ensemble d'informations descriptives de la fonction de représentation.

Cet ensemble d'information "documente" la représentation de la base en explicitant la façon dont des phénomènes du monde réel observé sont représentés par des données.

- Elles permettent de décrire des liaisons entre données

Exemple :

REP(HEBERGEMENT) = RELATION HEBERGEMENT(NOMH,CAPACITE,CATEGORIE)
REP(SITUATION) = RELATION(NOMPAYS,NOMHEB)
OU NOMHEB MENE-A HEBERGEMENT PAR PAYS-DE-HEBGT
NOMPAYS MENE-A PAYS PAR HEBGT-DU-PAYS

- Elles peuvent également décrire des "relations sémantiques" entre données

Exemple :

NOMH DANS HEBERGEMENT EST-IDENTIQUE-A NOMHEB DANS SITUATION
NOMPAYS DANS PAYS EST-IDENTIQUE-A NOMPAYS DANS SITUATION

b) - et d'autre part, des règles qui définissent comment utiliser l'explicitation de la représentation.

Exemples de règles :

R1 : Si la représentation d'une association R entre A et B est une relation TR(X,Y) où X et Y sont les clés respectives de A et de B
alors - (1) l'expression abstraite
$R^+(a)$, où $R^+ : A \rightarrow B$, est transcrite en
SELECT Y FROM TR WHERE X = a

- (2) l'expression $R^-(b)$, où $R^- : B \rightarrow A$, est traduite en
SELECT X FROM TR WHERE Y = b

- (3) R(a,b), où $R : A \times B \rightarrow BOOL$, est transcrite en
SELECT X,Y FROM TR WHERE X = a AND Y = b

R2 : Si la représentation de l'association R entre A et B est une relation RA constituée par les attributs de A et la clé Y de B [ie RA(X1,X2,...,Xn,Y) où $X_1,\ldots,X_n$ sont les attributs de A (X1 en étant la clé) et Y la clé de B].
alors - (1) $R^+(a)$ est transcrit en
SELECT Y FROM RA WHERE X1 = a

- (2) $R^-(b)$ est transcrit en
SELECT X1 FROM RA WHERE Y = b

- (3) R(a,b) l'est en
SELECT X1,Y FROM RA WHERE X1 = a AND Y = b

R3 :Si l'expression abstraite est de la forme $f_X(A)$ où f_X est un symbole de fonction d'accès à un attribut X de l'agrégat A alors elle est transcrite en

SELECT X FROM A

R4 : ....

Donc la réalisation de la transcription nécessite :

- une connaissance supplémentaire de la part du système. Cette connaissance est apportée :
  - par la description des choix faits lors du passage de la spécification abstraite à une représentation
  - par des règles d'utilisation de cette description.
- et évidemment des mécanismes "ad hoc" pour l'exploiter.

REMARQUE :

Dans une approche par le modèle relationnel, le problème de la transformation d'une expression abstraite en une expression concrète serait celui de la transcription d'une requête exprimée sur la relation universelle en une requête exprimée sur le schéma logique de la base qui serait, par exemple, le résultat de l'application d'un algorithme de décomposition de la relation universelle.

Dans cette optique, la transcription nécessite que l'on préserve les choix de décomposition que l'algorithme a fait durant son déroulement. (ie quelles sont les dépendances utilisées à chaque étape et quel est le résultat de l'étape).

La disponibilité de ces informations et celle des règles pour les exploiter seraient alors les matériaux essentiels du TRADUCTEUR.

V.2.2.- La fusion des transcriptions

Elle consiste à regrouper en une seule expression les fragments

de transcription. Elle peut mettre en oeuvre des mécanismes spécifiques pour ordonner les fragments, en éliminer certains...

Cette phase constituerait sûrement la partie la plus délicate du TRADUCTEUR.

Exemple :

- Dans l'exemple précédent nous avons d'emblée regroupé

  SELECT NOMH FROM HEBERGEMENT
  et SELECT CATEGORIE FROM HEBERGEMENT
  en SELECT NOMH, CATEGORIE FROM HEBERGEMENT

- puis nous avons éliminé certaines redondances d'expression comme SELECT NOMH, CAPACITE FROM HEBERGEMENT
  [WHERE NOMH = SELECT NOMH FROM HEBERGEMENT]WHERE...

  que nous avons transformé en

  SELECT NOMH, CAPACITE FROM HEBERGEMENT WHERE...

REMARQUE : Le traducteur existant actuellement dans SINDBAD ne considère que des représentations de type REP1.

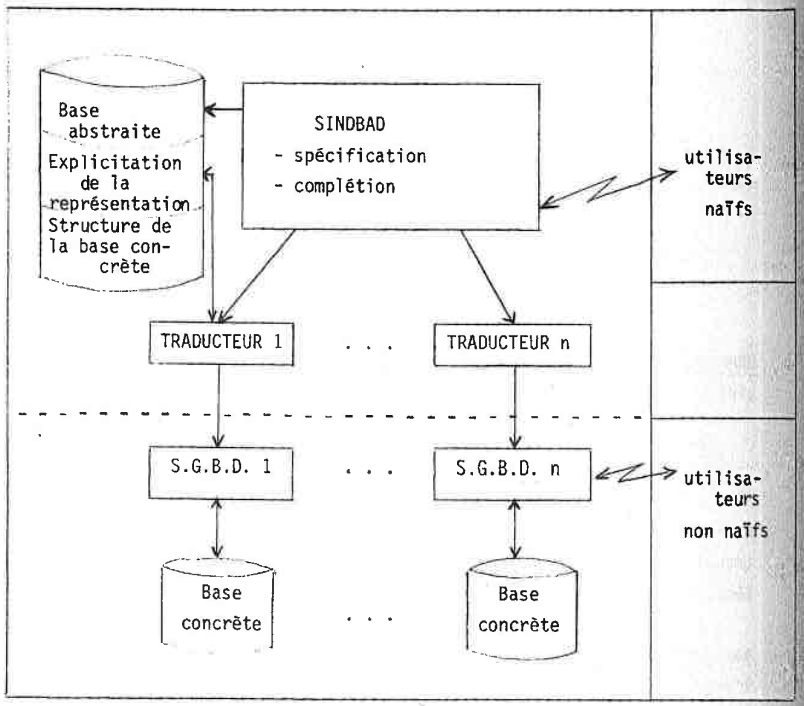
V.3.- Conclusion

L'exposé de ces quelques réflexions sur le passage d'une expression abstraite de requête à son expression concrète montre le travail qu'il reste encore à accomplir avant d'aboutir à un système complet (compréhension et évaluation de requêtes incomplètes).

Outre la facilité et la souplesse d'utilisation qu'un tel système offrirait à son(ses) utilisateurs(s) naïf(s), il aurait aussi l'avantage de rendre l'expression des requêtes indépendantes de la structure de représentation. Il permettrait ainsi au système de s'accomoder de certains changements dans la représentation, sans

incidence sur les expressions initiales des requêtes ni sur les mécanismes de complétion.

Par ailleurs, la dissociation de la compréhension et de l'évaluation fait que ce type de système pourrait être multi-cibles, comme schématisé ci-dessous.



- Une architecture d'un système "complet" -

CONCLUSION

Notre conclusion comportera trois volets :

- d'abord, un bref rappel des idées essentielles de notre travail.
- ensuite, une analyse critique de certaines de nos propositions.
- enfin une présentation de quelques perspectives de développement que nous envisageons comme suite à ce travail.

Ce travail a voulu montrer que les abstractions de données pouvaient être d'un grand apport pour élaborer des systèmes à capacités de déduction. En effet, elles supportent un formalisme rigoureux et des techniques de spécification dont l'utilisation et la mise en oeuvre permettent de concevoir et de réaliser progressivement la base sur laquelle portera la déduction.

Les mécanismes d'abstraction dégagent les utilisateurs des contraintes de représentation. La disponibilité d'outils d'aide à la spécification permet ainsi de "dissimuler" des aspects formels qui pourraient "effrayer l'utilisateur moyen".

Par ailleurs, la "hiérarchisation" des utilisateurs en fonction de leur degré de spécialisation nous a conduit à une hiérarchisation des langages qui permettent d'opérer sur la base. Nous avons, dans cette optique, montré comment :

- d'une part, on pouvait agir sur la base en utilisant le langage de manipulation de données de VEGA.
- et d'autre part, permettre une utilisation "presse-bouton" ne nécessitant pas de spécialisation de la part de ses auteurs.

Nous avons également mis l'accent sur le fait, très important, que l'expression des requêtes devait être indépendante de leur mode d'évaluation. En d'autres termes, que leurs réponses soient extraites directement à partir de données mémorisées, ou par l'activation

d'un processus de déduction qui met en jeu des données mémorisées, et des règles logico-mathématiques préalablement définies, n'a aucune espèce d'incidence sur la forme d'écriture des dites requêtes.

La description de nos expérimentations a permis de mettre en relief certaines insuffisances des systèmes de gestion de bases de données traditionnels.

Les problèmes relèvent, notamment, de leur incapacité à prendre en compte la définition, l'implémentation et l'utilisation de nouveaux types de données, et de l'indisponibilité de mécanismes généraux de déduction.

Elle a également permis de déceler les limites du logiciel utilisé, ouvrant ainsi de nouvelles perspectives de développement, qui font déjà l'objet de réflexions au sein de l'équipe.

En effet, VEGA, "construit sur PROLOG", s'est avéré un outil efficace pour la réalisation de maquettes, du fait de ses capacités (abstraction et déduction) et du peu d'effort de programmation qu'il requiert. Il est par contre inadapté dans un contexte d'application réelle, compte tenu du niveau assez faible de ses performances (en corrélation avec celles de PROLOG) et des capacités réduites de mise à jour de données qu'il offre.

Aussi, une architecture de système où seraient dissociées déduction et gestion des données serait peut être une solution plus efficace. Elle est actuellement à l'étude.

Sur un autre plan, les propositions et les expérimentations menées pour élargir les possibilités d'utilisation naïve d'une base visaient à réduire le degré de spécialisation requis de la part d'un utilisateur. Elles l'autorisent à exprimer ses requêtes à l'aide d'un langage simple, peu contraignant et qui permet imprécisions et ambiguïtés. Un système dit de compréhension essaie d'aboutir, avec le concours éventuel de l'utilisateur, à une forme

d'expression des requêtes où certains sous-entendus auront été explicités et où il ne subsiste plus ni ambiguïté ni imprécision.

Sur ce chapitre nous avons porté un certain nombre de réflexions qui, selon nous, permettraient d'accroître le degré d'intelligence du système dans son activité de complétion des requêtes. En effet, actuellement, celle-ci s'exerce sur des critères essentiellement syntaxiques (ce système complète la requête, et l'utilisateur décide de la validité du résultat produit). L'introduction d'un autre niveau de connaissance et des mécanismes appropriés pour son exploitation contribuerait sûrement à accroître le "pouvoir de décision" du système tout en en améliorant les performances et en réduisant l'intervention de l'utilisateur.

Mais, outre cet aspect, d'autres extensions sont également envisageables :

- la plus urgente concerne évidemment la réalisation d'un traducteur du résultat produit par la compréhension de requêtes en une phrase (requête ou programme) du langage du logiciel de gestion de la base. (L'outil de traduction d'une requête complétée en une requête PROLOG, que nous avons réalisé, quoique rudimentaire, constitue néanmoins un embryon de traducteur).
- une autre extension concernerait les aspects de "mise à jour déductive" de données, c'est-à-dire étudier les voies et moyens qui permettraient d'exploiter le "savoir" du système pour déterminer les incidences éventuelles d'une modification, a priori locale, d'une donnée sur d'autres données de la base. (C'est l'étude du problème dit de propagation des mise à jour).
- un autre aspect concernerait les possibilités d'interrogation à l'aide d'une langue naturelle (ou un de ses sous-ensembles). [COUL 81]. Le langage des bribes que nous avons défini et le système qui en a la charge seraient des intermédiaires au sens où

la reconnaissance d'une expression en langue naturelle produirait des résultats similaires à ceux produits par l'analyse d'une expression incomplète dans le système actuel. La complétion pourrait alors s'accomplir selon le processus existant.

- enfin, des recherches plus fondamentales peuvent également faire l'objet d'efforts. Elles concerneraient la justification théorique des mécanismes utilisés parfois (comme l'utilisation de types paramétrés comme paramètres d'un type ou encore l'utilisation d'une spécification en paramètre [TERR 83]). Ces recherches pourraient également concerner l'approfondissement de la notion de dérivation relationnelle pour éventuellement prouver formellement la validité du résultat de la complétion.

L'exposé de ces quelques extensions et perspectives montre bien l'ampleur des travaux qu'il reste à entreprendre.

Pouvait-on alors vraiment parler de conclusion... ?

ANNEXE I

ÉLÉMENTS DE LOGIQUE DU PREMIER ORDRE

Dans cette annexe nous définissons les notions essentielles de logique utilisées dans certains chapitres de la thèse. Pour plus de précisions, le lecteur intéressé pourra se référer aux ouvrages spécialisés (dont [KLEENE 71], [KORF 66], [MANN 74],...).

Un système logique peut être défini selon deux points de vue : l'un syntaxique et l'autre sémantique ; les deux sont basés sur un langage que nous allons commencer par définir.

1.- Le Langage de la logique du premier ordre

a) Le langage (réduit) comporte des symboles primitifs

- les symboles de variables (dénotés par x,y,z,...) et de constantes (dénotés par a,b,...).
- les symboles de fonctions (dénotés f,g,h...).
- les symboles de prédicats (dénotés par des lettres majuscules).
- les connecteurs logiques $\neg$ (négation) et $\rightarrow$ (implication).
- le quantificateur universel $\forall$
- (et).

les formules "légales" du langage (ou formules bien formées) sont alors construitesà l'aide de ces symboles primitifs, et à l'aide des termes et des formules atomiques.

b) Les termes sont définis récursivement comme suit :

(1) une constante est un terme.
(2) une variable est un terme.
(3) Si f est un symbole de fonction n-aire (à n arguments)

et $t_1, t_2, \ldots, t_n$ des termes alors $f(t_1, \ldots, t_n)$ est un terme.

(4) (1), (2), (3) sont les seuls termes possibles.

c) Formule atomique

- si P est un symbole de prédicat n-aire et $t_1, t_2, \ldots, t_n$ des termes alors $P(t_1, t_2, \ldots, t_n)$ est une formule atomique.
- si n = 0, la formule atomique est appelée proposition.
- une formule atomique (ou sa négation) sera appelée littéral.

d) Une formule bien formée (ou tout simplement formule) est définie comme suit :

(1) une formule atomique est une formule.

(2) Si A est une formule et x une variable alors $(\forall x)$ A est une formule.

(3) Si A et B sont des formules alors $\neg(A)$ et $(A) \rightarrow (B)$ sont des formules.

(4) les seules formules possibles sont celles obtenues par l'application, un nombre fini de fois, de (1), (2), (3).

REMARQUE :

Le langage n'a été défini qu'à l'aide des connecteurs logiques $\neg$ et $\rightarrow$ et du quantificateur universel. Les autres connecteurs $\wedge$(et), $\vee$(ou), $\leftrightarrow$ (équivalence logique) et le quantificateur existenciel ($\exists$) peuvent être exprimés à l'aide des symboles primitifs :

- $A \wedge B$ s'exprime par $\neg(A \rightarrow \neg B)$
- $\exists x A$ " $\neg(\forall x \neg A)$
- $A \vee B$ " $(\neg A \rightarrow B)$
- $A \leftrightarrow B$ " $(\neg A \vee B) \wedge (A \vee \neg B)$

e) Autre définition

- Une formule close est une formule dont toutes les variables

sont quantifiées (ou liées) ; c'est-à-dire qu'aucune de ses variables n'est libre.

2.- Le point de vue sémantique (Théorie des modèles)

Il vise à donner une "signification" à une formule. On dit que l'on donne une interprétation de la formule.

Dans le calcul propositionnel, une interprétation est l'assignation des valeurs VRAI ou FAUX aux formules atomiques.

Dans la logique du premier ordre, l'interprétation d'une formule nécessite le choix d'un domaine (dit domaine d'interprétation) et l'assignation de "valeurs" à chaque constante, symbole de fonction et symbole de prédicat de la formule :

- A chaque symbole de constante on associe un élément de D.
- A chaque symbole de fonction n-aire on associe une fonction de $D^n \rightarrow D$.
- A chaque symbole de prédicat n-aire, une fonction de $D^n \rightarrow Bool$.

La spécification du domaine D et de ces associations définit une interprétation ou un modèle de la formule.

Autres définitions :

- Une interprétation d'un ensemble $\mathcal{F}$ de formules est un modèle de $\mathcal{F}$ si et seulement si toute formule F de $\mathcal{F}$ est vraie dans cette interprétation.
- Une formule F est une conséquence logique d'un ensemble $\mathcal{F}$ de formules si F est vraie dans tout modèle de $\mathcal{F}$. (On le note $\mathcal{F} \models F$).
- Une formule F est valide si elle est vraie pour toute interprétation (ce que l'on note $\models F$).

3.- Le point de vue syntaxique (Théorie de la démonstration)

Sur un langage, comme celui défini au § 1, on peut définir un système formel (ou une théorie) en considérant :

- Un certain nombre (de schémas) d'axiomes dits axiomes logiques

  Comme : (1) $P \to (Q \to P)$

  (2) $(P \to (Q \to R)) \to ((P \to Q) \to (P \to R))$

  (3) $(\neg P \to \neg Q) \to (Q \to P)$

  (4) $\forall x(P \to Q) \to (P \to \forall xQ)$, x n'étant pas une variable libre de P.

  (5) $\forall xP \to P[t/x]$: substitution du terme t à x telle que t ne contienne aucune variable liée de P. (Voir définition de la substitution plus loin).

- des règles d'inférence

  (1) modus ponens : $P, P \to Q \vdash Q$
  (qui se "lit" de P et P implique Q inférer Q)

  (2) la généralisation : $P \vdash \forall x\, P$

Définitions :

- Une démonstration formelle d'une formule F est une suite finie d'occurrences de formules $F_0, F_1, \ldots, F$ où chacune des formules est :
  - soit un axiome
  - soit issue de l'application d'une règle d'inférence à des formules qui la précèdent dans la suite.

- Si une démonstration existe pour une formule F, on dit que F est démontrable (formellement) ou encore que F est un théorème (que l'on note $\vdash F$).

- Si on ajoute d'autres axiomes, on obtient un système formel appelé THEORIE du PREMIER ORDRE. Les axiomes ajoutés sont appelés axiomes propres.

- On appellera alors déduction une suite finie d'occurrences de formules $F_1, F_2, \ldots, F_n$ où chaque formule est :
  - soit un axiome logique
  - soit un axiome propre
  - soit le résultat de l'application d'une règle d'inférence à des formules qui la précèdent dans la suite.

- On notera $\underset{T}{\vdash} F$ un théorème F déductible à partir des axiomes d'une théorie T.

- Une formule F est une conséquence d'un ensemble $\mathcal{F}$ de formules si F est dérivable à partir des axiomes d'une théorie T et des formules de $\mathcal{F}$. (On la notera $\mathcal{F} \underset{T}{\vdash} F$).

- Un modèle d'une théorie T est une interprétation dans laquelle tous les axiomes de T sont vrais et où un théorème dérivable d'un ensemble d'axiomes est vrai dans tout modèle de ces axiomes.

Les points de vue, syntaxique (ou théorie de la démonstration ou de la preuve) et sémantique (ou théorie des modèles) sont des méthodes différentes, mais néanmoins équivalentes, pour mener des raisonnements. Dans la première approche on construit la preuve de la formule alors que dans la seconde on en teste la validité.

GÖDEL [en 1930] a démontré l'équivalence des deux points de vue. La relation entre eux est établie par les résultats de consistance et de complétude qui stipulent respectivement :

(1) si F est démontrable à partir d'un ensemble $\mathcal{F}$ d'hypothèses alors F est valide

$$\mathcal{F} \vdash F \to \mathcal{F} \models F$$

(2) si F est vraie dans tous les modèles d'un ensemble $\mathcal{F}$ d'axiomes, alors F est démontrable à partir de l'ensemble $\mathcal{F}$ d'hypothèses

$$\mathcal{F} \models F \rightarrow \mathcal{F} \vdash F$$

4.- Substitution - Unification

a) Substitution

- On appelle substitution σ un ensemble de couples $\langle x_i, t_i \rangle$ où x_i sont des variables et t_i des termes.

- L'application d'une substitution à un terme t (notée $t\sigma$) est le terme obtenu à partir de t dans lequel on a remplacé toute occurrence de x_i par t_i.

- On définit $t\sigma$ par récurrence sur la complexité des termes.
  - $a\sigma = a$
  - $x\sigma$ = si x est un des x_i alors t_i sinon x
  - $f(u_1,u_2,\ldots,u_n) = f(u_1\sigma,u_2\sigma,\ldots,u_n\sigma)$.

b) Unification

L'unificateur de deux termes t_1 et t_2 est une substitution σ telle que : $t_1\sigma = t_2\sigma$.

5.- Clause - Forme clausale - Clause de HORN

- Une clause est une formule de la forme

$$B_1 \vee B_2 \vee \ldots \vee B_n \leftarrow A_1 \wedge A_2 \wedge \ldots \wedge A_n$$

où A_i et B_j sont des formules atomiques.

- Si m = 1, la clause est dite de HORN

$$B \leftarrow A_1 \wedge A_2 \wedge \ldots \wedge A_n$$

- Une forme clausale est une formule de la forme

$C_1 \wedge C_2 \wedge \ldots \wedge C_k$ où les C_i sont des clauses dont les variables sont toutes quantifiées universellement.

- Notation
  - Une clause est généralement notée $B_1,B_2,\ldots,B_m \leftarrow A_1,A_2,\ldots,A_n$.
  - Une clause de HORN $B \leftarrow A_1,A_2,\ldots,A_n$.
  - Une forme clausale $\begin{matrix} C_1 \\ \vdots \\ C_k \end{matrix}$.

ANNEXE II

LE MODÈLE RELATIONNEL DE DONNÉES

Les rappels sur le modèle relationnel de données que contient cette annexe sont pour l'essentiel tirés de [CODD 70], [CADI 75], [BERN 76], [FAGI 77], [FAGI 82], [DELO 78], [DELO 82], [FLOR 82]. (On trouvera dans ces diverses références les théorèmes et démonstrations sur lesquels sont fondées les notions que contient cette annexe).

I - RELATION ET SCHEMA DE RELATION

En mathématiques, une relation est une partie d'un produit cartésien défini sur une liste de domaines, un domaine étant un ensemble de valeurs.

- On définira ainsi une relation n-aire sur les domaines D_1, $D_2,\ldots,D_n$ comme étant une partie du produit cartésien $D_1 \times D_2 \times \ldots \times D_n$.
- Les éléments d'une telle relation sont appelés n-uplets et notés $(d_1,d_2,\ldots,d_n)$ où $d_i \in D_i$, $\forall i \in [1,n]$.
- On notera également $(d_1,d_2,\ldots,d_n) \in R$ l'appartenance d'un n-uplet à une relation.
- Un attribut A_i (ou constituant) de la relation est un identificateur auquel est associé un domaine D_i. Il peut également être vu comme une variable prenant ses valeurs dans D_i.
- Souvent une propriété (ou une liste de propriétés) P doit être vérifiée par les n-uplets du produit cartésien $D_1 \times D_2 \times .. \times D_n$ pour qu'ils soient membres (éléments, occurrences) de la relation définie sur $D_1,D_2,\ldots,D_n$.

On appellera alors schéma de relation la donnée
- des constituants $A_1, A_2, \ldots, A_n$
- de leurs domaines respectifs $D_1, D_2, \ldots, D_n$
- de la propriété P que doit respecter tout n-uplet de la relation $D_1 \times D_2 \times \ldots \times D_n$.

Ainsi un schéma de relation définit l'intention de celle-ci. Le terme relation est alors utilisé pour désigner son extension.

- On notera quand nécessaire, un schéma d'une relation R par R(U,P) où U et P désignent respectivement la liste de constituants et celle des propriétés que doivent vérifier les n-uplets de la relation.

- Un schéma relationnel $\mathcal{S}$ d'une base de données est alors défini par un ensemble de schémas de relation.

$$\mathcal{S} = \{R_i(U_i, P_i) \mid i = 1, \ldots, m\}$$

Il sera également noté $\mathcal{S}(\mathcal{R}, \mathcal{P})$.

- Une base de données est, dans ce contexte, une collection, variable dans le temps, de relations.

- Enfin on représentera généralement une relation par un tableau dont les colonnes désignent les constituants et les lignes les n-uplets.

ex : RELATION HEBERGEMENT(NOMH, CAPACITE, CATEGORIE)

NOMH	CAPACITE	CATEGORIE
LE NID	5	2*
LES PIN-GOUINS	8	3*

II - LE LANGAGE ALGEBRIQUE

Un certain nombre d'opérateurs relationnels permettent de manipuler les relations, c'est-à-dire qu'appliqués à des relations-opérandes, ils produisent des relations-résultats.

Dans ce paragraphe, les exemples seront construits sur les domaines $Dom(A) = \{a_1, a_2, a_3\}$
$Dom(B) = \{b_1, b_2, b_3\}$
$Dom(D) = \{d_1, d_2, d_3\}$

1.- Somme et Produit

Soient R(X) et S(Y) deux relations et $Z = X \cup Y$

- somme : $R(X)+S(Y) = T(Z)$ également noté R+S ou (R+S)(Z)
avec $T(Z) = \{(x,y) \mid x \in R \wedge y \in S\}$

- produit : $R(X)*S(Y) = T(Z)$ (ou $(R*S)(Z)$)
avec $T(Z) = \{(x,y) \mid x \in R \wedge y \in S\}$

Exemple : Soit R(A,B) et S(A,D) avec leurs occurrences ci-dessous

R

A	B
a_1	b_1
a_1	b_2
a_2	b_3

S

A	D
a_1	d_1
a_1	d_2
a_2	d_1

R*S = T :

A	B	D
a_1	b_1	d_1
a_1	b_1	d_2
a_1	b_2	d_1
a_1	b_2	d_2
a_2	b_3	d_1

et R+S = T :

A	B	D
a_1	b_1	d_1
a_1	b_1	d_2
a_1	b_2	d_1
a_1	b_2	d_2
a_2	b_3	d_1
a_2	b_3	d_2
a_1	b_1	d_3
a_1	b_2	d_3

2.- Produit cartésien

Soient deux relations R(X) et S(Y) où les constituants X et Y sont disjoints (Si cela n'était pas le cas, on les rend disjoints en procédant à un renommage des constituants communs à R et à S).

Le produit cartésien de R par S noté R×S est une relation T résultat du produit de R par S

$$T = R \times S = R * S$$

Exemple : Le produit cartésien de R par S, R et S étant les relations du paragraphe précédent est :

R×S :

A	B	A'	D
a_1	b_1	a_1	d_1
a_1	b_1	a_1	d_2
a_1	b_1	a_2	d_1
a_1	b_2	a_1	d_1
a_1	b_2	a_1	d_2
a_1	b_2	a_2	d_1
a_2	b_3	a_1	d_1
a_2	b_3	a_1	d_2
a_2	b_3	a_2	d_1

3.- Union et intersection

L'union (U) et l'intersection (∩) de deux relations R(X) et S(X) sont respectivement les relations R∪S = R+S

et R∩S = R*S

Exemple :

R :

A	B
a_1	b_1
a_1	b_2

S :

A	B
a_2	b_2
a_1	b_2

R∪S :

A	B
a_1	b_1
a_1	b_2
a_2	b_2

R∩S :

A	B
a_1	b_2

4.- Complément et différence

- Complément : $\neg R(X)$

Le complément est défini par : $\neg R(X) = \{x | x \in X \wedge x \notin R(X)\}$

- Différence de deux relations R(X) et S(X), notée R-S :

$$(R-S)(X) = R(X) * \neg S(X)$$

Exemple :

(1) $\neg R(A,B)$ est définie par les occurrences de la relation ci-dessous :

$\neg R(A,B)$

A	B
a_1	b_3
a_2	b_1
a_2	b_2
a_3	b_1
a_3	b_2
a_3	b_3

(2) En considérant les relations R et S du paragraphe 3, R-S ne contient que $\{a_1, b_1\}$.

5.- Projection et anti-projection

- La projection d'une relation R(X,Y) sur Y, notée R(X,Y)[Y] est une relation S(Y) telle que :

$$\forall y \in S(Y), \exists x : (x,y) \in R(X,Y)$$

(Schématiquement, la relation S est constituée par la(les) colonne(s) Y de la relation R).

- L'anti-projection d'une relation R(X,Y) sur Y, notée R(X,Y)]Y[est une relation S(Y) telle que :

$$S(Y) = \{y \mid (a,y) \in R(X,Y) \text{ pour tout } a, a \in R(X,Y)[X]\}$$

Exemple :

R(A,B)

A	B
a_1	b_1
a_1	b_2
a_2	b_1
a_3	b_1

R(A,B)[A]

A
a_1
a_2
a_3

R(A,B)]B[

B
b_1

6.- Division

La division de R(X,Y) par S(Y), notée $R \div S$, est une relation T(X) définie par :

$$T(X) = \{x \mid \forall y \in S(Y), (x,y) \in R(x,y)\}$$

Exemple :

R(A,B)

a_1	b_1
a_1	b_2
a_2	b_2
a_3	b_1

S(A)

a_1
a_2

$R \div S$

b_1
b_2

7.- Sélection

La sélection est une opération permettant de sélectionner dans une relation des n-uplets qui satisfont à une propriété donnée (dite expression de sélection). Elle est notée : R : E, et ses occurrences sont celles de R qui vérifient E :

$$R(X) : E(X) = \{x \mid x \in R(X) \wedge E(x)\}$$

Exemple : Sur la relation R(A,B) du § 6

$$R(A,B) : (B = b_1) = \{(a_1,b_1),(a_3,b_1)\}$$

III- DEPENDANCES FONCTIONNELLES ET CLE DE RELATION

1.- Dépendance fonctionnelle (DF)

Soit R(X,Y,Z) avec Z éventuellement vide.

On dit que X dépend fonctionnellement de Y (notée $X \xrightarrow{R} Y$) dans la relation R si et seulement si

$$\forall (x,y,z),(x',y',z') \; [(x,y,z) \in R \text{ et } (x',y',z') \in R]$$

on a $x = x' \rightarrow y = y'$.

En d'autres termes, la donnée d'une valeur x de X dans R détermine une seule valeur y de Y dans R.

Exemple : Dans une relation R (PAYS, HEBREGEMENT), on a la dépendance fonctionnelle HEBERGEMENT → PAYS.

- Une DF est élémentaire si pour tout $X' \subset X$, $X' \xrightarrow{R} Y$ (La DF $X' \rightarrow Y$ n'est pas vérifiée pour tout sous-ensemble de X).

- Une DF $X \rightarrow Y$ est directe
  - si elle est élémentaire
  - et qu'il n'existe pas Z tel que $X \rightarrow Z$, $Z \rightarrow Y$

- Un ensemble $\mathcal{F}$ de dépendances fonctionnelles est sous forme canonique si chaque dépendance fonctionnelle de $\mathcal{F}$ ne comporte qu'un seul constituant en partie droite.

2.- Propriétés des dépendances fonctionnelles

Les dépendances fonctionnelles vérifient un certain nombre de

propriétés qui établissent des relations entre DF. Ces propriétés ou règles sur les DF peuvent être utilisées pour découvrir (on dira dériver) de nouvelles DF.

[Dans notre sous-système de complétion de requêtes (Partie C, chap. IV) nous mettons en oeuvre un processus similaire de dérivation des "sous-entendus" dans une requête, à partir des informations contenues dans la requête et des connaissances qu'à le système des composants de la base sur laquelle s'applique la requête].

Soit une relation $R(U, \mathcal{F})$ où U désigne la liste de ses constituants et $\mathcal{F}$ celle des dépendances fonctionnelles. On a alors les propriétés suivantes :

FR1 : Réflexivité : si $X \subseteq Y \subseteq U$ alors $Y \rightarrow X$

FR2 : Augmentation : si $X \rightarrow Y$ et $Z \subseteq W \subseteq U$ alors $X \cup W \rightarrow Y \cup Z$

FR3 : Transitivité : si $X \rightarrow Y$ et $Y \rightarrow Z$ alors $X \rightarrow Z$

FR4 : Pseudo-transitivité : si $X \rightarrow Y$ et $Y \cup W \rightarrow Z$ alors $X \cup W \rightarrow Z$

FR5 : Union : si $X \rightarrow Y$ et $X \rightarrow Z$ alors $X \rightarrow Y \cup Z$

FR6 : Décomposition : si $X \rightarrow Y$ et $Z \subseteq Y$ alors $X \rightarrow Z$

- On appelle fermeture d'un ensemble $\mathcal{F}$ de dépendances fonctionnelles, l'ensemble $\mathcal{F}'$ de dépendances fonctionnelles que l'on peut obtenir à partir de $\mathcal{F}$ par l'application des règles précédentes.

- On appelle couverture minimale de $\mathcal{F}$, l'ensemble $\mathcal{F}''$ de dépendances fonctionnelles vérifiant les propriétés suivantes :
  (1) $\mathcal{F}'' \subseteq \mathcal{F}$
  (2) toute dépendance fonctionnelle de $\mathcal{F}''$ est élémentaire
  (3) la fermeture de $\mathcal{F}''$ est égale à la fermeture de $\mathcal{F}$
  (4) $\mathcal{F}''$ est minimal, c'est-à-dire qu'il n'existe pas de partie $\mathcal{F}''_1$ de $\mathcal{F}''$ dont la fermeture est égale à celle de $\mathcal{F}$.

3.- Clé d'une relation

X est clé d'une relation R(X,Y,Z) si $X \rightarrow Y \cup Z$ est élémentaire.

La clé d'une relation est constituée par le(s) constituant(s) dont les valeurs permettront de distinguer les n-uplets de la relation (c'est-à-dire que tout n-uplet aura pour valeur de sa clé une valeur différente de celles des clés de tous les autres n-uplets).

IV - LES DEPENDANCES MULTIVALUEES (DM)

Les dépendances multivaluées sont un autre type de dépendance qui peut exister entre des constituants d'une relation. Elles sont également dotées de propriétés.

Par ailleurs des règles définissent leurs interactions avec les DF.

1.- Dépendance multivaluée

Soit une relation R(X,Y,Z) avec X,Y,Z disjoints deux à deux. On dira qu'il existe une DM notée $X \twoheadrightarrow Y$

si $Y_{xz} = Y_{xz}$, avec $X_{xz} = \{y \mid (x,y,z) \in R\}$

En d'autres termes, les valeurs de Y attachées à (x,z) sont identiques à celles attachées à x indépendamment de z.

Exemple : Dans la relation (HEBERGEMENT,PAYS), une DM existe

PAYS $\twoheadrightarrow$ HEBERGEMENT

(Elle exprime la liaison entre un pays et tous les hébergements qui y sont situés).

2.- Propriétés des DM

Soit une relation R(U)

MR1 : Réflexivité : si $Y \subseteq X$ alors $X \twoheadrightarrow Y$

MR2 : Augmentation : si $V \subseteq W$ et $X \twoheadrightarrow Y$ alors $X \cup W \twoheadrightarrow Y \cup V$

MR3 : Complémentation : si $X \twoheadrightarrow Y$ est vraie $X \twoheadrightarrow U-(X \cup Y)$

MR4 : Transitivité : si $X \twoheadrightarrow Y$ et $Y \twoheadrightarrow Z$ alors $X \twoheadrightarrow Z-Y$

MR5 : Pseudo-transitivité : si $X \twoheadrightarrow Y$ et $Y \cup W \twoheadrightarrow Z$
alors $X \cup W \twoheadrightarrow Z-(Y \cup W)$

MR6 : union : si $X \twoheadrightarrow Y$ et $X \twoheadrightarrow Z$ alors $X \twoheadrightarrow Y \cup Z$

MR7 : Décomposition : si $X \twoheadrightarrow Y$ et $X \twoheadrightarrow Z$ alors $X \twoheadrightarrow Y \cup Z$
$X \twoheadrightarrow Y-Z$

3.- Propriétés sur DF et DM

XR1 : si $X \rightarrow Y$ alors $X \twoheadrightarrow Y$

XR2 : si $X \twoheadrightarrow Z$ et $Y \rightarrow Z'$ avec $Z' \subseteq Z$ et $Y \cap Z = \emptyset$
alors $\twoheadrightarrow Z'$

XR3 : si $X \twoheadrightarrow Y$ et $X \cup Y \rightarrow Z$ alors $X \rightarrow Z-Y$

4.- Consistance et complétude d'un système de règles de dérivation

Les diverses règles définies sur les DF, les DM et entre DF et DM permettent de dériver de nouvelles dépendances à partir d'un ensemble initial de dépendances.

Un résultat fondamental concernant les dépendances est qu'il existe un ensemble de règles de dérivation qui est valide et complet.

Nous nous contenterons ici de donner quelques définitions et d'énoncer la propriété de complétude et de validité du système de règles. On trouvera certaines démonstrations notamment dans [DELO 82 - chap. 10].

• Une dépendance fonctionnelle f est une conséquence logique d'un

ensemble $\mathcal{F}$ de dépendances fonctionnelles, notée $\mathcal{F} \models f$, si f est vérifiée dans toutes les relations R de schéma $(U, \mathcal{F})$.

• Un ensemble de règles de dérivation est valide si lorsque f est dérivée de $\mathcal{F}$ alors f est une conséquence logique de $\mathcal{F}$.

encore $\mathcal{F} \vdash f \rightarrow \mathcal{F} \models f$.

• Un ensemble de règles de dérivation est complet si lorsque f est une conséquence logique de $\mathcal{F}$ alors f est dérivable de $\mathcal{F}$

soit $\mathcal{F} \models f \rightarrow F \vdash f$

Proposition : Les règles de dérivation - FR1,FR2,FR3,FR4
- MR1,MR2,MR3,MR4,MR5
- et XR1,XR2

forment un système complet et valide.

REMARQUE : DELOBEL a défini un autre type de dépendance, la dépendance hiérarchique, qui exprime une dépendance multivaluée sur une projection de la relation sur laquelle elle porte.

Il définit également ses propriétés. Nous n'en dirons pas plus ici. Le lecteur pourra se référer à [DELO 78] et [DELO 82] pour plus de détails.

IV - LES FORMES NORMALES

L'intérêt des formes normales des relations est de réduire ou d'éliminer des anomalies qui peuvent se produire lors d'opérations de mise à jour portant sur la relation (cf. partie A - chap. I § III.2).

CODD avait initialement défini 3 formes normales. D'autres, ont, par la suite, été introduits. Nous en donnons les définitions ci-dessous.

- Une relation est en première forme normale (1NF) si tous ses constituants sont élémentaires.
- Une relation de schéma $R(U,\mathcal{F})$ est en deuxième forme normale (2NF) si tous les constituants non clés de R dépendent de façon élémentaire de la clé de R.
- Une relation de schéma $R(U,\mathcal{F})$ est en troisième forme normale (3NF) si tous les constituants non clés dépendent de façon élémentaire et directe de la clé de R.
- Une relation R est en troisième forme normale de BOYCE-CODD-KENT (3BCKNF) si chaque fois que $X \rightarrow A$, où A est un constituant élémentaire et $A \notin X$, est vérifiée alors X contient une clé de R.

En d'autres termes, dans une relation en 3BCKNF, toutes les dépendances non triviales se déduisent de la connaissance des clés de la relation.

- Une relation $R(U,\mathcal{F})$ est en quatrième forme normale si chaque fois que la dépendance non triviale $X \twoheadrightarrow Y$ est vérifiée alors pour tout A, $A \in U$, $X \rightarrow A$.

Remarque : Pour l'essentiel, ces définitions ont été prise de [DELO 82].

V- NOTION DE DECOMPOSITION

Les dépendances entre données jouent un rôle dans le processus de décomposition des relations.

- Décomposition binaire :

Une relation R(X,Y,Z) est décomposable suivant la décomposition ({X,Y},{X,Z}) s'il existe deux relations R1 et R2 telles que

(1) R1 = R[X,Y] et R2 = R[X,Z] (R1 et R2 sont les projections de R selon {X,Y} et {X,Z} respectivement).

(2) R = R1*R2 (R est le produit de R1 et R2).

De par cette définition, la décomposition est un processus réversible :

- R1 et R2 sont issues de R
- R peut être engendrée à partir de R1 et R2.

De ce fait, son application assure la préservation des informations de R.

- Décomposition n-aire

Etant donné une relation R(U) et X un ensemble de parties de U

$$X = \{X_1, X_2, \ldots, X_n\} \text{ tel que } \bigcup_{i=1,n} X_i = U$$

R est décomposable en n parties, s'il existe $R_1, \ldots, R_n$ telles que :

(1) $R_i = R[X_i]$, $\forall i$, $i \in [1,n]$

(2) $R = \underset{i=1,n}{*} R_i$

- Les dépendances entre données établissent des conditions de décomposition :

1.- Si R(X,Y,Z) est une relation où $X \rightarrow Y$ est vérifiée alors R est décomposable suivant ({X,Y},{X,Z}) et on a R = R[X,Y]*R[X,Z]

2.- La DM $X \twoheadrightarrow Y$ est vérifiée dans R(X,Y,Z) si et seulement si R est décomposable selon ({X,Y},{X,Z}).

NOTE : On trouvera dans [DELO 78] les conditions de décomposition pour le cas des dépendances hiérarchiques.

ANNEXE III

L'ANALYSEUR DE SPÉCIFICATIONS

I - GRAMMAIRE DES SPECIFICATIONS EN ENTREE DE L'ANALYSEUR

Nous donnons ci-dessous la grammaire qui définit la syntaxe des spécifications soumises à l'analyseur de types ANALTYP.

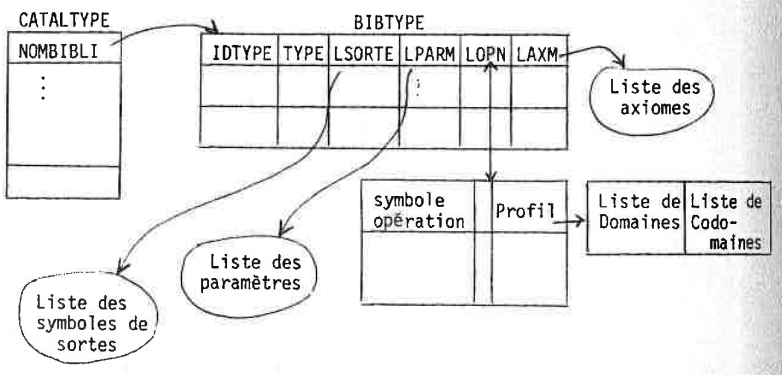
<fichier de specif> → <specif>|<specif><fichier specif>
<specif> → 'TYPE'<en-tête><corps>'FTYPE'
<en-tête> → <idtype >|<idtype param>|<inst-type>
<idtype> → <idattr>|<type predef>
<idtype param> → <idtype>'('<liste param>')'
<liste param> → <idparam>|<idparam>','<liste param>
<inst-type> → <ident-inst>'='<type instce>
<type instce> → <idtype>|<idtype>'('<listattr>')'
<listattr> → <defattr>|<defattr>','<listattr>
<defattr> → <idattr>|<idattr>':'<idtype>
<idparam> → <nomparam>|<nomparam>':'<typeparam>
<ident-inst> → <identificateur>
<idattr> → <identificateur>
<nomparam> → <identificateur>
<typpredef> → CAR|ENTIER|BOOL|...
<corps> → <sortes><opns><axiomes>|<opns><axiomes>
|<sortes><opns>
<sortes> → 'SORTES'<liste sortes>
<liste sortes> → <symbsorte>|<symbsorte>','<liste sortes>
<opns> → 'OPERATIONS'<listop>
<listop> → <oper>|<oper>' '<listop>
<oper> → <symbopn>':'<listedom>'→'<liste codom>
<liste dom> → <liste codom>
<liste codom> → <idom>|<idom>'*'<liste codom>
<idom> → <idattr>|<type predef>|<ident-inst>|<nomparam>
|<symbsorte>

<symbsorte>	→	<identificateur>
<axiomes>	→	'AXIOMES'<listaxm>
<listaxm>	→	<axiome>|<axiome><listaxm>
<axiome>	→	<pgche>'='<pdte>
<pgche>	→	CHAÎNE
<pdte>	→	CHAÎNE

II - ARCHITECTURE D'UN ENVIRONNEMENT DE TYPES

ANALTYP produit une spécification-objet correspondant à la spécification source qui lui est soumise. La spécification-objet consiste en un ensemble de tables descriptives. ANALTYP gère également un catalogue des environnements.

Le schéma ci-dessous symbolise les résultats d'une analyse de spécifications.




```
*-----------------------------------------------------*
*    EXEMPLE  DE  FICHIER  DE  SPECIFICATIONS        *
*-----------------------------------------------------*


TYPE SET(ITEM)
  SORTES:BOOL
  OPERATIONS :
    SVIDE :          -> SET
    ISVIDE: SET      -> BOOL
    INSERT: SET*ITEM -> SET
    DELSET: SET*ITEM -> SET
    ISIN   : SET*ITEM -> BOOL

  AXIOMES :
     Soient s : SET ; i, j : ITEM
     ISVIDE(SVIDE) = 'VRAI'
     ISVIDE(INSERT(s,i)) = 'FAUX'
     ISIN(SVIDE,i) = 'FAUX'
     ISIN(INSERT(s,i),j) = SI i = j ALORS 'VRAI'
                           SINON ISIN(s,j)
     DELSET(SVIDE,i) = SVIDE
     DELSET(INSERT(s,i),j) = SI i = j ALORS DELSET(s,j)
                             SINON INSERT(DELSET(s,j),i)
FTYPE

TYPE AGREGAT(ATTR1:T1,ATTR2:T2,ATTRN:TN)
  SORTES : T1,T2,TN

  OPERATIONS :
    FATTR1 : AGREGAT -> T1
    FATTR2 : AGREGAT -> T2
    FATTRN : AGREGAT -> TN
FTYPE

TYPE INT
  SORTES : BOOL
  OPERATIONS :
    INF    : INT*INT -> BOOL
    SUP    : INT*INT -> BOOL
    EGAL   : INT*INT -> BOOL
    ADD    : INT*INT -> INT
    SUB    : INT*INT -> INT
FTYPE

TYPE DATE(J:INT,M:INT,A:INT)
  SORTES : INT,BOOL

  OPERATIONS :
    JOUR       : DATE -> J
    MOIS       : DATE -> M
    AN         : DATE -> A
    PLUSTOT    : DATE*DATE -> BOOL
    PLUSTARD   : DATE*DATE -> BOOL
    MEMEDATE   : DATE*DATE -> BOOL
  AXIOMES :
    (* INVARIANTS : J DANS [1,31] ; M DANS [1,12];
                    A DANS [1900,2002] *)
    PLUSTOT(d1,d2) = SI INF(d1.A,d2.A)
                       OUB (INF(d2.M,d1.M) ETB EGAL(d2.A,d1.A))
                       OUB (INF(d2.J,d1.J) ETB EGAL(d2.M,d1.M)
                            ETB EGAL(d2.A,d1.A))
                     ALORS 'VRAI' SINON 'FAUX'
```

```
59:
60:     MEMEDATE(d1,d2) = SI EGAL(d1.J,d2.J) ETB EGAL(d1.M,d2.M)
61:                          ETB EGAL(d1.A,d2.A)
62:                       ALORS 'VRAI' SINON 'FAUX'
63:
64:     PLUSTARD(d1,d2) = SI SUP(d1.A,d2.A)
65:                          OUB (SUP(d2.M,d1.M) ETB EGAL(d2.A,d1.A))
66:                          OUB (SUP(d2.J,d1.J) ETB EGAL(d2.M,d1.M)
67:                               ETB EGAL(d2.A,d1.A))
68:                       ALORS 'VRAI' SINON 'FAUX'
69: FTYPE
70:
71: TYPE HEBERGE = AGREGAT(NOMHEB:STRING,CAPACITE:INT,CATEGORIE:INT)
72:    SORTES : STRING,INT,BOOL,AGREGAT
73:    OPERATIONS :
74:      NOMH : HEBERGE -) NOMHEB
75:      CAPH : HEBERGE -) CAPACITE
76:      CATH : HEBERGE -) CATEGORIE
77:      PLUSGRD : HEBERGE*HEBERGE -) BOOL
78:    AXIOMES :
79:      PLUSGRD(h1,h2) = SI SUP(CAPH(h1),CAPH(h2))
80:                       ALORS 'VRAI' SINON 'FAUX'
81: FTYPE
82:
83: TYPE LOISIRS = AGREGAT(NOML:STRING)
84:    SORTES : AGREGAT,STRING
85: FTYPE
86:
87: TYPE PAYS = AGREGAT(NOMPAYS:STRING,ALTITUDE:INT,POPULATION:INT)
88:    SORTES : AGREGAT,BOOL,STRING,INT
89:    OPERATIONS :
90:      PLUSELEVE : PAYS*PAYS -) BOOL
91:    AXIOMES :
92:      PLUSELEVE(p1,p2) = SI SUP(ALTITUDE(p1),ALTITUDE(p2))
93:                         ALORS 'VRAI' SINON 'FAUX'
94: FTYPE
95:
96: TYPE PERIODE = AGREGAT(DEBUT:DATE,FIN:DATE)
97:
98:    SORTES : AGREGAT,DATE,BOOL
99:
100:    (* INVARIANTS : d : DEBUT ;f : FIN; PLUSTOT(d,f)  *)
101:
102:   OPERATIONS :
103:     DEBPER    : PERIODE -) DEBUT
104:     FINPER    : PERIODE -) FIN
105:     HORSPER   : PERIODE*DATE -) BOOL
106:     DANSPER   : PERIODE*PERIODE -) BOOL
107:
108:   AXIOMES :
109:     HORSPER((d1,d2),d3) = SI (PLUSTOT(d3,d1) OUB PLUSTARD(d3,d2))
110:                          ALORS 'VRAI' SINON 'FAUX'
111:
112:     DANSPER((d1,d2),(d3,d4)) = SI (PLUSTOT(d3,d1) ETB PLUSTOT(d4,d
113:                                ALORS 'VRAI' SINON 'FAUX'
114: FTYPE
115:
116: TYPE SAISON = AGREGAT(NOMS:STRING,DEBSAISON:DATE,FINSAISON:DATE)
117:    SORTES : STRING,AGREGAT,DATE
118:
119:    (* INVARIANT : NOMS DANS éHTE-SAISON,MOYENNE-SAISON,BASSE-SAISON
120:
121: FTYPE
122:
123: TYPE CLIENT = AGREGAT(NOCLI:INT,NOMCLI:STRING,TEL:INT)
124:    SORTES : AGREGAT,STRING,INT
125: FTYPE
```



```
)
18:ANALTYP

        *-------------------------------------------------------------*
        I        ANALYSE  ET  MISE  EN  BIBLIOTHEQUE                I
        I    DE  SPECIFICATIONS  DE TYPES  ABSTRAITS  DE             I
        I                        DONNEES                            I
        *-------------------------------------------------------------*


- ENTREE DES SPECIFICATIONS A PARTIR DE :
      - LA CONSOLE     (ENTRER "C")
  ou  - D'UN FICHIER DISQUE ("D") ?
    ** (Pour arreter entrer "FIN") **
       ---

- Quel ENVIRONNEMENT de TYPES allons-nous definir : LOIS

- C'est donc un NOUVEL ENVIRONNEMENT ?(O/N) : O

- INITIALISATION  ENVIRONNEMENT

- INITIALISATION DE L'ENVIRONNEMENT : LOIS
  ---------------------------------

- nbr de postes deja occupes : 0


- NOM DU FICHIER CONTENANT LES SPECIFICATIONS ?
      (sous la forme  "NOMFICH.TYPE"): SPLOIS.BIB

- Sur QUELLE UNITE (A/B) se trouve  SPLOIS.BIB : A

- SI SPLOIS.BIB CONTIENT PLUS D'UNE SPECIFICATION, FAUT-IL
    LES ANALYSER TOUTES  OU UNE PARTICULIERE? :
      ( Entrez "TOUTES" ou "UNE" selon le cas)
                 -------      ---
WTES


 *-----------------------------------------*
 I   ANALYSE  DE  TYPES  (OPTION "TOUTES")  I
 *-----------------------------------------*

-  *  FIN DE LA LISTE D'OPERATIONS  *


-  *  FIN  DES  A X I O M E S  *

-  *   FIN  ANALYSE  DU  CORPS D'UNE SPECIF.  *

 ON PASSE A L'ANALYSE DU TYPE SUIVANT DANS  SPLOIS.BIB
 S'IL Y EN A ENCORE .

-  ECRITURE  DES  PARAMETRES DE SET
   ----------------------------
      - 1 - ITEM

   ECRITURE  DES  SORTES DE SET
   ---------------------------
      - 1 - BOOL

-  *    FIN TRAITEMENT DES SORTES SUPPORT   *
```

```
T-   * FIN DE LA LISTE D'OPERATIONS *

T-   * FIN  DES  A X I O M E S *

T-   *   FIN  ANALYSE  DU  CORPS D'UNE SPECIF.  *

T-  ON PASSE A L'ANALYSE DU TYPE SUIVANT DANS  SPLOIS.BIB
    S'IL Y EN A ENCORE .

T-     ECRITURE  DES  PARAMETRES DE AGREGAT
       ------------------------------
         - 1 - ATTR1
         - 2 - ATTR2
         - 3 - ATTRN

T-     ECRITURE  DES  SORTES DE AGREGAT
       ------------------------------
         - 1 - T1
         - 2 - T2
         - 3 - TN

T-   *    FIN TRAITEMENT DES SORTES SUPPORT    *

T-   * FIN DE LA LISTE D'OPERATIONS *

T-   *   FIN  ANALYSE  DU  CORPS D'UNE SPECIF.  *

T-  ON PASSE A L'ANALYSE DU TYPE SUIVANT DANS  SPLOIS.BIB
    S'IL Y EN A ENCORE .


T-     ECRITURE  DES  PARAMETRES DE HEBERGE
       ------------------------------
         - 1 - NOMHEB
         - 2 - CAPACITE
         - 3 - CATEGORIE

T-     ECRITURE  DES  SORTES DE HEBERGE
       ------------------------------
         - 1 - STRING
         - 2 - INT
         - 3 - BOOL
         - 4 - AGREGAT

T-   *    FIN TRAITEMENT DES SORTES SUPPORT    *

T-   * FIN DE LA LISTE D'OPERATIONS *

T-   * FIN  DES  A X I O M E S *

T-   *   FIN  ANALYSE  DU  CORPS D'UNE SPECIF.  *

T-  ON PASSE A L'ANALYSE DU TYPE SUIVANT DANS  SPLOIS.BIB
    S'IL Y EN A ENCORE .
```



```
-    ECRITURE  DES  PARAMETRES DE PAYS
     ------------------------------
       - 1 - NOMPAYS
       - 2 - ALTITUDE
       - 3 - POPULATION

-    ECRITURE  DES  SORTES DE PAYS
     ------------------------------
       - 1 - AGREGAT
       - 2 - BOOL
       - 3 - STRING
       - 4 - INT

-   *    FIN TRAITEMENT DES SORTES SUPPORT    *

-   * FIN DE LA LISTE D'OPERATIONS *

-   * FIN  DES  A X I O M E S *

-   *   FIN  ANALYSE  DU  CORPS D'UNE SPECIF.  *

-  ON PASSE A L'ANALYSE DU TYPE SUIVANT DANS  SPLOIS.BIB
   S'IL Y EN A ENCORE .

                    :
                    :


-  ON PASSE A L'ANALYSE DU TYPE SUIVANT DANS  SPLOIS.BIB
   S'IL Y EN A ENCORE .

      *-------------------------------------*
   *  FIN DE L'ANALYSE  ( OPTION "TOUTES")  *
      -------------------------------------

-  FIN DE L'ANALYSE  DES  TYPES  DANS  SPLOIS.BIB.
      ------------------------------------


-  ENTREE DES SPECIFICATIONS A PARTIR DE :
        - LA CONSOLE     (ENTRER "C")
     ou - D'UN FICHIER DISQUE ("D") ?
       ** (Pour arreter entrer "FIN") **
       ------------------------------

FIN

   *-----------------------------------------------------*
   I      F I N  DE  L'A N A L Y S E  DES  T Y P E S    I
   I                                                     I
   I                 *  A U  R E V O I R  *              I
   *-----------------------------------------------------*
```

ANNEXE IV

LA DÉFINITION DE BASES ABSTRAITES DE DONNÉES

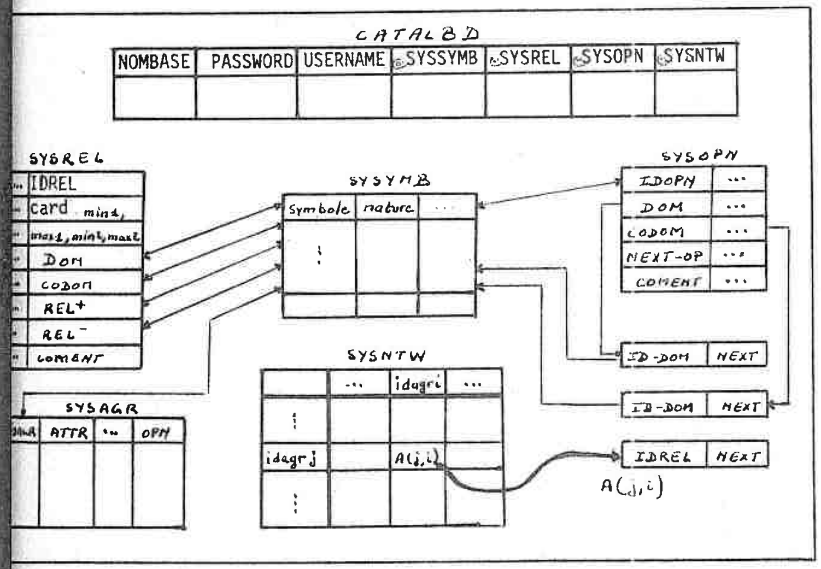
I - ARCHITECTURE DE LA BASE ABSTRAITE

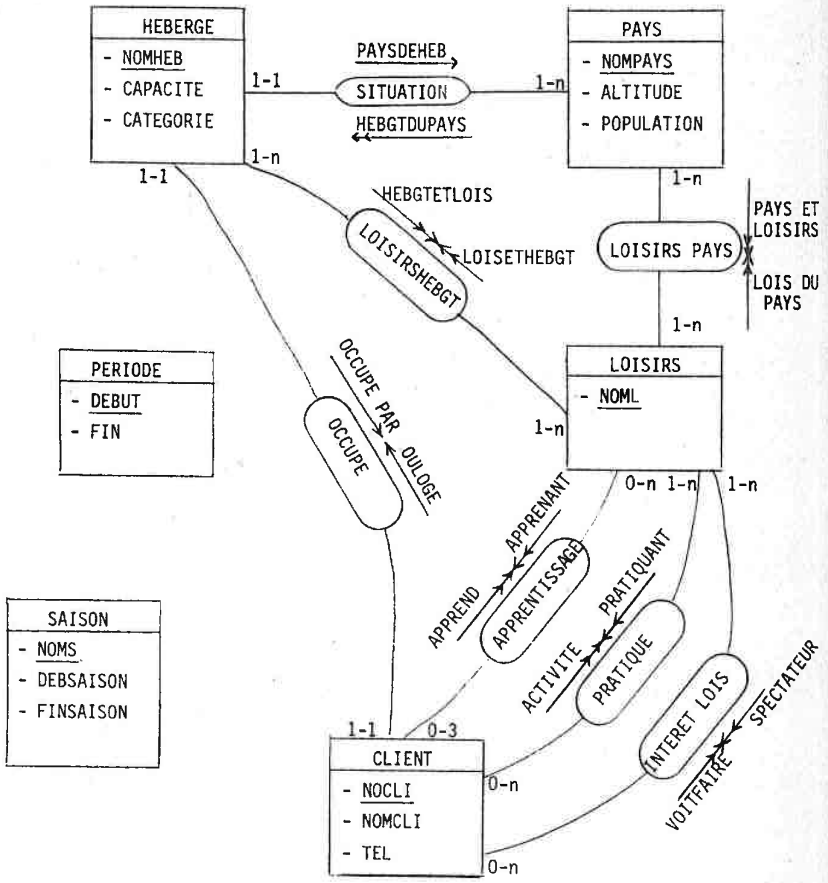
Le module de définition d'une base de données produit une description de la base comportant :

- un dictionnaire des symboles [SYSYMB] d'agrégats, d'attributs, d'opérations, d'associations...
- un descriptif des agrégats [SYSAGR]
- un descriptif des associations [SYSREL]
- un descriptif des opérations (symbole+profil) [SYSOPN]
- un descriptif de l'ensemble du "réseau" d'associations de la base [SYSNTW]

Par ailleurs, à chaque base de données est associée une entrée dans un catalogue des bases existantes [CATALBD].

Le schéma ci-dessous symbolise le résultat d'une définition de base de données par l'intermédiaire de DEFDB.





- Schéma de la base à définir -


```
*-------------------------------------------*
I   DEFINITION D'UNE DATA BASE     I
*-------------------------------------------*

Voulez-vous decliner votre IDENTITE, SVP? :  CMOI
Quel est votre MOT de PASSE ? :      TUTU
Dans quel CATALOGUE voulez-vous ranger votre base ? : CATBD
Entrez un NOM de BASE a definir ou "FIN" :     LOZERE
La base sera definie a partir de quel(s) ENVIRONNEMENT(S) : LOIS
Autre ENVIRONNEMENT (O/N) ? :  N
Initialisation de l'environnement A:LOIS.TYP
Initialisation de la BASE ABSTRAITE LOZERE

Definition :
------------
        - d'Agregats      ( Entrez "A" ),
        - d'Associations  ( Entrez "R" ),
        - d'Operations    ( Entrez "O" ),
        - d'Axiomes       ( Entrez "S" ) ou
      (Si fin de la definition,Entrez "FIN") :  A


*-------------------------------------------*
    DEFINITION  DES  AGREGATS
*-------------------------------------------*


ENTRER UN NOM AGREGAT OU "FIN" : HEBERGE

IMPORTATION DES SORTES DE HEBERGE
      - 1 - STRING
      - 2 - INT
      - 3 - BOOL
      - 4 - AGREGAT
LECTURE DES ATTRIBUTS DE  HEBERGE

VOULEZ-VOUS RESTREINDRE LA LISTE D'ATTRIBUTS ? : N

LEQUEL, parmi ces attributs,designera de facon unique toute
      occurrence de l'agregat?
      - ENTREZ SON NOM ou "AUCUN" : NOMHEB

SUR L'AGREGAT ET SES ATTRIBUTS PEUVENT PORTER DES OPERATIONS;
VOULEZ-VOUS LES GARDER TOUTES OU Y APPORTER DES RESTRICTIONS ?
  - ENTREZ  "G" ( pour Garder ) OU "R" ( pour Restreindre ) : G

LECTURE DES OPERATIONS DE  HEBERGE .
PAS D'OPERATIONS DEFINIES SUR : NOMHEB
PAS D'OPERATIONS DEFINIES SUR : CAPACITE
PAS D'OPERATIONS DEFINIES SUR : CATEGORIE
```

```
D- ENTRER UN NOM AGREGAT OU "FIN" : PAYS

D- IMPORTATION DES SORTES DE PAYS
        - 1 - AGREGAT
        - 2 - BOOL
        - 3 - STRING
        - 4 - INT
D- LECTURE DES ATTRIBUTS DE  PAYS

D- VOULEZ-VOUS RESTREINDRE LA LISTE D'ATTRIBUTS ? : N


D- LEQUEL, parmi ces attributs,designera de facon unique toute
        occurrence de l'agregat?
        - ENTREZ SON NOM ou "AUCUN" : NOMPAYS

D- SUR L'AGREGAT ET SES ATTRIBUTS PEUVENT PORTER DES OPERATIONS;
   VOULEZ-VOUS LES GARDER TOUTES OU Y APPORTER DES RESTRICTIONS ?
     - ENTREZ  "G" ( pour Garder ) OU "R" ( pour Restreindre ) : G

D- LECTURE DES OPERATIONS DE  PAYS
D- PAS D'OPERATIONS DEFINIES SUR : NOMPAYS
D- PAS D'OPERATIONS DEFINIES SUR : ALTITUDE
D- PAS D'OPERATIONS DEFINIES SUR : POPULATION

D- ENTRER UN NOM AGREGAT OU "FIN" : LOISIRS

D- IMPORTATION DES SORTES DE LOISIRS
        - 1 - AGREGAT
        - 2 - STRING
D- LECTURE DES ATTRIBUTS DE  LOISIRS

D- VOULEZ-VOUS RESTREINDRE LA LISTE D'ATTRIBUTS ? : N


D- LEQUEL, parmi ces attributs,designera de facon unique toute
        occurrence de l'agregat?
        - ENTREZ SON NOM ou "AUCUN" : NOML

D- SUR L'AGREGAT ET SES ATTRIBUTS PEUVENT PORTER DES OPERATIONS;
   VOULEZ-VOUS LES GARDER TOUTES OU Y APPORTER DES RESTRICTIONS ?
     - ENTREZ  "G" ( pour Garder ) OU "R" ( pour Restreindre ) : G

D- PAS D'OPERATIONS DEFINIES SUR : LOISIRS
D- PAS D'OPERATIONS DEFINIES SUR : NOML

D- ENTRER UN NOM AGREGAT OU "FIN" : CLIENT

D- IMPORTATION DES SORTES DE CLIENT
        - 1 - AGREGAT
        - 2 - STRING
        - 3 - INT
D- LECTURE DES ATTRIBUTS DE  CLIENT

D- VOULEZ-VOUS RESTREINDRE LA LISTE D'ATTRIBUTS ? : N


D- LEQUEL, parmi ces attributs,designera de facon unique toute
        occurrence de l'agregat?
        - ENTREZ SON NOM ou "AUCUN" : NOCLI

D- SUR L'AGREGAT ET SES ATTRIBUTS PEUVENT PORTER DES OPERATIONS;
   VOULEZ-VOUS LES GARDER TOUTES OU Y APPORTER DES RESTRICTIONS ?
     - ENTREZ  "G" ( pour Garder ) OU "R" ( pour Restreindre ) : G

D- PAS D'OPERATIONS DEFINIES SUR : CLIENT
```



```
AS D'OPERATIONS DEFINIES SUR : NOCLI
AS D'OPERATIONS DEFINIES SUR : NOMCLI
AS D'OPERATIONS DEFINIES SUR : TEL

NTRER UN NOM AGREGAT OU "FIN" : FIN


 FIN DE LA DEFINITION DES AGREGATS

efinition :
-----------
      - d'Asregats      ( Entrez "A" ),
      - d'Associations  ( Entrez "R" ),
      - d'Operations    ( Entrez "O" ),
      - d'Axiomes       ( Entrez "S" ) ou
    (Si fin de la definition,Entrez "FIN") :  R



-------------------------------------*
  DEFINITION  DES  ASSOCIATIONS
-------------------------------------*


NTRER UN NOM ASSOCIATION OU "FIN" : SITUATION

ELS AGREGATS LIE SITUATION ?
GREGAT1 : HEBRERGE
GREGAT2 : PAYS


omment appellerez-vous l'application qui a
HEBERGE fait correspondre PAYS ? : PAYSDEHEB
t son inverse ? :HEBGTDUPAYS

tout(e) HEBERGE peut etre en relation par PAYSDEHEB avec ,
 au MINIMUM combien de PAYS
  - ENTREZ 0 ou le nombre exact ou 99 : 1

Et au MAXIMUM : 1

l a l'inverse,  tout(e) PAYS peut etre en relation par HEBGTDUPAYS av
 au MINIMUM combien de HEBERGE
  - ENTREZ 0 ou le nombre exact ou 99 : 1

Et au MAXIMUM : 99


NTRER UN NOM ASSOCIATION OU "FIN" : LOISIRSPAYS

ELS AGREGATS LIE LOISIRSPAYS ?
GREGAT1 : PAYS
GREGAT2 : LOISIRS


omment appellerez-vous l'application qui a
PAYS fait correspondre LOISIRS ? : PAYSETLOISIRS
t son inverse ? :LOISDUPAYS

tout(e) PAYS peut etre en relation par PAYSETLOISIRS avec ,
 au MINIMUM combien de LOISIRS
  - ENTREZ 0 ou le nombre exact ou 99 : 1

Et au MAXIMUM : 99
```

```
D- Et a l'inverse,  tout(e) LOISIRS peut etre en relation par LOISDUPAYS a
     au MINIMUM combien de PAYS
      - ENTREZ 0 ou le nombre exact ou 99 : 1

D-  Et au MAXIMUM : 99


D- ENTRER UN NOM ASSOCIATION OU "FIN" : LOISIRSHEBGT

D- QELS AGREGATS LIE LOISIRSHEBGT ?
D- AGREGAT1 : LOISIRS
D- AGREGAT2 : HEBERGE


D- Comment appellerez-vous l'application qui a
    LOISIRS fait correspondre HEBERGE ? : LOISETHEBGT
D- et son inverse ? :HEBGTETLOISIRS

D-  tout(e) LOISIRS peut etre en relation par LOISETHEBGT avec ,
     au MINIMUM combien de HEBERGE
      - ENTREZ 0 ou le nombre exact ou 99 : 1

D-  Et au MAXIMUM : 99

D- Et a l'inverse,  tout(e) HEBERGE peut etre en relation par HEBGTETLOISI
c ,
     au MINIMUM combien de LOISIRS
      - ENTREZ 0 ou le nombre exact ou 99 : 1

D-  Et au MAXIMUM : 99


D- ENTRER UN NOM ASSOCIATION OU "FIN" : APPRENTISSAGE

D- QELS AGREGATS LIE APPRENTISSAGE ?
D- AGREGAT1 : CLIENT
D- AGREGAT2 : LOISIRS


D- Comment appellerez-vous l'application qui a
    CLIENT fait correspondre LOISIRS ? : APPREND
D- et son inverse ? :APPRENANT

D-  tout(e) CLIENT peut etre en relation par APPREND avec ,
     au MINIMUM combien de LOISIRS
      - ENTREZ 0 ou le nombre exact ou 99 : 0

D-  Et au MAXIMUM : 99

D- Et a l'inverse,  tout(e) LOISIRS peut être en relation par APPRENANT av
     au MINIMUM combien de CLIENT
      - ENTREZ 0 ou le nombre exact ou 99 : 1

D-  Et au MAXIMUM : 99


D- ENTRER UN NOM ASSOCIATION OU "FIN" : PRATIQUE

D- QELS AGREGATS LIE PRATIQUE ?
D- AGREGAT1 : CLIENT
D- AGREGAT2 : LOISIRS


D- Comment appellerez-vous l'application qui a
    CLIENT fait correspondre LOISIRS ? : ACTIVITE
```



```
 son inverse ? :PRATICANT

tout(e) CLIENT peut etre en relation par ACTIVITE avec ,
 au MINIMUM combien de LOISIRS
  - ENTREZ 0 ou le nombre exact ou 99 : 1

Et au MAXIMUM : 99

 a l'inverse,  tout(e) LOISIRS peut etre en relation par PRATICANT a
 au MINIMUM combien de CLIENT
  - ENTREZ 0 ou le nombre exact ou 99 : 1

Et au MAXIMUM : 99


NTRER UN NOM ASSOCIATION OU "FIN" : INTERETLOIS

ELS AGREGATS LIE INTERETLOIS ?
GREGAT1 : CLIENT
GREGAT2 : LOISIRS


mment appellerez-vous l'application qui a
CLIENT fait correspondre LOISIRS ? : VOITFAIRE
 son inverse ? :SPECTATEUR

tout(e) CLIENT peut etre en relation par VOITFAIRE avec ,
 au MINIMUM combien de LOISIRS
  - ENTREZ 0 ou le nombre exact ou 99 : 1

Et au MAXIMUM : 99

 a l'inverse,  tout(e) LOISIRS peut etre en relation par SPECTATEUR
 au MINIMUM combien de CLIENT
  - ENTREZ 0 ou le nombre exact ou 99 : 1

Et au MAXIMUM : 99


NTRER UN NOM ASSOCIATION OU "FIN" : RESERVE

ELS AGREGATS LIE RESERVE ?
GREGAT1 : HEBERGE
GREGAT2 : CLIENT


mment appellerez-vous l'application qui a
HEBERGE fait correspondre CLIENT ? : RESERVEPAR
 son inverse ? :RESACLIENT

tout(e) HEBERGE peut etre en relation par RESERVEPAR avec ,
 au MINIMUM combien de CLIENT
  - ENTREZ 0 ou le nombre exact ou 99 : 1

Et au MAXIMUM : 99

 a l'inverse,  tout(e) CLIENT peut etre en relation par RESACLIENT a
 au MINIMUM combien de HEBERGE
  - ENTREZ 0 ou le nombre exact ou 99 : 1

Et au MAXIMUM : 1


NTRER UN NOM ASSOCIATION OU "FIN" : OCCUPE

ELS AGREGATS LIE OCCUPE ?
```

```
D- AGREGAT1 : HEBERGE
D- AGREGAT2 : CLIENT

D- Comment appellerez-vous l'application qui a
    HEBERGE fait correspondre CLIENT ? : OCCUPEPAR
D- et son inverse ? :OULOGE

D-  tout(e) HEBERGE peut etre en relation par OCCUPEPAR avec ,
     au MINIMUM combien de CLIENT
      - ENTREZ 0 ou le nombre exact ou 99 : 1

D-  Et au MAXIMUM : 1

D- Et a l'inverse,  tout(e) CLIENT peut etre en relation par OULOGE avec
     au MINIMUM combien de HEBERGE
      - ENTREZ 0 ou le nombre exact ou 99 : 1

D-  Et au MAXIMUM : 1

D- ENTRER UN NOM ASSOCIATION OU "FIN" : FIN

D-  FIN DE LA DEFINITION DES ASSOCIATIONS

D- Definition :

        - d'Agregats      ( Entrez "A" ),
        - d'Associations  ( Entrez "R" ),
        - d'Operations    ( Entrez "O" ),
        - d'Axiomes       ( Entrez "S" ) ou
      (Si fin de la definition,Entrez "FIN") :  0
```

```
*-----------------------------------------*
     DEFINITION  DES  OPERATIONS
*-----------------------------------------*
```

```
D- ENTRER UN NOM OPERATION OU "FIN" : LIBRE

D-       *   DEFINITION  DU  DOMAINE     *

D- ENTREZ UN SYMBOLE DU PROFIL OU "FIN" : LOZERE

D- AUTRE SYMBOLE DU PROFIL OU "FIN" : DATE

D? CE SYMBOLE DU PROFIL N'EST PAS CONNU.

D- AUTRE SYMBOLE DU PROFIL OU "FIN" : HEBERGE

D- AUTRE SYMBOLE DU PROFIL OU "FIN" : FIN

D-       *   DEFINITION  DU  CODOMAINE   *

D- ENTREZ UN SYMBOLE DU PROFIL OU "FIN" : BOOL
```



```
D- AUTRE SYMBOLE DU PROFIL OU "FIN" : FIN

D- ENTRER UN NOM OPERATION OU "FIN" : FIN

D-  FIN DE LA DEFINITION DES OPERATIONS

D- Definition :

        - d'Agregats      ( Entrez "A" ),
        - d'Associations  ( Entrez "R" ),
        - d'Operations    ( Entrez "O" ),
        - d'Axiomes       ( Entrez "S" ) ou
      (Si fin de la definition,Entrez "FIN") :  FIN

*-----------------------------------------*
I  ECRITURE DU CATALOGUE DES  B D D  I
*-----------------------------------------*
      - NBR DE POSTES OCCUPES :  1

D- FIN DE LA DEFINITION DE LA BASE  LOTEL

D- Entrez un NOM de BASE a definir ou "FIN" :    FIN

     *-----------------------------------------*
     I   F I N   DE  LA   D E F I N I T I O N   I
     I     DES  B A S E S   DE  D O N E E S     I
     *-----------------------------------------*
```

ANNEXE I

LA COMPRÉHENSION DE REQUÊTES

I - GRAMMAIRE DU LANGAGE DE REQUETES

```
<requête>      → <cmde><liste cible>':'<liste select>'.'
<cmde>         → 'SORTIR'
<liste cible>  →<descible>|<descible>','<liste cible>
<descible>     → <idtup>|<id-f-accès>'('<argcible>')'
<idtup>        → <idattr>|<idagr>
<idattr>       → %identificateur d'attribut%
<idagr>        → %identificateur d'agrégat%
<id-f-acces>   → %identificateur de fonction d'accès%
<liste select> → <bribe>|<bribe>','<liste select>
<bribe>        → <idfonct>'('<liste arg>')'|<id-0-aire>
<liste arg>    → <arg>|<arg>','<liste arg>
<arg>          → <idtup>|<idtup>"<littéral>"|"<littéral>"|<bribe>
<littéral>     → %constante≡chaîne de caractères%
<id-0-aire>    → %symbole de fonction sans argument%|<idtup>
                 |<idtop>"<littéral>"
<idfonct>      → <id-f-accès>|<id-rel-sem>|<id-f-arg>|<id-f-dérivée>
<id-rel-sem>   → %symbole d'association ou symbole d'une des deux
                  fonctions associées%
<id-f-agr>     → %symbole d'opération sur un agrégat%
<id-f-dérivée > →%symbole d'opération définie niveau base%
<argcible >     <idtup>|<idtup>"<littéral>"|"<littéral>"
```

```
A)B:SINDBAD

*-----------------------------------------------------------------*
I                                                                 I
I                        S I N D B A D                            I
I                                                                 I
I          S Y S T E M E   I N T E R A C T I F   DE               I
I                                                                 I
I  D E D U C T I O N  dans des B A S E S  A B S T R A I T E S     I
I                                                                 I
I                        DE  D O N N E E S                        I
I                                                                 I
*-----------------------------------------------------------------*

*--* S I N D B A D  VOUS OFFRE SES SERVICES :
   *-------------------------------------------------*
           I  - S : AFFICHAGE DES SERVICES ,    I
           I  - F : FIN DE LA SESSION,          I
           I  - I : INTERROGATION D'UNE BASE ,  I
           I  - M : MODIFICATION ,              I
           I  - D : DEFINITION D'UNE BASE .     I
           *-------------------------------------*
   - ENTREZ VOTRE DEMANDE : I

Q- QUEL CATALOGUE DE B.D.D. VOULEZ-VOUS UTILISER : CATBD

Q- QUELLE BASE VOULEZ-VOUS INTERROGER ? : LOZERE

Q- Base LOZERE prete.

Q- ENTREZ VOTRE REQUETE ou "FIN" :

SORTIR HEBERGE:PAYS "MARGERIDE".

      I    ANALYSE  DE  LA  REQUETE   I

- Ma premiere interpretation de votre demande est :

    SORTIR  X01     TEL QUE :
     - 01 - X01 = NOMHEB(X02)
     - 02 - X02 = HEBERGE
     - 03 - X03 = NOMPAYS(X04)
     - 04 - X04 = PAYS

    AVEC :

    NOMPAYS = MARGERIDE
```



```
I    COMPLETION DE LA REQUETE    I

EBERGE ET PAYS sont lies par le fait que :

 - 1 - HEBERGE . SITUATION . PAYS

T-A-T-IL DES INCOHERENCES DANS MON INTERPRETATION (O/N) ? : N

OUS EN SOMMES A LA SUITE DE PROPOSITIONS :

  - 1 -  HEBERGE . SITUATION . PAYS

n interpretation finale de votre demande est :

SORTIR  X01     TEL QUE :
 - 01 - X01 = NOMHEB(X02)
 - 02 - X02 = HEBERGE
 - 03 - X03 = NOMPAYS(X04)
 - 04 - X04 = PAYS

AVEC :

 NOMPAYS = MARGERIDE

  ET :

      1)  SITUATION(HEBERGE,PAYS)


Voulez-vous TRADUIRE la requete dans un langage particulier ? (O/N) :

Dans le langage de quel SYSTEME CIBLE ? : PRLOG
Le TRADUCTEUR en PRLOG n'est pas encore disponible !
Autre SYSTEME CIBLE (O/N) ? : O

Dans le langage de quel SYSTEME CIBLE ? : PROLOG

   I    TRADUCTION DE LA REQUETE EN PROLOG    I

*-----------------------------------------------------------------*
REMARQUE IMPORTANTE: I                                            *
*-------------------*CE TRADUCTEUR de requete suppose que        ù
les choix de representation suivants ont ete faits :              ù
                                                                  ù
 rep(AGREGAT(C1,...,Cn)) = *AGREAGT(*C1,...,*Cn)                  ù
 rep(ASSOCIATION(AGR1,AGR2)) = *ASSOCIATION(*CLE1,*CLE2)          ù
   ou CLE1 est la cle de l'agresat1 (AGR1) et                     ù
      CLE2 celle de l'agresat2 (AGR2)                             ù
*-----------------------------------------------------------------*

TE(*X01)-HEBERGE(*X01,*01,*02)-PAYS(MARGERIDE,*03,*04)
              -SITUATION(*X01,MARGERIDE)-SOR(*X01)
```

```
Q- ENTREZ VOTRE REQUETE ou "FIN" :

SORTIR PAYS:HEBERGE "LE NID".

        ------------------------------------
        I    ANALYSE  DE  LA  REQUETE   I
        ------------------------------------


- Ma premiere interpretation de votre demande est :
  ------------------------------------------------

     SORTIR  X01    TEL QUE :
      - 01 - X01 = NOMPAYS(X02)
      - 02 - X02 = PAYS
      - 03 - X03 = NOMHEB(X04)
      - 04 - X04 = HEBERGE

     AVEC :

      NOMHEB = LE NID


        ------------------------------------
        I    COMPLETION DE LA REQUETE    I
        ------------------------------------


C- PAYS ET HEBERGE sont lies par le fait que :

      - 1 - PAYS . SITUATION . HEBERGE

C- Y-A-T-IL DES INCOHERENCES DANS MON INTERPRETATION (O/N) ? : O

C- VOULEZ-VOUS M'INDIQUER SUR QUELLE(S) LIGNE(S) ? : 1
C- AUTRE NUMERO DE LIGNE ou 99 (pour FIN) : 99
C- SAVEZ-VOUS QUELLE RELATION LIE PAYS ET HEBERGE (O/N) ? : N

C- JE VAIS ESSAYER AUTRE CHOSE, SI POSSIBLE.


C? IL N'Y A PAS D'AUTRE RELATION ENTRE PAYS
     ET  HEBERGE .

C- PAYS ET HEBERGE sont lies par le fait que :

      - 1 - PAYS . LOISIRSPAYS . LOISIRS
      - 2 - LOISIRS . LOISIRSHEBGT . HEBERGE

C- Y-A-T-IL DES INCOHERENCES DANS MON INTERPRETATION (O/N) ? : N


C- NOUS EN SOMMES A LA SUITE DE PROPOSITIONS :

      - 1 -  PAYS . LOISIRSPAYS . LOISIRS
      - 2 -  LOISIRS . LOISIRSHEBGT . HEBERGE
```



```
Mon interpretation finale de votre demande est :
------------------------------------------------

 SORTIR  X01    TEL QUE :
  - 01 - X01 = NOMPAYS(X02)
  - 02 - X02 = PAYS
  - 03 - X03 = NOMHEB(X04)
  - 04 - X04 = HEBERGE

 AVEC :

   NOMHEB = LE NID

   ET :

       1)  LOISIRSPAYS(PAYS,LOISIRS)
       2)  LOISIRSHEBGT(LOISIRS,HEBERGE)


-Voulez-vous TRADUIRE la requete dans un langage particulier ? (O/N)

- Dans le langage de quel SYSTEME CIBLE ? : PROLOG

    ------------------------------------------------
    I    TRADUCTION DE LA REQUETE EN PROLOG     I
    ------------------------------------------------


----------------------*
REMARQUE IMPORTANTE: I-------------------------------------------------*
----------------------*CE TRADUCTEUR de requete suppose que            ù
les choix de representation suivants ont ete faits :                   ù
                                                                       ù
-  rep(AGREGAT(C1,...,Cn)) = +AGREAGT(*C1,...,*Cn)                     ù
-  rep(ASSOCIATION(AGR1,AGR2)) = +ASSOCIATION(*CLE1,*CLE2)             ù
     ou CLE1 est la cle de l'agregat1 (AGR1) et                        ù
        CLE2 celle de l'agregat2 (AGR2)                                ù
-----------------------------------------------------------------------*


#QTE(*X01)-PAYS(*X01,*01,*02)-HEBERGE(LE NID,*03,*04)
                -LOISIRSPAYS(*X01,*X10)-LOISIRSHEBGT(*X10,LE NID)
                -SOR(*X01)
```

```
Q- ENTREZ VOTRE REQUETE ou "FIN" :

SORTIR HEBERGE.:PAYS.

      ------------------------------------
      I    ANALYSE  DE  LA  REQUETE    I
      ------------------------------------


- Ma premiere interpretation de votre demande est :
  ------------------------------------------------

    SORTIR  X01    TEL QUE :
     - 01 - X01 = NOMHEB(X02)
     - 02 - X02 = HEBERGE
     - 03 - X03 = NOMPAYS(X04)
     - 04 - X04 = PAYS


      ------------------------------------
      I    COMPLETION DE LA REQUETE     I
      ------------------------------------


C- HEBERGE ET PAYS sont lies par le fait que :

     - 1 - HEBERGE . SITUATION . PAYS

C- Y-A-T-IL DES INCOHERENCES DANS MON INTERPRETATION (O/N) ? : O

C- VOULEZ-VOUS M'INDIQUER SUR QUELLE(S) LIGNE(S) ? : 1
C- AUTRE NUMERO DE LIGNE ou 99 (pour FIN) : 99
C- SAVEZ-VOUS QUELLE RELATION LIE HEBERGE ET PAYS (O/N) ? : N

C- JE VAIS ESSAYER AUTRE CHOSE, SI POSSIBLE.


C? IL N'Y A PAS D'AUTRE RELATION ENTRE HEBERGE
     ET  PAYS .

C- HEBERGE ET PAYS sont lies par le fait que :

     - 1 - HEBERGE . RESERVE . CLIENT
     - 2 - CLIENT . APPRENTISSAGE . LOISIRS
     - 3 - LOISIRS . LOISIRSPAYS . PAYS

C- Y-A-T-IL DES INCOHERENCES DANS MON INTERPRETATION (O/N) ? : O

C- VOULEZ-VOUS M'INDIQUER SUR QUELLE(S) LIGNE(S) ? : 1
C- AUTRE NUMERO DE LIGNE ou 99 (pour FIN) : 99
C- SAVEZ-VOUS QUELLE RELATION LIE HEBERGE ET CLIENT (O/N) ? : N

C- JE VAIS ESSAYER AUTRE CHOSE, SI POSSIBLE.

C- LA RELATION LIANT HEBERGE ET CLIENT
 NE SERAIT-ELLE PAS : OCCUPE (O/N) ? : O

C- NOUS EN SOMMES A LA SUITE DE PROPOSITIONS :

     - 1 -  HEBERGE . OCCUPE . CLIENT
     - 2 -  CLIENT . APPRENTISSAGE . LOISIRS
     - 3 -  LOISIRS . LOISIRSPAYS . PAYS
```



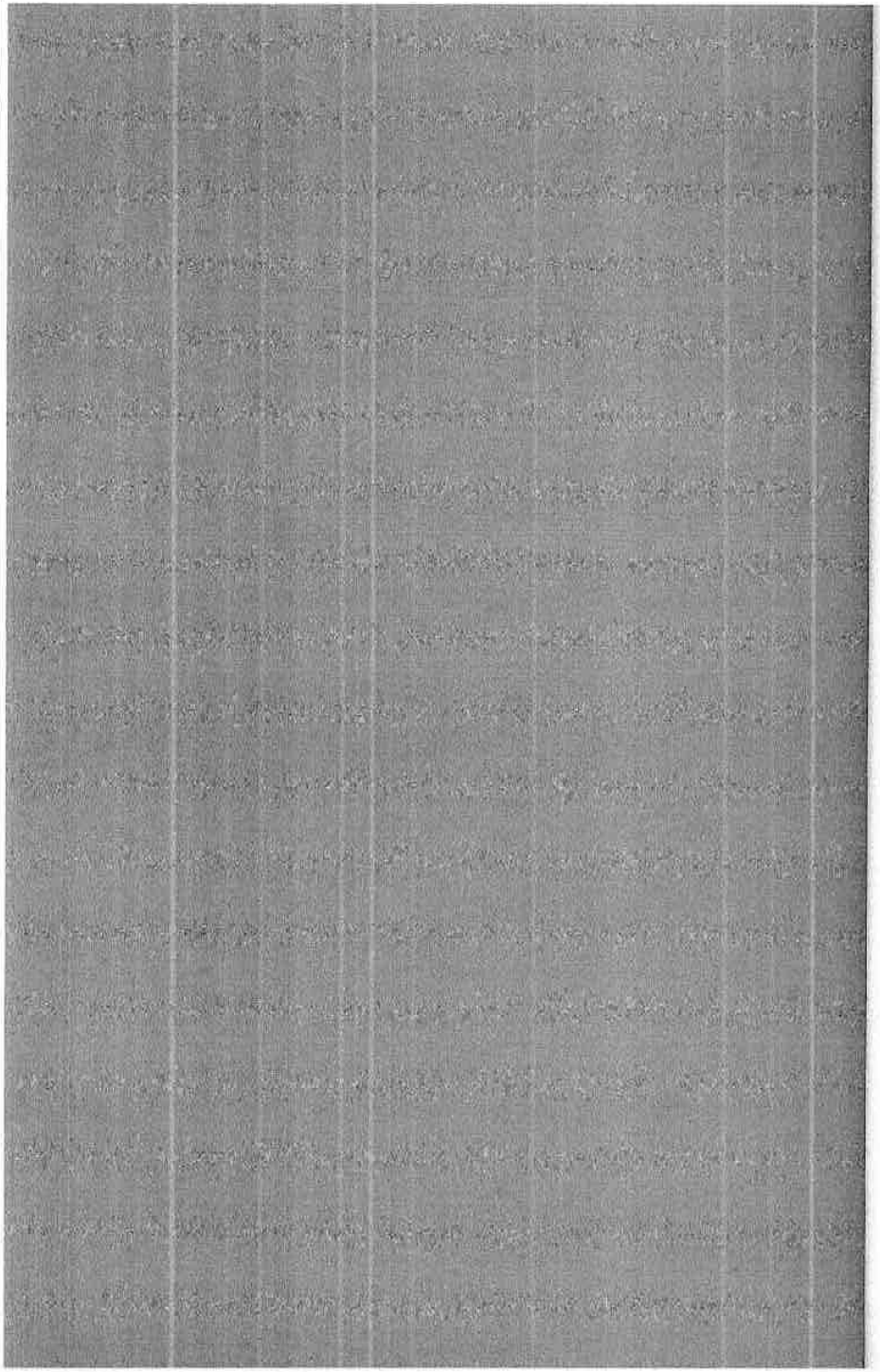
```
n interpretation finale de votre demande est :
  --------------------------------------------

 SORTIR  X01    TEL QUE :
  - 01 - X01 = NOMHEB(X02)
  - 02 - X02 = HEBERGE
  - 03 - X03 = NOMPAYS(X04)
  - 04 - X04 = PAYS

   ET :

        1)  OCCUPE(HEBERGE,CLIENT)
        2)  APPRENTISSAGE(CLIENT,LOISIRS)
        3)  LOISIRSPAYS(LOISIRS,PAYS)


Voulez-vous TRADUIRE la requete dans un langage particulier ? (O/N) :

Dans le langage de quel SYSTEME CIBLE ? : PROLOG

    ------------------------------------------------
    I    TRADUCTION DE LA REQUETE EN PROLOG     I
    ------------------------------------------------


                        *
EMARQUE IMPORTANTE: I-------------------------------------------------*
                    *CE TRADUCTEUR de requete suppose que ù
les choix de representation suivants ont ete faits :      ù
                                                          ù
 rep(AGREGAT(C1,...,Cn)) = +AGREAGT(*C1,...,*Cn)          ù
 rep(ASSOCIATION(AGR1,AGR2)) = +ASSOCIATION(*CLE1,*CLE2)  ù
   ou CLE1 est la cle de l'agregat1 (AGR1) et             ù
      CLE2 celle de l'agregat2 (AGR2)                     ù
-------------------------------------------------------------*


0TE(*X01)-HEBERGE(*X01,*01,*02)-PAYS(*X03,*03,*04)
                -OCCUPE(*X01,*X10)-APPRENTISSAGE(*X10,*X09)
                -LOISIRSPAYS(*X03,*X09)-SOR(*X01)
```

```
Q- ENTREZ VOTRE REQUETE ou "FIN" :

SORTIR HEBERGE,PAYS:ALTITUDE "1200",LOISIRS "SKI DE FOND",CAPACITE "4".

      ------------------------------------
      I    ANALYSE  DE  LA  REQUETE    I
      ------------------------------------

- Ma premiere interpretation de votre demande est :
  ------------------------------------------------

    SORTIR  X01, X03     TEL QUE :
     - 01 - X01 = NOMHEB(X02)
     - 02 - X03 = NOMPAYS(X04)
     - 03 - X04 = PAYS
     - 04 - X02 = HEBERGE
     - 05 - X05 = ALTITUDE(X03)
     - 06 - X06 = NOML(X07)
     - 07 - X08 = CAPACITE(X01)
     - 08 - X07 = LOISIRS

     AVEC :

    ALTITUDE = 1200
        NOML = SKI DE FOND
    CAPACITE = 4


      ------------------------------------
      I    COMPLETION DE LA REQUETE    I
      ------------------------------------

C- HEBERGE ET PAYS sont lies par le fait que :

     - 1 - HEBERGE . SITUATION . PAYS

C- Y-A-T-IL DES INCOHERENCES DANS MON INTERPRETATION (O/N) ? : N

C- NOUS EN SOMMES A LA SUITE DE PROPOSITIONS :

      - 1 -  HEBERGE . SITUATION . PAYS

C- HEBERGE ET LOISIRS sont lies par le fait que :

     - 1 - HEBERGE . LOISIRSHEBGT . LOISIRS

C- Y-A-T-IL DES INCOHERENCES DANS MON INTERPRETATION (O/N) ? : N

C- NOUS EN SOMMES A LA SUITE DE PROPOSITIONS :

      - 1 -  HEBERGE . SITUATION . PAYS
      - 2 -  HEBERGE . LOISIRSHEBGT . LOISIRS
```



```
a interpretation finale de votre demande est :
  --------------------------------------------

SORTIR  X01, X03     TEL QUE :
 - 01 - X01 = NOMHEB(X02)
 - 02 - X03 = NOMPAYS(X04)
 - 03 - X04 = PAYS
 - 04 - X02 = HEBERGE
 - 05 - X05 = ALTITUDE(X03)
 - 06 - X06 = NOML(X07)
 - 07 - X08 = CAPACITE(X01)
 - 08 - X07 = LOISIRS

AVEC :

ALTITUDE = 1200
    NOML = SKI DE FOND
CAPACITE = 4

  ET :

      1)  SITUATION(HEBERGE,PAYS)
      2)  LOISIRSHEBGT(HEBERGE,LOISIRS)


Voulez-vous TRADUIRE la requete dans un langage particulier ? (O/N) :

Dans le langage de quel SYSTEME CIBLE ? : PROLOG

      ------------------------------------
      I    TRADUCTION DE LA REQUETE EN PROLOG    I
      ------------------------------------


-------------------*
EMARQUE IMPORTANTE: I-------------------------------------------------*
-------------------*CE TRADUCTEUR de requete suppose que à
es choix de representation suivants ont ete faits :        à
                                                           à
rep(AGREGAT(C1,...,Cn)) = +AGREAGT(*C1,...,*Cn)            à
rep(ASSOCIATION(AGR1,AGR2)) = +ASSOCIATION(*CLE1,*CLE2)    à
  ou CLE1 est la cle de l'agresat1 (AGR1) et               à
     CLE2 celle de l'agresat2 (AGR2)                       à
-----------------------------------------------------------*


TE(*X01,*X03)-HEBERGE(*X01,4,*01)-PAYS(*X03,1200,*02)
              -LOISIRS(SKI DE FOND)-SITUATION(*X01,*X03)
              -LOISIRSHEBGT(SKI DE FOND,*X01)-SOR(*X01,*X03)

ENTREZ VOTRE REQUETE ou "FIN" :

N

AUTRE COMMANDE ou "F(in)" ou "S(ervices)" : F

   *----    S I N B A D   A  ETE TRES  HEUREUX  DE VOUS  SERVIR   ----*
                ET ESPERE  VOUS  REVOIR  TRES  BIENTOT
                          SUR  SES  LIONES
                ------------------------------------
```

BIBLIOGRAPHIE



[ABRI 74] J.R. ABRIAL : Data Semantics
in. D.B. Management - Klimbie & Koffeman Eds
North Holland Pub. Comp. 1974 [1-59].

[ACM 80] ACM 80 : Proceedings of the workshop on data abstraction...

[ANSI 75] ANSI/X3/SPARC : Interim report
ACM Vol. 7 N° 2 (1975)

[ADJ 78] J.A. GOGUEN - J.W. THATCHER - E.G. WAGNER :
An initial algebra approach to the specification, correctness and implementation of abstract data types. in Current Trends in programming methodology (chap. 5 - vol. IV) - R.T.Yeh - Editor - Prentice Hall.

[ASTR 76] M.M. ASTRAHAN & al : a relational approach to database management. ACM TODS - Vol. 1 N° 2 (juin 76).

[AUTR 82] J. AUTRAN, M. FLORENZANO : Divers modèles de données utilisés dans les SGBD.
GAMSAU - Rapports internes 03/81 et 02/82.

[BARB 82] G. BARBEYE, T. JOUBERT, M. MARTIN : OASIS : un outil d'aide à la spécification interactive et structurée.
Journées BIGRE - GRENOBLE - Janvier 1982.

[BACK 78] J. BACKUS : Can Programming be liberated from the VON NEUMANN style ? A functional style and its algebra of programs. CACM - Vol 21 - N° 8 - Août 1978
[pp. 613-641].

[BART 81] BARTELS U, W. OLTHOFF, RAULEFS : APE : an expert system for automatic programming from specifications of data types and algorithms - IJCAI 1981 [pp. 1037-1043].

[BAZ 83] P. BAZEX, M. ZANNANE, J.M. CORIS, A. MARCOUX, C. PERCEBOIS : QUERYLOG, un système de bases de données qui utilise la déduction.

[BENA 78] H. BENALI - M. HAMITI : Conception d'un système d'information de la scolarité du CERI. Mémoire Ingénieur - CERI - ALGER (1978).

[BERN 76] P.A. BERNSTEIN : Synthesizing third normal form relations from functional dependencies. ACM. TODS - Vol 1 N° 4, Dec 76 [pp. 277-298].

[BOBR 77] D.G. BOBROW & al : GUS, a frame driven-dialog system Artif. Intelligence 8 (1977) [pp. 155-173].

[BOUD 75] N. BOUDJLIDA : Utilisation d'une base de données à partir d'un langage évolué : réalisation d'un interface MIISFIIT-PL/1. CETE - AIX-EN-PROVENCE (1975).

[BOUD 82] N. BOUDJLIDA : Méthode et outils pour la conception et l'exploitation de la base de connaissances "MONT-LOZERE". Rapport d'avancement du projet MONT-LOZERE SYSECA-CRIN (juil. 82).

[BOUS 74] E. DE BOUSSINEAU : Le superviseur MIISFIIT sous OS.360. Thèse de 3ème cycle - Univ. de PARIS VI (1974).

[BORK 78] S.A. BORKIN : Data Model Equivalence MIT - Computation Structures Group Memo 158 (Février 1978).

[BRAC 77] BRACHMAN R.J. : what's in a concept : structural foundations for semantic networks. Intern. Journ. of Man Machine Studies 1977-9 [pp. 127-152].

[BUND 79] BUNDY & al : Solving mechanics problems using meta-level inference. IJCAI 1979 [pp. 1017-1027].

[CADI 75] J.M. CADIOU : On semantics issues in the relational model of data. LNCS sep 75. Proceedings of Int. Symp. on Math. Foundations of Comp. Sce (GDANSK-POLOGNE).

[CHAB 79] JJ.CHABRIER & al : Projet TYP - Manuel d'utilisation du langage ATM. CRIN Nancy - Rapport interne 79 - R. 081 (1979).

[CHAB 82] JJ.CHABRIER : Spécification et construction de systèmes orientés bases de données : Techniques et langages basés sur le concept de type abstrait. Thèse d'état - CRIN - Univ. NANCY I (oct. 82).

[CHABJ 82] J. CHABRIER : Présentation et utilisation du langage PROLOG - Rapport CRIN Nancy 82-R-078 (1982).

[CHAN 76] CL. CHANG : DEDUCE : a deductive query language for relational data bases. in AI & Pattern Recognition C.G. CHEN Ed ; Academic Press New York (1976) [pp. 108-134].

[CHAN 78a] SHI-KUO CHANG & WU-HAUNG CHENG : Data base skeleton and its application to logical database synthesis. IEEE. Trans. on Soft. Eng. Vol SE4 N° 1 Janv 1978 [pp. 18-30].

[CHAN 78b] SHI-KUO CHANG & JYH-SHENG KE : Database skeleton and its application to fuzzy query translation. IEEE - Trans. on Soft. Eng - Vol 4 N° 1 Janv 78 [pp. 31-44].

[CHAR 78] E. CHARNIAK : On the use of framed - knowledge in language comprehension. AI - 11 - 1978 [pp 225-265].

[CHEN 76] P.P. CHEN : The entity-relationship model - Toward a unifed view of data. ACM TODS Vol 1 N° 1 Mars 76 [pp. 9-36].

[CHOU 81] E. CHOURAQUI : Contribution à l'étude théorique de la représentation des connaissances : Le système symbolique ARCHES.
Thèse d'état - IN Polyth. de LORRAINE (Oct. 1981).

[CLAN 81] W.J. CLANCEY & R. LESTSINGER : NEOMYCIN : reconfiguring a rule-based expert system for application to teaching. IJCAI - 1981 [pp. 829-836].

[CLAR 79] K.L. CLARK : Predicate logic as a computational formalism. Imperial College of Sce & Technology Univ. of London. Research Monograph 79/59 TOC (Dec. 79).

[CNRS 81] CNRS - UPS - UTM - ADI : Approches formelles de la sémantique naturelle. Ecole d'été de linguistique pour informaticien. TOULOUSE (Août-Sept 81).

[CODD 70] E.F. CODD : A relational model of data for large shared data banks. CACM Vol 13 N° 6 Juin 1970 [pp. 377-387].

[COUL 81] D. COULON : Le langage naturel. Une méthode très simple : les mots-clés. Congrès AFCET - Sept 81 NANCY.

[CRE 75] M. CREHANGE : Description formelle, représentation, interrogation des informations complexes : SYSTEME PIVOINES. Thèse d'état Univ. NANCY I (Déc. 75).

[COLM 82] A. COLMERAUER : PROLOG II : Manuel de référence et modèle théorique. Groupe Int. Artif. Marseille LUMINY (mars 1982).

[DAHL 82] V. DAHL : On database systems through logic.
ACM TODS Vol 7 N° 1 - Mars 82 [pp. 102-123].

[DELO 78] C. DELOBEL : Normalization and hierarchical dependencies in the relational data model.
ACM TODS Vol 3 N° 3 - sept 78 [pp. 201-222].

[DELO 82] C. DELOBEL - A. ADIBA : Bases de données et systèmes relationnels. DUNOD-INFORMATIQUE (1982).

[DERA 82] P. DERANSART : Dérivation de programmes PROLOG à partir de spécifications algébriques.INRIA (mai 82).

[DERFI 79] J.C. DERNIAME, J.P. FINANCE : Types abstraits de données : spécification, utilisation et réalisation.
Ecole d'été de l'AFCET - Monastir (1979).

[DERN 84] J.C. DERNIAME, M.J. KIM : Data type of relation in a high level programming language : LTR V3.
Rapport CRIN NANCY 84-R-003 (1984).

[DERN 81] J.C. DERNIAME, JJ. CHABRIER, C. GODART, N. TOUNSI, A. AIT-HADDOU : Un système de gestion multibase.
CRIN Nancy - Rapport interne 81-R-53 (1981).

[DERM 80] Mc DERMOTT : R1 a rule based configurer systems Dept of Comp. Sce. CARNEGIE-MELLON UNIV. (av.1980).

[DINC 79] M. DINCBAS : PROLOG et un exemple de système expert écrit en PROLOG. Doc. CERT (1979).

[DOSC 82] W. DOSCH, G. MASCARI, M. WIRSING : On the algebraic specification of databases. 8th. conf. on VLDB - Mexico (sept. 1982).

[DUB 84] E. DUBOIS : Cadre et méthode de spécification de systèmes d'informations fondés sur les types de données. Thèse doct. ing. INPL (mars 1984).

[DUFO 80] J.F. DUFOURD : Maquettes pour évaluer les systèmes d'information des organisations. Thèse d'état Univ. NANCY I (Janvier 1980).

[DUFO 81] J.F. DUFOURD : Spécification des systèmes d'information. CRIN Nancy - Rapport interne 81-E-023 (1981).

[DUR 84] J. DURAND : Use a rewriting system in your logic program. Rapport CRIN (1984).

[FAG 79] LM FAGAN & al : Représentation of dynamic clinical knowledge : measurement, interprétation in the intensive care unit. IJCAI 1979.

[FAGI 77] R. FAGIN : Multivalued dependencies and a new normal form for relational databases. ACM TODS - Vol 2 N° 3 - sept 77 [pp. 262-278].

[FAGI 82] R. FAGIN, A. MENDELZON, J. ULLMAN : A simplified universal relation assumption and its properties. ACM TODS - Vol 7 N° 3 - sept 82 [pp. 343-360].

[FIN 79] J.P. FINANCE : Etude de la construction des programmes. Thèse d'état - Univ. NANCY I (oct. 79).

[FLO 82] A. FLORY : Bases de données - Conception et réalisation. Ed. ECONOMICA (1982).

[FOUC 82] O. FOUCAULT : Modèle et outil pour la conception des systèmes d'information dans les organisations : Projet REMORA. Thèse d'état Univ. NANCY I (Juin 1982).

[GALL 78] H. GALLAIRE, J. MINKER, J.M. NICOLAS : An overview and introduction to logic and databases. In Logic & databases - Edited by H. GALLAIRE & J. MINKER (1978).

[GAL 79] H. GALLAIRE, C. LASSERE : Issues in controlling a deduction process in a declarative mode. dans Représentation des connaissances et raisonnement dans les sciences de l'Homme. Colloque St MAXIMIM - INRIA-LISH-sept 79 publié par l'INRIA.

[GAUD 80] M.C. GAUDEL : Génération et preuve de compilateurs basées sur une sémantique formelle des langages de programmation. Thèse d'état I.N. Polyth. de LORRAINE (1980).

[GESC 77] C. GESCHKE & al : Abstraction Early experience whith MESA. CACM - Vol 20, 8 (août 77) [pp 540-553].

[GUT 78a] J.V. GUTTAG, JJ. HORNING : The algebraic specification of abstract data types. Acta informatica - 10 - (1978) [pp. 27-52].

[GUT 78b] J.V. GUTTAG, E. HOROWITZ, D. MUSSER : Abstract data types and software validation. CACM vol 21 N° 12 (déc. 78) [pp. 1048-1064].

[GUT 78c] J.V. GUTTAG, E. HOROWITZ, D. MUSSER : The design of data type specifications. In Current trends in programming methodology (vol. IV. chap. IV). R.T. YEH Ed. - Prentice Hall.

[GRIF 82] R.L. GRIFFITH : Three principles of representation for semantic networks. ACM TODS. vol 8 N° 3 - sept 82 [pp. 417-442].

[HUET 80] G. HUET, D.C. OPPEN : Equations and rewrite rules : a survey in Formal languages. Perspectives and open problems. R. Book. Ed. Academic Press (1980).

[JOUA 83] J.P. JOUANNAUD : Les systèmes de réécriture. Notes de cours - Univ. de NANCY I (DEA 1982/1983).

[KAYS 81] D. KAYSER, D. COULON : Utilisation des techniques de raisonnement à profondeur variable dans un système de réponse aux questions. Journ. SYSTEMES EXPERTS - AVIGNON (Mai 1981).

[KELL 76] C. KELLOG, P. KLAHR, L. TRAVIS : A deductive capability for data management. In systems for large data bases. (Lockemann & Neuhold Eds) North Holland Pub. Comp. 1976 [pp. 181-196].

[KERS 76] L. KERSHBERG, A. KLUG, D. TSICHRITZIS : A taxonomy of data models. in systems for large data bases. (Lockemann & Neuhold Eds). North Holland Pub. Comp. (1976).

[KLEEN 71] S.C. KLEENE : Logique mathématique. Collection U - Armand Colin Ed. (1971).

[KORF 66] R.R. KORFHAGE : Logic and algorithms (with applications to the computer & information sciences). J. Wiley & Sons Inc. (1966).

[KOUL 83] J. KOULOUMDJIAN, G. KOUM : Réalisation d'un logiciel d'interrogation pour non informaticien. Séminaire INFORSID (25-27 mai 83) - Campo dell'oro - CORSE.

[KOWA 76] R.A. KOWALSKI & M.H. VAN EMDEN : The semantic of predicate logic as a programming language JACM - Vol 23,4 (oct 1976) [pp. 733-742].

[KOWA 79a] R.A. KOWALSKI & A. DELIYANNI : Logic and semantic networks. CACM Vol 22,3 mars 79 [pp. 184-192].

[KOWA 79b] R.A. KOWALSKI : Logic for problem solving. Colloque St MAXIMIM - INRIA-LISH (sept 79) - Publié par l'INRIA.

[KOWA 81] R.A. KOWALSKI : PROLOG as a logic programming language. Research report N° DOC 81/26 (juillet 81) Imperial College of LONDON.

[LAUR 82] J.L. LAURIERE : Représentation et utilisation des connaissances 1 - Les systèmes experts ; 2 - Représentation des connaissances. TSI Vol 1, N° 1 et 2 (1982).

[LEAV 81] B. LEAVENWORTH : Data programming with data abstractions. Nat. Comput. Conf. 1981 [pp. 537-542].

[LENA 82] D.B. LENAT : The nature of heuristics. Artif. Intell. 1982 [pp. 189-249].

[LESC 83] P. LESCANNE : An Introduction to REVE. CRIN Nancy (1983).

[LESS 77] V.R. LESSER & L.D. ERMAN : A retrospective view of the HEARSAY II architecture. I.J.C.A.I. 1977 [pp. 790-800].

[LISK 75] B. LISKOV & S.N. ZILLES : Spécification techniques for data abstractions. IEEE - Trans. on Soft. Eng vol SE 1-1 (mars 75) [pp. 7-19].

[LISK 77] B. LISKOV & al : abstraction mechanisms in CLU CACM Vol 20,8 - août 77 [pp. 564-576].

[LISK 81] B. LISKOV & al : CLU - reference manual LNCS 114 - Goos & Hartmanis Eds. Springer Verlag.

[LIV 78] C. LIVERCY : Théorie des programmes : schémas, preuves, sémantique. DUNOD-INFORMATIQUE (1978).

[MELO 76] H. MELONI : PROLOG : Mise en route de l'interpréteur et exercices (contrats SESORI/SFER N° 75-104) (1976).

[MARQ 83] G. MARQUE-PUCHEU, J.M. GALLAUSIAUX, G. JOMIER : Interfacing PROLOG and relational database management systems. Rapport de rech. N° 16 - ISEM-U.PARIS SUD (Sept 83).

[MAIB] TSE MAIBAUM : Data base instances, abstract data types and data base specification. Univ. of WATERLOO - Comp. Sce. Dept - TORONTO.

[MANN 74] Z. MANNA : Mathematical Theory of computation. Mac Graw Hill Eds (1974).

[MUSS 80] D.R. MUSSER : Abstract data types specification in the AFFIRM system. IEEE. Trans. on Soft. Eng. Vol SE6,1 (janv. 80) [pp. 24-32].

[NASS 83] R. NASSIF : Projet ADONIS : conception et réalisation d'un système de gestion de multibase de données. Thèse doct. ing. - INP LORRAINE (sept 83).

[NICO 82a] J.M. NICOLAS : Bases de données déductives : un aperçu. (Document de travail-groupe BD3 juin 82).

[NICO 82b] J.M. NICOLAS, K. YAZDANIAN : Un aperçu du système BDGEN Séminaire Bases de données Nov 1982 - TOULOUSE. Edité par ADI.

[NILS 71] N.J. NILSSON : Problem solving methods in artificial intelligence. Mac Graw Hill Eds. Comp. Series (1971).

[OUAD 78] R. OUADI : Méthode, modèles et outils pour la conception de la base de données d'un système d'informations : le passage au modèle interne. Mémoire d'ingénieur - CETE AIX-EN-PROVENCE & CERI-ALGER (nov 78).

[ONER 82] ONERA-CERT-DERI : Journées d'études sur : bases logiques pour bases de données - TOULOUSE (14-17 déc. 82).

[PAIR 74] C. PAIR : Formalization of the notions of data, information and information structure
in D.B. Management Klimbie & Koffeman Eds
North Holland Pub. Comp. 1974.

[PAIR 80] C. PAIR : Types abstraits et sémantique algébrique des langages de programmation. Rapport CRIN Nancy 80-R-011 (1980).

[PARN 72a] D.L. PARNAS : A technique for software module specification with examples.
CACM - May 72 - Vol 15,5 [pp. 330-336].

[PARN 72b] D.L. PARNAS : On the criteria to be used in decomposing system into modules.
CACM Dec 72 Vol 15,12 [pp. 1053-1058].

[PINS 81] S. PINSON : Représentation des connaissances dans les systèmes experts.
RAIRO Informatique Vol 15,N°4 (1981) [pp. 343-367].

[PROC 82] K. PROCH : ORSEC : un outil de recherche de spécifications équivalentes par comparaison d'exemples.
Thèse 3ème cycle Univ. Nancy I (déc. 1982).

[ROBI 65] J.A. ROBINSON : A machine oriented logic based on the resolution principle.
JACM - Vol 12 N° 1 (janv. 65) [pp. 23-41].

[SACE 74] E.D. SACERDOTI : Planning in a hierarchy of abstraction Spaces. AI - 5(2) - 1974 [pp. 115-135].

[SALE 80] A. SALES : Utilisation de la notion de type abstrait générique en bases de données relationnelles.
Actes du congrès AFCET - NANCY (1980).

[SCH 75] R.C. SCHANK, R.P. ABELSON : Scripts, plans & knowledge
IJCAI 1975 [pp. 151-157].

[SHIP 81] D.W. SHIPMAN : The functional data model and the data language DAPLEX.
ACM TODS - Vol 6,1 - Mars 1981 [pp. 140-173].

[SHOR 81] E.H. SHORTLIFE & al : ONCOCIN : an expert system for oncology protocole management.
I.J.C.A.I. 1981 [pp. 876-881].

[SMIT 77a] J.M. & D.C.P. SMITH : Data base abstractions : Agregation and generalization.
ACM TODS Vol20,2 - juin 1977 [pp. 105-133].

[SMIT 77b] J.M. & D.C.P. SMITH : Data base abstractions : agregation.
CACM Vol 20,6 - juin 1977 [pp. 405-413].

[SMIT 82] J.M. & D.C.P. SMITH : Principles of data base conceptual design.
in LNCS 132 (1982) - Data base Techniques vol I
[pp. 114-146].

[STEF 82] M. STEFIK & al : The organization of expert systems : a tutorial. AI. 1982 [pp. 135-173].

[STON 75] M. STONEBAKER : Getting started in INGRES : a tutorial.
Memo ERL. M518 (Avril 1975).

[SUND 74] Bo SUNDGREN : Conceptual foundation of the infological approach to databases in D.B. Management - Klimbie & Koffeman Eds.
North Holland Pub. Comp. (1974) [pp. 61-96].

[TARD 75] H. TARDIEU, H. HECKENROTH, D. NANCI : Etude d'une méthodologie d'analyse et de conception d'une base de données. CETE AIX-EN-PROVENCE (Mai 1975).

[TARD 76] H. TARDIEU, H. HECKENROTH, D. NANCI, A. ISSADI : Méthode, modèles et outils pour la conception de la base de données d'un système d'information : Manuel de référence. CETE AIX-EN-PROVENCE (Dec 76).

[TARD 79] H. TARDIEU, D. NANCI, D. PASCOT : Conception d'un système d'information : construction de la base de données. Editions d'organisation (1979).

[TARD 83] H. TARDIEU, A. ROCHFELD, R. COLETTI : La méthode MERISE - Principes et outils.
Editions d'organisation (1983).

[TOMP 80] F.W. TOMPA : A practical example of the specification of abstract data types.
ACTA INFORMATICA - 13 (1980) [pp. 205-224].

[TOUN 83] N. TOUNSI : TYP-R : réalisation en TYP d'un système multibase relationnel.
Thèse 3ème cycle Univ. NANCY I (avril 1983).

[WEIN 81] J.L. WEINER & M. PALMER : The design of a system for designing knowledge representation systems
IJCAI 1981 [pp. 277-282].

[WULF 76] W.A. WULF, R.L. LONDON, M. SHAW : An introduction to the construction and verification of ALPHARD programs
IEEE Trans. on Soft. Eng. Vol SE2, 4 (Dec. 76).

[WULF 77] W.A. WULF, M. SHAW : Abstraction and verification in ALPHARD : Defining and specifying iteration and gegerators.
CACM Vol 20,8 - août 77 [pp. 553-564].

[YEH 78] R.T. YEH (Editor) : Current trends in programming methodology.
R.T. YEH Ed. - PRENTICE HALL.

institut
national
polytechnique
de lorraine

Le Président,

N/Réf. : Scol.

AUTORISATION DE SOUTENANCE DE THESE DE DOCTORAT-D'INGENIEUR

-:-:-:-:-:-:-:-:-:-:-:-:-:-:-

VU LE RAPPORT ETABLI PAR :

Monsieur le Professeur CHABRIER J-J.

le Président de l'Institut National Polytechnique de Lorraine autorise :

Monsieur BOUDJLIDA Nacer

à soutenir, devant l'I.N.P.L., une thèse intitulée :

SINDBAD : UN SYSTEME EXPERIMENTAL D'AIDE A LA SPECIFICATION ET A L'UTILISATIO DE BASES DE DONNEES DEDUCTIVES. CONCEPTS ET OUTILS BASES SUR LES TYPES ABSTRAITS DE DONNEES

en vue de l'obtention du titre de DOCTEUR-INGENIEUR

Spécialité *"INFORMATIQUE"*

Fait à NANCY, le *17 Avril 1984*

Le Président de l'I.N.P.L.

M. LUCIUS

R E S U M E

L'expérience SINDBAD (Système INteractif de Déduction dans des Bases Abstraites de Données) a pour objectif de contribuer à l'élaboration de systèmes de données déductifs, selon une approche par les types abstraits de données. Elle a privilégié deux aspects :

- l'un concerne les capacités de ces systèmes à déduire des données non mémorisées à partir de celles qui le sont explicitement.
- l'autre est relatif à leurs aptitudes à s'accommoder d'expressions incomplètes de requêtes.

Nous montrons comment le premier aspect peut être concrétisé à l'aide d'un système de spécification et d'implantation de types abstraits de données (VEGA) doté d'un mécanisme de déduction (celui de PROLOG).

Nous présentons comment SINDBAD permet d'étendre les possibilités "d'utilisation naïve" de ces systèmes à l'aide d'un langage d'interrogation simple et d'un mécanisme intéractif de compréhension de requêtes. (SINDBAD offre également une aide à l'utilisateur pour spécifier sa (ses) [illegible] de données).

MOTS-CLES :

Systèmes experts - Bases de données déductives - abstraite - déduction - résolution - complétion - comp sion - requête incomplète - types abstraits de donnée abstraction - spécification.