Le modèle de données et sa représentation relationnelle dans un système de gestion de base de données généralisées : projet TIGRE

M. José Palazzo Moreira de Oliveira

THESE

présentée à

l' Institut National Polytechnique de Grenoble

pour obtenir le grade de

DOCTEUR INGENIEUR

"Informatique"

par

M. José

PALAZZO MOREIRA DE OLIVEIRA

LE MODELE DE DONNEES ET SA REPRESENTATION RELATIONNELLE DANS UN SYSTEME DE GESTION DE BASE DE DONNEES GENERALISEES.

Projet TIGRE

Thèse soutenue le 29 juin 1984 devant la Commission d'Examen :

Monsieur	C. DELOBEL	:	Président
Messieurs	M. ADIBA M. LEONARD M. LOPEZ J. MOSSIERE		Examinateurs

INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

Année universitaire 1982-1983

Président de l'Université : D. BLOCH

Vice-Président : René CARRE
Hervé CHERADAME
Marcel IVANES

PROFESSEURS DES UNIVERSITES :

ANCEAU François	E.N.S.I.M.A.G.
BARRAUD Alain	E.N.S.I.E.G.
BAUDELET Bernard	E.N.S.I.E.G.
BESSON Jean	E.N.S.E.E.G.
BLIMAN Samuel	E.N.S.E.R.G.
BLOCH Daniel	E.N.S.I.E.G.
BOIS Philippe	E.N.S.H.G.
BONNETAIN Lucien	E.N.S.E.E.G.
BONNIER Etienne	E.N.S.E.E.G.
BOUVARD Maurice	E.N.S.H.G.
BRISSONNEAU Pierre	E.N.S.I.E.G.
BUYLE BODIN Maurice	E.N.S.E.R.G.
CAVAIGNAC Jean-François	E.N.S.I.E.G.
CHARTIER Germain	E.N.S.I.E.G.
CHENEVIER Pierre	E.N.S.E.R.G.
CHERADAME Hervé	U.E.R.M.C.P.P.
CHERUY Arlette	E.N.S.I.E.G.
CHIAVERINA Jean	U.E.R.M.C.P.P.
COHEN Joseph	E.N.S.E.R.G.
COUMES André	E.N.S.E.R.G.
DURAND Francis	E.N.S.E.E.G.
DURAND Jean-Louis	E.N.S.I.E.G.
FELICI Noël	E.N.S.I.E.G.
FOULARD Claude	E.N.S.I.E.G.
GENTIL Pierre	E.N.S.E.R.G.
GUERIN Bernard	E.N.S.E.R.G.
GUYOT Pierre	E.N.S.E.E.G.
IVANES Marcel	E.N.S.I.E.G.
JAUSSAUD Pierre	E.N.S.I.E.G.
JOUBERT Jean-Claude	E.N.S.I.E.G.
JOURDAIN Geneviève	E.N.S.I.E.G.
LACOUME Jean-Louis	E.N.S.I.E.G.
LATOMBE Jean-Claude	E.N.S.I.M.A.G.

.../...

LESSIEUR Marcel	E.N.S.H.G.
LESPINARD Georges	E.N.S.H.G.
LONGEQUEUE Jean-Pierre	E.N.S.I.E.G.
MAZARE Guy	E.N.S.I.M.A.G.
MOREAU René	E.N.S.H.G.
MORET Roger	E.N.S.I.E.G.
MOSSIERE Jacques	E.N.S.I.M.A.G.
PARIAUD Jean-Charles	E.N.S.E.E.G.
PAUTHENET René	E.N.S.I.E.G.
PERRET René	E.N.S.I.E.G.
PERRET Robert	E.N.S.I.E.G.
PIAU Jean-Michel	E.N.S.H.G.
POLOUJADOFF Michel	E.N.S.I.E.G.
POUPOT Christian	E.N.S.E.R.G.
RAMEAU Jean-Jacques	E.N.S.E.E.G.
RENAUD Maurice	U.E.R.M.C.P.P.
ROBERT André	U.E.R.M.C.P.P.
ROBERT François	E.N.S.I.M.A.G.
SABONNADIERE Jean-Claude	E.N.S.I.E.G.
SAUCIER Gabrielle	E.N.S.I.M.A.G.
SCHLENKER Claire	E.N.S.I.E.G.
SCHLENKER Michel	E.N.S.I.E.G.
SERMET Pierre	E.N.S.E.R.G.
SILVY Jacques	U.E.R.M.C.P.P.
SOHM Jean-Claude	E.N.S.E.E.G.
SOUQUET Jean-Louis	E.N.S.E.E.G.
VEILLON Gérard	E.N.S.I.M.A.G.
ZADWORNY François	E.N.S.E.R.G.

PROFESSEURS ASSOCIES

BASTIN Georges	E.N.S.H.G.
BERRIL John	E.N.S.H.G.
CARREAU Pierre	E.N.S.H.G.
GANDINI Alessandro	U.E.R.M.C.P.P.
HAYASHI Hirashi	E.N.S.I.E.G.

PROFESSEURS UNIVERSITE DES SCIENCES SOCIALES (Grenoble II)

BOLLIET Louis
Chatelin Françoise

PROFESSEURS E.N.S. Mines de Saint-Etienne

RIEU Jean
SOUSTELLE Michel

CHERCHEURS DU C.N.R.S.

FRUCHART Robert	Directeur de Recherche
VACHAUD Georges	Directeur de Recherche

.../...

ALLIBERT Michel	Maître de Recherche
ANSARA Ibrahim	Maître de Recherche
ARMAND Michel	Maître de Recherche
BINDER Gilbert	
CARRE René	Maître de Recherche
DAVID René	Maître de Recherche
DEPORTES Jacques	
DRIOLE Jean	Maître de Recherche
GIGNOUX Damien	
GIVORD Dominique	
GUELIN Pierre	
HOPFINGER Emil	Maître de Recherche
JOUD Jean-Charles	Maître de Recherche
KAMARINOS Georges	Maître de Recherche
KLEITZ Michel	Maître de Recherche
LANDAU Ioan-Dore	Maître de Recherche
LASJAUNIAS J.C.	
MERMET Jean	Maître de Recherche
MUNIER Jacques	Maître de Recherche
PIAU Monique	
PORTESEIL Jean-Louis	
THOLENCE Jean-Louis	
VERDILLON André	

CHERCHEURS du MINISTERE de la RECHERCHE et de la TECHNOLOGIRE (Directeurs et Maîtres de Recherches, ENS Mines de St. Etienne)

LESBATS Pierre	Directeur de Recherche
BISCONDI Michel	Maître de Recherche
KOBYLANSKI André	Maître de Recherche
LE COZE Jean	Maître de Recherche
LALAUZE René	Maître de Recherche
LANCELOT Francis	Maître de Recherche
THEVENOT François	Maître de Recherche
TRAN MINH Canh	Maître de Recherche

PERSONNALITES HABILITEES à DIRIGER des TRAVAUX de RECHERCHE (Décision du Conseil Scientifique)

ALLIBERT Colette	E.N.S.E.E.G.
BERNARD Claude	E.N.S.E.E.G.
BONNET Rolland	E.N.S.E.E.G.
CAILLET Marcel	E.N.S.E.E.G.
CHATILLON Catherine	E.N.S.E.E.G.
CHATILLON Christian	E.N.S.E.E.G.
COULON Michel	E.N.S.E.E.G.
DIARD Jean-Paul	E.N.S.E.E.G.
EUSTAPOPOULOS Nicolas	E.N.S.E.E.G.
FOSTER Panayotis	E.N.S.E.E.G.

.../...

GALERIE Alain	E.N.S.E.E.G.
HAMMOU Abdelkader	E.N.S.E.E.G.
MALMEJAC Yves	E.N.S.E.E.G. (CENG)
MARTIN GARIN Régina	E.N.S.E.E.G.
NGUYEN TRUONG Bernadette	E.N.S.E.E.G.
RAVAINE Denis	E.N.S.E.E.G.
SAINFORT	E.N.S.E.E.G. (CENG)
SARRAZIN Pierre	E.N.S.E.E.G.
SIMON Jean-Paul	E.N.S.E.E.G.
TOUZAIN Philippe	E.N.S.E.E.G.
URBAIN Georges	E.N.S.E.E.G. (Laboratoire des ultra-réfractaires ODEILLON)
GUILHOT Bernard	E.N.S. Mines Saint Etienne
THOMAS Gérard	E.N.S. Mines Saint Etienne
DRIVER Julien	E.N.S. Mines Saint Etienne
BARIBAUD Michel	E.N.S.E.R.G.
BOREL Joseph	E.N.S.E.R.G.
CHOVET Alain	E.N.S.E.R.G.
CHEHIKIAN Alain	E.N.S.E.R.G.
DOLMAZON Jean-Marc	E.N.S.E.R.G.
HERAULT Jeanny	E.N.S.E.R.G.
MONLLOR Christian	E.N.S.E.R.G.
BORNARD Guy	E.N.S.I.E.G.
DESCHIZEAU Pierre	E.N.S.I.E.G.
GLANGEAUD François	E.N.S.I.E.G.
KOFMAN Walter	E.N.S.I.E.G.
LEJEUNE Gérard	E.N.S.I.E.G.
MAZUER Jean	E.N.S.I.E.G.
PERARD Jacques	E.N.S.I.E.G.
REINISCH Raymond	E.N.S.I.E.G.
ALEMANY Antoine	E.N.S.H.G.
BOIS Daniel	E.N.S.H.G.
DARVE Félix	E.N.S.H.G.
MICHEL Jean-Marie	E.N.S.H.G.
OBLED Charles	E.N.S.H.G.
ROWE Alain	E.N.S.H.G.
VAUCLIN Michel	E.N.S.H.G.
WACK Bernard	E.N.S.H.G.
BERT Didier	E.N.S.I.M.A.G.
CALMET Jacques	E.N.S.I.M.A.G.
COURTIN Jacques	E.N.S.I.M.A.G.
COURTOIS Bernard	E.N.S.I.M.A.G.
DELLA DORA Jean	E.N.S.I.M.A.G.
FONLUPT Jean	E.N.S.I.M.A.G.
SIFAKIS Joseph	E.N.S.I.M.A.G.
CHARUEL Robert	U.E.R.M.C.P.P.
CADET Jean	C.E.N.G.
COEURE Philippe	C.E.N.G. (LETI)

.../...

DELHAYE Jean-Marc	C.E.N.G. (STT)
DUPUY Michel	C.E.N.G. (LETI)
JOUVE Hubert	C.E.N.G. (LETI)
NICOLAU Yvan	C.E.N.G. (LETI)
NIFENECKER Hervé	C.E.N.G.
PERROUD Paul	C.E.N.G.
PEUZIN Jean-Claude	C.E.N.G. (LETI)
TAIEB Maurice	C.E.N.G.
VINCENDON Marc	C.E.N.G.

LABORATOIRES EXTERIEURS

DEMOULIN Eric	C.N.E.T.
DEVINE	C.N.E.T. (R.A.B.)
GERBER Roland	C.N.E.T.
MERCKEL Gérard	C.N.E.T.
PAULEAU Yves	C.N.E.T.
GAUBERT C.	I.N.S.A. Lyon

Je tiens particulièrement à remercier:

Monsieur C Delobel, Professeur à l'université Scientifique et Medicale de Grenoble, de m'avoir fait l'honneur de présider le jury de cette thèse;

Monsieur M ADIBA, Professeur à l'Université Scientifique et Medical de Grenoble, pour m'avoir reçu dans l'équipe de Base de Données et avoir pris la responsabilité de cette thèse. Je veux aussi lui exprimer m'a plus sincère gratitude pour la confiance qu'il m'a toujours témoigné et pour sa totale disponibilité;

Monsieur M LEONARD, Professeur à l'Université de Genève, pour ses critiques concernant le modèle TIGRE, pour avoir bien voulu juger mon travail et faire partie de ce jury;

Monsieur M LOPEZ, ingénieur de recherche de la compagnie BULL, pour avoir bien voulu faire partie de ce jury;

Monsieur J MOSSIERE, Professeur à l'Institut National Polytechnique de Grenoble et Directeur du Laboratoire de Génie Informatique de l'IMAG, pour avoir bien voulu faire partie de ce jury;

Tous les membres de l'équipe TIGRE pour l'amitié qu'ils m'ont toujours témoignée.

le Service de Réprographie de l'IMAG, qui a assuré le tirage de ce document;

l'Universidade Federal do Rio Grande do Sul, Brésil, et la CAPES (Coordenação de Aperfeiçoamento do Pessoal do Ensino Superior) pour leur soutien financier.

RESUME

Cette thèse traite du problème des bases de données où les informations à stocker et à manipuler sont de natures différentes de celles habituellement traitées par les SGBDs. Parmi ces données on peut citer les textes et les images. Nous présentons le modèle de données TIGRE comme une extension du modèle Entité-Association. Il comprend les concepts d'agrégation et de généralisation et les types de données nécessaires pour la stockage et la manipulation des données généralisées. Nous proposons également la modélisation des données temporelles.

Nous décrivons ensuite la réalisation effectuée dans le cadre du serveur de base de données TIGRE. Cette réalisation permet de traduire les énoncés de définition du modèle TIGRE en une représentation relationnelle n-aire et faire la création d'une base de données TIGRE.

MOTS CLE

Base de données généralisées, modèle de données, architecture de SGBD, représentation du temps et de documents, modèle Entité-Association.

à ma famille

TABLE DE MATIERES

BIBLIOGRAPHIE

Chapitre 0

INTRODUCTION

0. Introduction

0.1 Nouvelles perspectives en bases de données

Les systèmes de base de données disponibles actuellement sont capables de gérer de grandes quantités de données élémentaires. En général ces données sont de petites tailles et les possibilités de structuration restent faibles. Ces SGBDs sont utilisés principalement dans le domaine de la gestion. Cependant, beaucoup d'applications ont besoin de stocker et de manipuler des données généralisées tels que les documents structurés, les formulaires, les images et graphiques, par exemple.

Contrairement aux données alphanumériques classiques, les données généralisées ne sont pas atomiques et possèdent une structure interne et une sémantique qui leur est propre. Elles ont, aussi, une taille importante, ce qui pose des nouveaux problèmes de stockage et de manipulation.

Quelques exemples des applications généralisées sont: la bureautique qui manipule des documents, la conception assistée par l'ordinateur -CAO- qui manipule des structures de données très complexes, le génie logiciel avec la gestion de programmes et leur documentation, l'économie avec ses séries chronologiques et ses modèles, la cartographie, le traitement d'images etc.

Parmi ces applications la bureautique est l'une des premières qui peut tirer partie de la technologie des bases de données. Actuellement les fonctionalités des systèmes de traitement de texte sont très limitées dans le stockage et la classification. Ces possibilités de manipulation sont liées étroitement à une représentation lineaire des documents. Par opposition, les SGBDs actuels offrent des outils assez pauvres

pour la manipulation des données textuelles. En plus il y a une difficulté grave de communication entre ces deux outils de la bureautique.

Un étude générale des nouvelles perspectives concernat les applications, fonctionnalités et technologies des bases de données a été developpé par le groupe BD3 de l'INRIA et ses resultats publiés dans <BD3_83>.

Ces nouvelles applications ont besoin de modèles de données plus riches, prenant en compte davantage d'aspects sémantiques de l'application et elles demandent des outils de manipulation plus puissants en raison de la complexité des données. Après cette conclusion le rapport du groupe BD3 suggère des travaux pour le developpement des nouvelles possibilités. Les différents niveaux indiqués sont:

1) expérimentation dans de nouveaux domaines tels que la Conception Assisté par Ordinateur, la bureautique etc.

2) prise en compte des résultats de recherche aquis, en particulier ceux des modèles et langages de l'intelligence artificielle;

3) recherche de base pour améliorer les résultats théoriques disponibles, par exemple, sur la dérivation d'informations implicites, sur les contraintes d'intégrité, sur l'intégration entre le langage de définition et le langage de manipulation, etc, de manière à prendre en compte les modèles réels;

4) développement de solutions efficaces sur le plan informatique à ces problèmes;

5) étude d'architectures de matériel intégrant les interfaces d'entrée et sortie relatives aux donées multi-media et les modes de stockage des données.

0.2 Description générale du projet TIGRE

Pour developper la recherche sur le thème des bases de données généralisées un projet appelé TIGRE -Traitement d'Information Généralisée et REpartie- fut lancé en 82, <TIGRE_1>. Ce projet fut précédé par l'étude des bases de données textuelles, <JKL_81> et <KL_82>, et par des études préliminaires sur des nouvelles applications de base de données, <ABDR_83>. Des besoins de la CAO, des modèles économiques et l'étude de la voix ont été modélisés a l'aide du modèle entité-association Z, <Abr_78>.

Le projet TIGRE est mené conjointement à Grenoble par le Centre de Recherche Bull et le laboratoire de Génie Informatique de l'IMAG. Sa finalité est l'étude des applications des bases de données généralisées et les fonctionalités correspondantes du SGBD, ainsi que les organisations de données nécessaires. A terme, il doit conduire à la réalisation d'un système experimental s'appuyant sur un serveur base de données généralisées et plusieurs postes de travail communicant grace à un réseau local, fig 1.

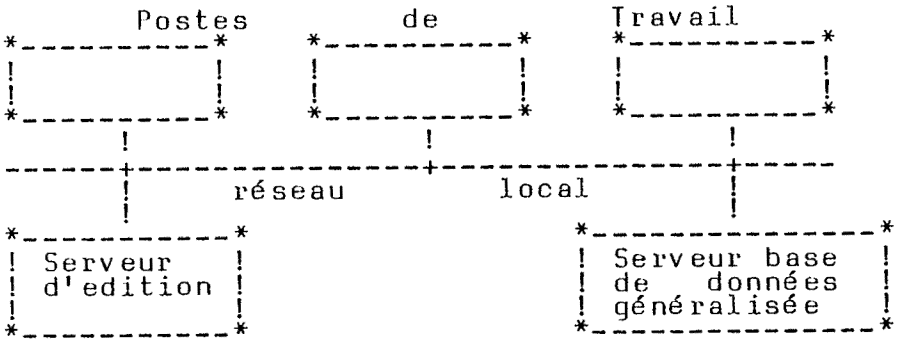


Figure 0.1 : L'architecture générale.

Une importance particulière a été donnée aux applications bureautiques. Le document généralisé, <TIGRE_3>, sert d'unité d'échange conmune dans le système. Néanmois , des études sont menées dans d'autres domaines comme la CAO et la progammation logique, <TIGRE_13>, <TIGRE_7> et <TIGRE_12>.

Le prototype du SGBD TIGRE est construit sur un système de gestion de base de données relationnel et de modules independants pour la gestion des objets généralisés. L'interface entre les deux niveaux se compose de primitives de l'algébre relationnelle et des primitives spécifiques pour les objets généralisés tels que les documents. Dans la mesure du possible le serveur TIGRE sera compatible avec tout SGBD relationnel offrant une interface algèbrique, ce qui permetra, en particulier, d'employer la machine base de données VERSO, developpée à l'INRIA, <BS_80> et <BS_82>.

L'ensemble de la recherche dans le cadre du projet TIGRE est presenté dans les différents Rapports de Recherche TIGRE (voir bibliographie).

0.3 Présentation de la thése

A partir des resultat des recherches sur les bases de données textuelles commencées à Grenoble en 79 et rapportées dans <JKL_81>, <Jol_82> et <KL_82> une première version d'un modèle de données avait été developpé pour TIGRE, <Vel_82> et <JLV_83>. Dans le cadre du developpement de l'application burautique du projet TIGRE, nous avons travaillé d'abord à la définition du modèle de données généralisé TIGRE, tel qu'il existe aujourd'hui. Ce modèle est capable de prendre en compte aussi bien des données alphanumériques classiques que des documents volumineux et complexes. Un document généralisé, dans TIGRE, peut contenir du texte mais aussi des images et des formulaires, <TIGRE_3> et <TIGRE_6>.

Nous avons ensuite participé à la définition de l'architectrure et à la réalisation du prototype du serveur de base de données généralisé developpé sur le système MULTICS. Nous avons traité plus particulièrement la traduction des énoncés de définition modèle généralisé en termes relationnels. La description du modèle de données TIGRE est faite au chapitre 1.

Un des types de données avec une sémantique particulière est le temps. Beaucoup d'applications telles que les modèles économétriques et la bureautique ont besoin, dans la base de données, du concept de temps d'une façon explicite. Nous proposons comme extension au modèle TIGRE l'introduction du concept de temps. Cette extension est décrite au chapitre 2.

L'emploi d'un modèle aussi riche que le modèle TIGRE, pour la représentation de systèmes réels ne se fait pas sans difficultés. Nous proposons, dans le chapitre 3, une methodologie pour la definition d'un schéma conceptuel.

Les différentes architectures de SGBD et celle du serveur de base de données TIGRE sont discutées au chapitre 4. Nous décrirons, en détail, au chapitre 5 la représentation du schéma conceptuel en termes relationnels.

Chapitre 1

LE MODELE DE DONNEES TIGRE

1. Le modèle de données TIGRE

Un modèle de données est un formalisme utilisé pour représenter les objets qui appartiennent à la partie de la realité concernée par un groupe d'applications et leur sémantique. La sémantique correspond aux propriétés de structure et de comportement des objets selectionés. En particulier, elle prend en compte l'effet des opérateurs existants sur les objets.

Le modèle de données TIGRE a été developpé dans le but de fournir les types de données et le formalisme necessaires à la modélisation d'applications qui manipulent des données généralisées. Il intègre des concepts developpés dans le domaine des bases de données (Entité-Association, <Che_76>) et dans le domaine des langages de programmation (types de données).

Notre modèle appartient à la classe de modèles sémantiques. Nous avons consideré que les structures de données d'un modèle de données doivent exprimer une grande portion de la signification d'une base de donées, <HM_81>. Après l'analyse des différents modèles proposés il nous parait clair qu'il est necessaire de pouvoir représenter des objets complexes dans le schéma. Cette position se retrouve dans d'autres travaux, par exemple dans <Schm-77>, <SS_77>, <Cod_79>, <SNF_80>, <HM_81>.

L'approche la plus courante pour la gestion des données généralisées consiste à etendre un modéle et le système existant pour tenir compte d'un nouveau type de données: données textuelles <HL_81>, <KL_82>, <SSKG_82>; données photographiques <Tan_81>; formulaires <Tsi_80>, <Tsi_81>. D'autres efforts pour intégrer les bases de données et la bureautique sont <Zlo_79>, <Hamm_79>, <EN_79> et <RSh_82>.

Par rapport aux systèmes existantes ou proposés le Projet TIGRE offre un modèle homogéne où la définition et manipulation des différents types de données sont faites par un même langage.

La demarche suivie pour le developpement du modèle TIGRE a été le choix du modèle Entité-Association pour son interpretation naturelle de la realité perceptible. Après ce choix nous en avons developpé une extension en incluant des concepts de propriétés atomiques et structurées ("atomic and molecular semantics", <Cod_79>) représentées par les types non structurés et structurés et, aussi, par les concepts de généralisation et d'agrégation. Le concept de types structurés est le moyen d'inclure dans le modèle des nouveaux types de données telles que les documents, les formulaires etc. Deux formes d'agrégation sont fournies: agrégation d'entités et agrégation d'association.

La représentation du sous-ensemble de la réalité est faite par un schéma conceptuel. Un schéma représente les objets, et leurs propriétés structurelles, comme un ensemble de types de données. Ces types appartiennent à des structures définies par le modèle de données, <Bro_80>. Pour décrire le schéma conceptuel a été crée un langage de définition de données apelé LAMBDA, <Vel_82>, <TIGRE_2>. La syntaxe BNF des enoncés de définition de LAMBDA est présentée dans l'annexe A1.

Les types du modèle TIGRE sont une extension de certains types rencontrés dans le langage PASCAL, <JW_78>. L'homogéneité de la définition des types facilite l'intégration entre LAMBDA et PASCAL qui a été choisi comme langage hôte.

Deux catégories de types non structurés sont définies: les types de base (simples et restreints) et les types renommés. Les types structurés sont réunis dans deux groupes: les types

construits et les types classes. Les types construits sont: enregistrement, liste et document. Les types classe corrèspondent aux classes du modèle Entité-Association, ils sont les types entité et le type association.

Dans les sections suivantes nous ferons la déscription des différents types de données et la correspondance entre les variables du langage de programation, les classes de la base de données et les types classe.

1.1 Les types non structurés

Les types non structurés correspondent aux domaines dans le modèle relationnel. Nous employons les types non structurés pour définir des ensembles de valeurs atomiques.

1.1.1 Types simples

Les types simples sont les composantes à partir desquelles tous les autres types sont construits. L'exception est le type documment qui est défini dans un environement independent. Les types simples sont: booléen, entier, réel et chaine de caractères. Les types booléen, entier et réel sont identiques aux types PASCAL. Le type chaine de caractères est une suite de caractères tout'à fait similaire à un "packed array" de caractères.

1.1.2 Types restreints

Les types restreints sont les types scalaires et intervalles. Ils correspondent aux types "scalar" et "subrange" de PASCAL. Le type intervalle a la restriction, par rapport au type "subrange", de ne pouvoir être défini que sur un type simple entier. Nous les appellons restreints parce qu'ils spécifient

implicitement une contrainte d'integrité sur les types sur lesquels ils sont définis. Cette contrainte est une restriction sur l'ensemble des valeurs qu'une propriété modélisée par ce type peut avoir. A part cette restriction, les types restreints ont les mêmes caractéristiques que les types simples sur lesquels ils sont définis. En particulier les theta-produits sont acceptés entre deux attributs définis sur un type restreint et sur un type égal au type simple sous-jacent au type restreint. Ils sont aussi acceptés entre deux attributs définis sur des types restreints qui ont le même type simple sous-jacent, nous appelons ces types "equivalents".

Exemples: (vert, rouge, noir)
(1..120)

1.1.3 Types renommés

En PASCAL les types définis sur les mêmes ensembles de valeurs sont vus comme égaux. Dans la définition du schéma d'une base de données cette approche n'est pas envisageable parce qu'elle traite de la même façon de propriétés sémantiquement différentes. Avec une telle définition il pourrait être possible de selectioner tous les patients qui ont une âge égal a leur témperature si ces attributs ont été définis sur des types identiques! Cette erreur serait dûe à la définition de l'égalité des types de PASCAL. Pour permetre la distinction entre de types représentant des propriétés sémantiquement différentes nous avons introduit en LAMBDA le type renommé.

Un type renommé est un emploi particulier d'un type de base distingué par un identificateur unique. L'identificateur est employé pour montrer la signification du type, il peut être consideré comme le nom de l'unité, <Bro_78>.

```
Exemples:    type nom          : string (24) ;
             type temperature  : real ;
             type code_postal  : integer ;
             type age          : (0..120) ;
```

1.2 Les types structurés

Le concept de type structuré est un moyen d'incorporer plus de signification dans les structures de données. Nous avons choisi d'offrir aux usagers la possibilité de modéliser les propriétés d'un objet en ensembles minimaux significatifs, tels que une adresse composée par le numéro de l'habitation, le nom de la rue, le code postal, le nom de la ville et du pays, ou une liste de valeurs. Les types structurés permetent, aussi, l'inclusion dans le modèle des nouveaux types de données généralisés tels que les documents.

Les types structurés disponibles dans le modèle TIGRE sont: les types construits (type liste, type enregistrement, type document) et les types classe.

1.2.1 Types construits

Type liste

Le type liste est similaire au type tableau ("array") à une dimension de PASCAL. Une liste permet l'accès direct à ses composantes. Les composants d'un type liste appartiennent à un des types non structurés.

Des propriétés qui ont des valeurs multiples telles que l'ensemble des travaux d'un employé ou l'ensemble des plus hautes temperatures pendant un année peuvent être représentées par types liste:

```
type Hist_travaux : list (12) of string (15) ;
type Maxima       : list (365) of real ;
```

Le type liste est utile pour modéliser des dépendences multivaluées. Par exemple, si pour l'entité Cours la connaissance du nom detérmine l'ensemble des livres employés dans le cours nous pourrons représenter cette dépendence par:

```
type Cours : entity

    key nom   : string (12) end_key ;
        livres: list (3) of string (18)
end ;
```

Type enregistrement

Plusieurs fois il est nécessaire de représenter des objets qui ont une existence qui dépend d'autres objets. Par exemple, l'adresse d'une personne est un objet pour lequel nous ne voulons stocker la valeur dans la base que si cette personne est aussi représentée dans la base. Pour la modélisation de ce type d'objet la solution est l'emploi du type enregistrement.

Exemple:

```
type Adresse : record
                   no          : integer ;
                   rue         : string (20) ;
                   code_postal : integer ;
                   ville       : Nom_de_ville ;
                   pays        : Nom_de_pays
               end ;
```

Dans cet exemple les propriétés numéro, nom de la rue et code postal sont définis sur des types de base et les propriétés nom de la ville et du pays sur des types renommés. L'entité Personne aura une propriété représentée par un attribut défini

sur le type structuré Adresse.

Un attribut est l'emploi particulier d'un type pour représenter d'une propriété d'un objet. Les attributs définis sur des types structurés permettent un degré d'abstraction dans la définition descendent du schéma. Dans un premier temps il est possible d'identifier et de nommer la propriété et ensuite, il faut faire la définition du type structuré sous-jacent. Dans le modèle TIGRE un type enregistrement doit être composé par de types non structurés.

Type document

Le type document est utilisé pour faire la définition de ce que nous appelons un "document généralisé", <TIGRE_3>, <TIGRE_6>, <BRV_83>. Le type document spécifie une structure hiérarchique qui doit être satisfaite par toutes les occurrences du type. Cette structure sert à construire les documents, à les parcourrir, à les présenter, ou encore à faire référence à d'autres parties dans le même document , par exemple une référence à un autre chapitre.

La structure hiérarchique d'un document est construite à l'aide d'éléments composés qui sont les noeuds non terminaux. Ces éléments composés sont obtenus par des opérateurs de composition à partir des objets déjà définis. Ces derniers peuvent être des composants élémentaires appelés unités, des éléments composés ou des structures hiérarchiques de documents déjà définis.

Le terme générique unité recouvre tous les éléments terminaux acceptés par la grammaire de document. Une unité possède l'une des natures suivantes: textes, graphique, image, formule, tableau, programme, vocal ou référence à un élément associé.

Dans la définition d'un type document il est possible d'employer certains éléments dits associés. Leur position dans un document n'est pas toujours liée à la structure mais peut être liée à la présentation. Chaque élément associé appartient à une classe qui definit sa structure, ces classes sont: illustrations, note, énoncé bibliographique, marque interne, commentaire, paramètre et fonction.

Exemple: Copyright apparaisant en note de bas de page de la première page de toutes les publications et référencé après le titre du document.

```
type Première_page : document

        structure
            begin
               titre    : text ;
               ref_note : N1
            end ;

         constants
            unité N1 := 'Copyrigth .......'
            end ;
    end ;
```

Les paramètres et fonctions ont une importance particulière parce qu'elles font la liaison entre les documents, manipulés par l'editeur, et les autre types de données gerés par le serveur de base de données.

Dans un document, un paramètre désigne une partie variable accessible directement par son nom, sans passer par la structure hiérarchique du document. Une fonction désigne un programme de calcul et un domaine d'evaluation. Le résultat d'une évaluation est une unité, <TIGRE_6>.

Il est possible de lier à chaque élément associé ou unité d'un document un ou plusieurs "attributs" , <TIGRE_6>. Un attribut a pour but de fournir une certaine information sémantique (confidentialité, indexation, etc.) ou une certaine présentation à un élément du document. Les attributs de présentation sont utilisés pour la restituition d'un document. Ils sont liés à la nature des éléments qui les portent. Par exemple, la mise en évidence est différente si elle porte sur un texte ou sur une illustration.

Exemple:

```
type Paragraphe

        structure
           list (1,*) of unité
        end
   end ;

type Section : document

        structure
           titre  : text ;
           contenu : list (1,*) of Paragraphe
        end
   end ;
```

```
type  Rapport : document

      structure
       begin
       page_de_garde  : begin
                        entête : text := T0 ;
                        Titre att mise_en_évidence : text ;
                        auteur : référence := nom_auteur
                        end ;
       introduction   : begin
                          titre : text := T1 ;
                          contenu : text
                        end ;
       corps : list (1,*) of Section ;
       conclusion  : Paragraphe
       end ;

       constantes

           begin
             unité T0 := 'Laboratoire IMAG' ;
             unité T1 := 'Introduction' ;
             parametre nom_auteur : Text
            end ;
   end ;
```

1.2.2 Types classe

Type entité

Un type entité est une agrégation des types qui représentent les propriétés des entités qui appartient à une classe

d'entités. Les attributs sont définis sur de types autres que les types classe.

Les types classe sont les uniques que correspondent à des objets independents et à chacun correspond un ensemble de occurrences dans la base de données.

Une entité est identifiée de façon unique par la valeur de un ou plusieurs attributs qui constituent la clé primaire. Si une clé n'a pas été définie l'ensemble des attributs non structurés constitue la clé de l'entité.

Exemples:

```
type Employé : entity

     key numéro    : integer   end key ;
         nom       : string (32) ;
         salaire   : Argent ;
         adresse   : T_adresse
         tâches    : Hist_travaux ;
         catégorie : (ingenieur, programmeur, secretaire)
    end ;

type Rap_recherche : entity

         nro    : integer ;
         mois   : (1..12) ;
         année  : integer ;
         corps  : Rapport
   end ;
```

Comme une clé n'a pas été spécifiée pour l'entité Rap_recherche, l'ensemble des attributs non-structurés est, par defaut, sa clé. L'attribut "corps" n'appartient pas à la

clé parce qu'il est défini sur le type Rapport qui est un type structuré document.

Type association

Les types association représentent un lien sémantique entre entités. Chaque occurrence d'une association lie une ou plusieurs occurrences de chaque clase d'entités participant à l'association. Les propriétés d'une association sont représentées par attributs de la même façon que pour les entités. L'identification d'une occurrence d'une association est faite par la concatenation des clés des occurrences des entités associées.

```
Exemple:    type Localisation : relationship
                    between Employé : employé (1,1)
                    and     Site    : lieu_de_travail (0,*)

                    depuis : jour
            end ;
```

Les types d'entités liées par une association jouent chacun un rôle spécifique qui est indiqué dans la définition de cette association. On peut y ajouter aussi les contraintes de cardinalité minimale et maximale pour les occurrences. Dans l'exemple chaque occurrence d'Employé doit être liée avec exactement une occurrence de Site et pour Site il n'y a pas de restrictions de cardinalité.

Classes, types et variables

Pour designer le schéma d'une classe nous utilisons l'expression type classe. Le mot classe est employé pour désigner l'extension d'une classe et celui d'occurrence pour

désigner une représentation d'un objet entité ou association appartenant à une classe.

Le résultat d'une réquête LAMBDA est une classe d'entités. Cette classe doit être définie par un type entité. Si le type du resultat de la requête n'est pas un type entité connu , il doit être défini avant la requête.

Pour stocker le résultat d'une requête il faut définir une variable temporaire comme dans l'exemple suivante:

```
var Emp_produtif : Employé ;
```

Emp_productif est une variable temporaire ayant le même type qui Employé.

Une variable temporaire appartient à la zone de travail d'un programme. Pour transformer une variable temporaire en une variable permanent ou db_var il faut inclure la définition de son type dans le catalogue, créer une db_var et effectuer le transfert des occurrences appartennant à la variable temporaire vers la nouvelle db_var. L'ensemble de ces actions peuvent être realisées par l'activation de la commande "Restore" du langage de manipulation de données.

Pour interfacer le SGBD avec la langage hote, est faite la définition d'une variable de programmation associée à chaque classe de la base de données. Comme la variable de programmation et la classe sont définies sur le même type classe elles ont la même structure. Cette approche évite la verification de compatibilité entre types pour les opérations d'inclusion.

1.3 Généralisation et agrégation

Le modéle TIGRE offre des opérateurs de généralisation et d'agrégation. Ce sont des opérateurs qui s'appliquent sur des types classe et produissent d'autres types classe dérivés.

Les opérateurs de généralisation sont la spécialisation, l'union et l'intersection. L'agrégation opére sur des entités ou des associations.

1.3.1 Spécialisation

Les opérateurs de spécialisation sont un outil pour modéliser le fait qu'une classe d'objets est plus spécifique ou plus générale qu'une autre. Ils expriment aussi, de façon implicite, une dépendence existentielle et un héritage d'attributs. Les types classe dérivés héritent des attributs du type hiérarchiquement supérieur et ils peuvent avoir des attributs propres.

Par exemple, si le type Programmeur est obtenu par l'application de l'opérateur de spécialisation sur le type Employé alors Programmeur hérite implicitement des attributs de Employé et toute occurrence de Programmeur doit être une occurrence de Employé. Programmeur a un attribut propre "langage" qui caracterise les capacités de travail d'un programmeur.

```
Exemple:    type t_langage    : (cobol, pl1, fortran, pascal) ;
            type Programmeur  : specialization_of Employé
                      langage : list (3) of t_langage
                 end ;
```

Une occurrence de la classe Progammeur aura les attributs: numéro, salaire, adresse, tâche ainsi que catégorie hérités d'Employé ainsi que l'attribut propre langage.

Avec les opérateurs de généralisation il est possible de former ce que nous appelons une hiérarchie de généralisation. Dans ce genre de structure le noeud sommet est un type classe non dérivé et les autres noeds sont des dérivations par généralisation des types parents. La clé primaire de chaque

type classe dérivé, dans une hiérarchie, sera la clé du type au sommet.

L'opérateur de spécialisation produit un type classe spécialisé a partir d'un type argument. Les occurrences de la classe spécialisée sont celles qui satisfont un prédicat de restriction.

Il est possible d'ajouter une condition appellée "manuelle" à un prédicat de restriction si cette option est prise seules les occurrences qui satisfont le prédicat et qui sont explicitement inclues dans le type dérivé appartiendront à celui-ci.

Exemples:

```
type Programmeur s:specialization_of Employé
     where catégorie = 'progammer' ;

type Rep_national : specialization_of Personne
                    manual

        pays : nom_pays
   end ;

type Attendu : specialisation_of Invité
               where reponse = TRUE
               and manual

      inscription : record
                      date  : jour ;
                      value : real
                    end
   end ;
```

Une classe d'association spécialisée répresente des relations entre entités appartenant aux mêmes classes d'entités liées

par la classe d'association père. Elles peuvent aussi appartenir à des classes d'entités obtenues par dérivation des classes liées par l'association de départ. Dans une spécialisation d'association, les noms des rôles restent les mêmes que dans l'association originale. Les contraintes de cardinalité peuvent être restreintes.

```
Exemple:  type Loc_bureau : specialization_of Localisation
                  between Emp_bureau
                  and Site

                  no_bureau : integer ;
                  poste     : (1..4000) ;
          end ;
```

1.3.2 Union et intersection

A partir de deux ou plusieurs types d'entités arguments ayant un ancêtre commun, les opérateurs d'union et d'intersection produisent un type d'entité comme resultat. Une occurrence appartenant à une classe dérivée par union doit appartenir à au moins une classe argument de l'opérateur d'union. D'une façon similaire pour une classe dérivée par intersection, une occurrence doit appartenir à toutes les classes arguments de l'opérateur d'intersection. Des options d'inclusion manuelle pourront être appliquées aux types entité arguments de l'opérateur d'union et au type entité résultat de l'opérateur d'intersection.

Les attributs d'une intersection sont les attributs propres plus ceux qui appartiennent au résultat de l'union ensembliste des attributs des opérandes. Ceux de l'union sont les attributs propres plus ceux qui appartiennent à l'intersection ensembliste des attributs des opérandes.

```
Exemple:  type Emp_bureau : union
                    of  Secretaire
                    and Programmeur

                    no_de_bureau : integer
              end ;
```

1.3.3 Agrégation

Dans le modèle TIGRE, l'agrégation a un objetif double. Le premier est de modéliser les objets qui sont composés par d'autres objets comme un dossier qui, en plus d'attributs propres, est constitué par de lettres, un contrat et des annexes. Le second but est de permettre une vision agregée d'une association. Ainsi, il est possible de redéfinir une association, et les entités liées, comme une entité composée de niveau plus élevé.

```
Exemples:     type Dossier : entity_aggregation
                        of  Lettre  (0,*)
                        and Contrat (1,1)
                        and Annexe  (0,*)

                        no_dossier : integer ;
                        responsable: nom
                  end ;
```

Dans cet exemple nous supposons que les types d'entités Lettre, Contrat et Annexe ont déjà été définis. Une occurrence de Dossier a un "no_dossier" et un "responsable" comme des propriétés propres, zéro ou plusieurs occurrences de l'entité Lettre et de l'entité Annexe et exactement une occurrence de l'entité Contrat.

```
type Fourniture : relationship
        between Fournisseur
        and     Article

        quantité : integer
   end ;

type Expedition : relationship_aggregation
        of Fourniture

        date         : jour ;
        destinataire : code_dest
   end ;
```

L'agrégation associative Expedition permet voir l'association Fourniture comme une entité qui a toutes les attributs de l'association et de ses entités composantes et, en plus, ses attributs propres: date et destinataire.

Chapitre 2

LE TEMPS

2. Le temps

Dans les systèmes traditionnels le contenu de la base est sensé de représenter une photographie de l'état du l'environnement d'une application <HM_81> . Les SGBD actuels offrent au usager la version la plus récente des objets représentés dans la base. A chaque actualisation de la valeur d'un attribut l'ancienne valeur est détruite. Pour plusieurs applications cette modèlisation ne sufit pas. Les modèles économétriques <ABDR_83> et les systèmes pour le controle de la production, par la nature même des séries traitées, conduisent a gérer le temps avec les autres variables - séries chronologiques. Pour les applications bancaires et pour les systèmes médicaux, entre autres, l'information historique compose la plus grande partie des données necessaires. Pour ces types d'applications le concept de temps est un besoin imperatif.

D'autre part dans les SGBDs, les transactions et la journalisation sont étroitemente liées au concept de temps. Chaque enregistrement du journal doit avoir une information sur le moment où il a été généré. Le journal peut être vu comme un ensemble des états de la base , adressables par le temps. Ce parallelisme entre le concept de journalisation et l'emploi d'un domaine défini sur le temps est étudié dans <Gra_81> . Dans le même article et dans <LP_82> sont traités le problème des transactions longues ou transactions conversationelles. Ce dernier type de transaction est en rapport direct avec les gros objets manipulés par TIGRE, tel que les documents. Une transaction sur un document est un processus long qui se deroule par l'action conversationelle entre l'auteur et l'editeur de texte. L'addition du concept de temps d'une façon explicite dans le modèle de données est une aide pour la gestion de ce genre de transactions et aussi une possibilité pour resoudre le probème de la reprise après

panne.

La gestion de temps est aussi un besoin très important pour les applications bureautiques. Un des objectifs du projet TIGRE est servir à cette classe d'applications. Pour savoir de ce qu'il faut fournir comme outils pour la bureautique on peut reprendre la définition proposé dans <Hame_81>. Cette définition dit que les applications bureautiques sont celles particulièrement concernées par:

- la gestion des textes

- la gestion des communications

- la gestion du temps

Le modèle de données TIGRE fournit déjà toute la structure necessaire pour la gestion des textes avec la notion de document (section 1.3). Le modèle de base entité_association fournit les structures necessaires pour le developpement des applications de gestion traditionelles. Jusqu'ici, deux des besoins des applications bureautiques ne sont pas remplis par le modèle/serveur TIGRE: la gestion des communications et la gestion de temps. La gestion des communications, la messagerie életronique, apporte de nouvelles possibilités pour la réorganisation du travail de bureau. La gestion du temps est un besoin plus proche de l'analyse et de la modèlisation des systemes. Nous avons choisi d'aborder ce dernier point en priorité pour avoir un modéle suffisamment complet qui puisse s'employer dans des applications bureautiques. La messagerie électronique, dans un premier temps, pourra être realisée par les procédures actuelles de communication. Plus tard elle doit être étudiée et incluse dans le cadre du projet.

Dans ce chapitre, nous allons faire la description des concepts importants pour l'inclusion de temps dans les bases de données. Ensuite nous allons faire la dèscription de trois méthodologies différentes pour la gestion de temps dans les systèmes informatiques. Les dérnières sections montreront l'extension du modèle TIGRE et du langage de manipulation de données.

2.1 Le temps et les bases de données

Depuis des années le besoin d'avoir la possibilité de maintenir l'information sur les états antérieurs a été mis en evidence. Un des premiers systèmes à manipuler le temps fut "TOD - Time Oriented Database" developpé à l'Université de Stanford <WFW_75>. A cette époque B Sundgren <Sun_75>.faisait resortir l'importance de temps dans la modèlisation des systèmes d'information dans son modèle "Infological Approach to Data Bases". Auparavant plusieurs recherches avaient été réalisées sur le sujet du temps principalement en inteligence artificielle et en logique. A partir de cette époque la notion de temps a été abordée par différents auteurs et des réalisations ont été développées. Nous allons d'abord faire une synthèse des concepts importants pour la manipulation de temps dans les bases de données et nous decrivons ensuite des systèmes qui manipulent le type de données temps.

Le premier concept est lie à la perception de temps par le concepteur de la base. Dans une base de données classique on ne peut accéder qu'à l'état présent de la base. Un état est défini comme une photographie de la base à un instant détérminé. Chaque modification de l'état de la base est associé a un événement. Si on veut connaitre l'instant où l'événement a été produit il faut introduire le concept de temps.

Le temps peut être vu comme isomorphe aux nombres réels. Pour faire la représentation d'un certain point dans le temps la solution, la plus immédiate, est de définir un instant initial et une unité. Chaque événement peut être associé à un point dans le temps, l'interpretation de l'association c'est dire : l'évenement est produit à N unitées de temps à partir de l'instant initial. Si cette solution est très bien adaptée aux problémes physiques elle ne l'est pas aux problémes de

gestion.

Dans cette classe d'applications on employe généralement un calendrier. Un calendrier est une association de périodes, qui peuvent être hiérarchiques, par exemple: mois, jour, heure et minute, à la représentation interne de temps. Il faut noter que les points référencés par un calendrier sont vraiment des intervalles <Fal_74> . Avec l'introduction du concept de calendrier des problémes se posent pour la manipulation des intervalles car les operations pouront être définies sur des intervalles de différentes granularitées.

A partir de la définition du type temps il est possible de définir deux classes d'attributs: les statiques et les dynamiques. Un attribut statique est invariable par rapport au temps, par exemple la date de naissance d'une personne. Les attributs dynamiques sont variables comme l'adresse, qui pet changer plusieurs fois dans la vie d'une personne. Il est possible de définir aussi le maintien de différentes versions d'un objet de la base, chaque version est associée à un instant du passé. Cette génération de differentes copies d'un même objet est connue sous le nom de photographie ("snapshot").

Une base qui maintient des états successifs de certains attributs est appelée une Base de Données Historique ou quelquefois par sa version anglaise "Information Preserving Databases" comme in <KL_83>, ou aussi "Historical Databases" comme in <CW_83> . Une base de données historique permet des requêtes pour des instants définis sur toute sa période d'existence. Dans cette période il y aura des objets qui ont une existence définie sur une sous-période. Si une requête fait référence à ces objets pour un instant non inclus dans sa période d'existence la valeur à retourner doit être indefinie. Cela se produit également quand un attribut a deux valeurs, Vi

et V2, définies respectivement aux temps t1 et t2 et quand une requête faite référence à un instant compris entre t1 et t2. Dans le cas où il est possible de fournir une fonction de calcul exacte pour l'interpolation la valeur résultant est précise. Si on ne peut pas fournir une telle fonction il faut disposer d'une approximation et la valeur obtenue sera affectée d'un erreur.

Des études sur la formalisation du concept de temps ont été entreprises. Au niveau de la représentation de temps dans le schema conceptuel une des plus significatives est l'extension du modèle "Data Semantics" de JR Abrial, <Abr_74>, developpé par TL Anderson, <And_82>, <And_83>. Dans ces articles l'auteur fait une description de son "Extended Abrial Conceptual Level model - EACL" . Ce modèle est constitué par le modèle binaire initial enrichi avec le concept de généralisation. Sur cette extension est défini le "Time Generic", un modèle conceptuel explicite et extensible de temps. Il est basé sur des hypothèses décrites formellement sur les systémes de temps et inclut les descriptions d'intervalles de temps, temps périodique, une metrique temporelle et plusieurs opérateur temporels.

Pour un étude plus étendue sur les différents travaux sur le temps en Informatique voir <BADW_82> . Cette tres complète étude bibliographique présente environ soixate-dix documents significatifs publiées entre 1960 et 1982. Des points importants et actuels pour la recherche sur le temps et les bases de données sont présentés dans l'article d'introduction au panel sur le temps de la conférence SIGMOD_83, <ACJ_83>.

La classification des méthodologies

Les réalisations de systèmes informatiques qui considèrent le temps peuvent être groupées en catégories. Le critère pour la

classification est l'integration croissante du concept de temps dans le système. Dans la première catégorie la manipulation du temps est externe au SGBD. Dans la deuxième le concept de temps est associé au SGBD par des procedures spécifiques définies par l'usager et finalement dans la dernière le type temps appartient au modéle de données. Les trois catégories sont décrites ensuite:

1) la manipulation de temps est faite d'une façon explicite par les programmes d'application. Le SGBD, ou plus généralement le système de gestion de fichiers, ne peut que stocker des données de type traditionel comme des entiers, des réels où des chaines de caractères . Toute sémantique associée au temps appartient à la structure logique des programmes. A ce niveau l'usager doit conaitre la sémantique associée au temps et assurer la validité des opérations sur les données définies sur ce type. L'integtité de la base est aussi de la responsabilité de l'usager . Un système de cette catégorie est présenté dans <WFW_75> .

2) différentes extensions pour la manipulation de données avec une sémantique extendue aux types de données de base sont realisées par des programmes dits "experts". Cette solution a l'avantage d'une modification minimale du SGBD sousjacent. M Stonebraker <OS_82> fait reference à une modification de seulement 62 lignes de code pour permetre au système INGRES la gestion des experts. Le code de toutes les procédures n'arrive pas a un millier de lignes. Pour l'usager, la sémantique d'un attribut défini sur un type étendu, tel que le temps, parait appartenir à la définition de la base. Sa plus grande faiblesse est le decoupage entre l'extension sémantique de temps et la représentation du schema de la base. Autrement dit, il est impossible de représenter

le concept de temps dans le schema conceptuel. Si dans les systèmes où intervient l'intelligence artificielle il peut être interessante d'avoir une extension de la base de données par des programmes experts, dans les extensions sémantiques des bases de données cette solution modifie définition et manipulation des valeurs des attributs.

3) finalement il est possible choisir un modèle sémantique de données et définir une extension pour la gestion de temps <KL_83> . Dans cette solution la sémantique de temps devient structurelle, ell'apartient au modèle de données. La définition d'un schema contient la représentation explicite de temps et des concepts associés tels que des series historiques, versions et photographies. L'incovenient de cette proposition est le besoin de modifier le logiel d'un SGBD pour qu'il accepte le nouveau type de données et la sémantique associée. Dans le cadre du developpment d'un nouveau système de gestion de bases de données c'est la solution la plus envisagable.

Pour ces catégories nous allons donner des exemples caractéristiques. Nous ferons ensuite l'analyse de differents systèmes, de façon à avoir une vision plus complete de chaque methodologie.

2.1.1 Time Oriented Database : TOD

Les concepteurs des systèmes d'information médicale ont perçu depuis longtemps le besoin d'avoir la possibilité de stocker et de manipuler des informations associées au temps. Un des premiers systèmes à remplir cette necessité fut "Time Oriented Database - TOD" developpé par G Wiederhold et allii à l'Université de Stanford, <BADW_82>. Ce système est décrit

dans <WFW_75> et la description suivante est basée sur ces articles.

L'experience clinique de plusieurs patients accumulée sur des années contient des données interessant presque tous les problèmes cliniques. En plus des l'accès aux données sur les maladies antérieures, l'enregistrement medical idéal maintient aussi le raisonnement suivi pour faire face aux problèmes medicaux présentés. La disponibilité du systèmes avec la possibilité de manipulation de grands fichiers a permis le developpement de plusieurs applications orientées vers différentes specialités. Pour arriver à un emploi commun de la base de données et afin de permetre la mise en place d'applications utilisant cette base un système complet a été crée.

Le système TOD contient des informations cliniques organisées d'une façon chronologiques et généralement collectées par les docteurs sur des formulaires orientés pour faire ressortir la sequence des visites. Le schema d'une base de données clinique contient deux types d'entités: le patient et les visites des patients à la clinique. Un schéma est une structure qui définit le contenu spécifique d'une base individuelle. Pour maintenir ce genre d'information, TOD emploi deux types de fichiers: le HEADER qui maintient les données non-chronologiques, tel que le nom du patiente, et PARAMETER dans lequel sont stockées les informations sur les visites. Le fichier HEADER fait référence au fichier PARAMATER au moyen de clés symboliques 'visite-number'. La première et la dernière visite sont référencées par le HEADER, les autres seront chainées doublement dans le fichier des paramètres. La chaine est maintenue ordonnée chronologiquement et il est possible de récuperer la série complète ou un sous-ensemble des dernières visites.

Chaque enregistrement du fichier PARAMETER correspond à une visite et ses elements sont dues à des observations independantes. Les données du type DATE sont stockées sous forme numérique. Le format de présentation est: DDMONYY. DD sont deux chiffres pour le jour, MON sont les trois caractères initials (en anglais) du mois et YY les deux chiffres de l'année. Chaque tuple de PARAMETER contient un attribut du type DATE indiquant le jour de la visite. Un fichier optionnel OVERFLOW est employé si l'usager a besoin de maintenir des données non codés tel que des commentaires sur les décisions prises.

Une bibliothèque de programmes d'applications est disponible pour tous les usagers. Pour employer ces programmes il faut avoir fait la description du schema suivant certaines régles. L'usager a la possibilité d'employer celles qui peuvent lui être utiles. Un usager peut developper ses propres programmes ou copier et modifier ceux existant dans le système. La possibilité du partage des programmes, obtenue par la définition uniforme des données, a permis un developpement très rapide de nouvelles applications. L'utilité du système a été prouvé duns un environnement réel.

2.1.2 Le "time_expert" du sytème INGRES

L'addition, dans les bases de données, de nouveaux types de données, qui ont une sémantique propre peut être obtenue par la définition de "experts". Un expert peut être vu comme une procédure de controle et un ensemble de procédures spécialisées dans le traitement d'un type particulier de donnée. Cette methode a pour caracteristique principale l'independence entre le SGBD et les experts. Le SGBD n'a pas besoin de connaitre la façon de traiter le type de conaissance sémantique contenue dans un expert. Au lieu de faire l'extension du modèle de données on fait des programmes

experts qui sont facilement ajoutés au SGBD. Cette solution a été proposée et implantée comme une extension du système INGRES par M Stonebraker <SWKH_75>, <SK_80>, <OS_82> . La suite de cette section est constituée par la description de ce travail.

Un expert est associé à un attribut pendant la définition d'une relation Pour permetre cette définition la syntaxe de la commande CREATE a été modifiée de la façon suivante:

CREATE rel_name ((field_name=format!expert_name)...)

cette syntaxe étendue donne la possibilité de définir un attribut sur un nom d'expert en plus des types de base. L'effet de cette commande est la création d'une relation de nom "rel_name" et l'indication que l'attribut "field_name" est associé a l'expert de nom "expert_name".

La relation crée par la commande suivante sera employée pour illustrer l'action des experts:

CREATE EVENT (ename = char (20), time = time_expert)

La relation contient deux champs: "ename" défini comme une chaine de caractéres de longeur 20 et "time" associé à l'expert "time_expert". Un expert est consideré comme une procédure reconnue par le SGBD et qui peut traiter quatre variétés d'appels:

a) "expert_field operator value"

Une requête de ce type est:

```
RANGE OF E IS EVENT
REPLACE (E.time = "present_time")
WHERE E.name = "e1"
```

quand l'analyseur trouve le terme:

time = "present_time"

il fait appel au time_expert pour exécuter l'interpretation de la chaine "present_time" et produire une réprésentation interne pour cette valeur.Cette réprésentation est une structure de données associée à l'expert, structure connue egalement par le SGBD.

b) "value_1 comparaison_operator value_2"

pour ce type de requête la réponse sera une valeur qui appartient à l'ensemble:

(true,maybe,false)

Exemple:

```
RETRIEVE (E.name)
WHERE E.time = "12:00"
```

La valeur "maybe" peut apparaitre au cours de l'execution d'une comparaison entre intervalles.

c) "expert_field arithmetic_operator constant"

Exemple:

```
REPLACE (E.time = E.time + "1:30")
WHERE E.name = "lunch"
```

cette requête deplace l'heure du déjeuner d'une heure et demi.

d) "external_représentation <--- value"

ce type de requête produit une chaine de caractères avec la représentation externe (pour l'usager) de la valeur. Par exemple:

```
RETRIEVE (E.time)
WHERE E.name = "lunch"
```

Pour traiter complètemente cette requête le SGBD doit appeler l'expert qui effectuera la conversion de la valeur du constituant "time" en une forme externe accessible à l'usager.

Les formats externes acceptés par le time_expert sont :

1) dates simples - "simple dates"

le format d'une date a la forme:

(jour_semaine mois jour_mois heure minute secondannée)

jeudi novembre 17 14:55:30 1983

2) période de dates _ "range of dates"

l'expért accept des périodes pour lesquelles les extremités sont des dates simples. Par exemple:

jeudi 18 novembre 17 - vendredi 18 novembre 18:30

3) période relatif de temps - "relative range of time"

il est possible de définir en plus des périodes absolues des périodes relatives au instant présente. Par exemple: "aujourd'hui", "demain", "l'année derniere" et "à 9 semaines d'aujourd'hui" sont des

constructions acceptées.

4) intervalles de temps - "intervalles of time"

exemples: "7 jours", "4 années", "99 minutes".

5) intervalles répétitifs - "repetitive intervalles"

exemples: "lundi", "vendredi", "12:00 - 13:33".

Une grammaire complète des définitions acceptées au départ par le time_expert est fournie dans l'annexe 1 de <OS_82> .

Pour permettre la définition de nouveaux concepts l'expert est aidé par une relation appellée TIME_EXPERT qui a la structure suivante:

TIME_EXPERT(EXTNAME_1, INTCODE, EXTNAME_2, OP)

Le champ INTCODE fait la transformation entre la codification interne et la représentation de EXTNAME_1. Par exemple: l'expert produit le code interne pour le litteral "aujourd'hui" à partir de la lecture de l'horloge du système et en modifiant cette représentationà partir de l'information stockée dans INTCODE. Une des valeurs de EXTNAME_1 est employée pour la représentation de la configuration d'un jour de la semaine. De cette façon un seul tuple est nécessaire pour la codification de tous les jours. La même technique est employée pour la codification des années, des mois etc. Ce methode est similaire au emploi des masques binaires pour la selection d'une partie d'un opérand par le fonction "AND" dans des programmes écrits en assembler. EXTNAME_2 a pour but la codification des références indirectes, par exemple: "hier" est représenté par:

(hier, 'code pour 1 jour', aujourd'hui, '-')

On peut profiter de l'existence de cette relation pour étendre la grammaire de base de l'expert.

2.1.3 L'extension du modèle de données, TERM

Dans la plupart des systèmes de base de données les modifications sont faites en changeant les anciennes valeurs par des nouvelles. Les anciennes sont perdues à l'exeption de l'apparition dans le journal. Les requêtes font toujours référence au présent. MR Klopprogge <Klo_81> a traité ces deux aspects de temps: la gestion des anciennes valeurs et les requêtes historiques. Les bases de données historiques sont réalisées par l'extension du modèle "Entity-Relationship", <Che_76>, et une proposition est faite pour déduire les états du passé qui n'ont pas été stockés dn'une façon explicite.

Dans TERM, <Klo_81>, <KL_83>, une entité ou une association sont décrites par une assertion d'existence et par des attributs et des rôles. Ces deux derniers peuvent être constants ou variables par rapport au temps. S'ils sont constants une valeur leur sera associée, s'ils sont variables nous leur associerons un historique. Dans le modèle sont définies trois structures: une "time structure" pour la représentation de temps, une "value structure" pour la représentation des valeurs des attributs et/ou rôles et une "history structure" pour l'ensemble des valeurs passé. Un historique est vu comme le "power set" du produit cartésien des ensembles des valeurs d'une "value structure" et d'une "time structure". La représentation d'un historique est un ensemble de tuples du type (valeur,temps) qui sont appelées les états de l'historique.

Un problème qui se pose pour la représentation des historiques est la possibilité de l'existence d'un ensemble infini d'états. Dans ce cas on a besoin de faire la représentation de l'historique au moyen d'un ensemble fini d'états caractéristiques et de fournir la fonction de dérivation pour les états non définis explicitement. Parfois il est impossible de specifier une fonction de dérivation , dans ce cas des functions d'approximation telle que la méthode des moindres carées peut être employée. Ces algorithmes sont spécifiques à une structure d'historique et sont appelés opérateurs d'induction.

Les opérations de base sont de deux types: enregistrement et correction. Pour les enregistrements existent deux opérations: l'initiation et la conclusion. La première a comme resultat la création d'une entité où association et la deuxième indique la fin d'existence. L'opération de correction a pour but de permettre au responsable la correction d'eventuelles données erronées.

Les concepts presentés dans <Klo-81> sont repris et étendus par la notion d'incertitude. Les moyens pour limiter ce problème dans un ensemble de situations bien définies se trouvent dans l'article <KL-83>. Il y a aussi un exemple complet d'une application bancaire décrite avec l'aide de TERM.

2.2 L'extension du modèle TIGRE

Notre objectif est de décrire ici une proposition d'extension pour la gestion de temps dans le modèle de données TIGRE. Le but de l'extension est double. Premièremente nous voulons traiter le problème sous l'angle conceptuel et dernièrement nous voulons aboutir à une réalisation dans le cadre du prototype du serveur base de données TIGRE. A partir de l'étude des concepts et des réalisations décrits dans la section 2.1 nous avons choisi l'extension du modèle de données par inclusion du type temps. Nous allons faire la description de cette extension et ensuite nous indiquerons les nouvelles possibilités apportées au langage de manipulation par l'inclusion du concept de temps.

2.2.1 Types associés au temps

Pour la représentation de temps il est possible de reprendre la vision de la physique. Le temps est vu comme une variable continue, isomorphe aux réels, avec une unité associée, par exemple la seconde dans la système CGS. N'importe quelle grandeur peut être décrite par rapport à l'unité. Une semaine , par exemple, est représéntée par 604.800 secondes. Cette dernière représentation a très peu d'intêret car dans la manipulation des modèles des systèmes physiques il est indispensable de maintenir l'homogeneité dimensionelle. Dans les formules on doit employer une seule unité pour la représentation du temps. Pour adopter cette solution il n'est pas nécessaire de faire des modifications dans le SGBD. L'existence des réels suffit pour la modèlisation. La semantique du temps est equivalent à celle des réels.

Les problèmes de gestion ont des besoins très différents. Pour cette classe d'applications on emploie généralement un calendrier. Un calendrier peut être décrit comme une

association entre des périodes et la représentation de temps. Par exemple, dans les actes de la vie civile on emploie le calendrier Grégorien. Dans ce cas chaque activité est associée à une date. Une date est une référence à un instant de temps.

Exemple:

"La vignette doit être payée jusqu'au 1er décembre 1983.".

L'interpretation de cette phrase est: l'activité de payer l'impôt sur les voitures doit être effectuée jusqu'à l'instant de temps représenté par la date "1er décembre 1983" du calendrier Grégorien.

Un système de calendrier suppose une vision discrète de temps en contraste avec la vision continue. Dans l'exemple, le niveau de précision est le jour. Nous avons choisi comme niveau de base une précision de l'ordre de la seconde. De plus il est possible de restreindre le calendrier pour travailler avec un niveau moins fin de précision.

Les types de base simples.

La représentation de temps dans le modèle TIGRE est faite par un nouveau type de base: le temps. La règle de grammaire pour la définition des types de base simples est etendue par:

```
<basic_simple_type> ::= 'integer' ! 'real' ! 'boolean' !
            'string' '(' <unsigned_integer> ')' ! 'time'
```

Avec cette extension il est possible de faire la définition d'un attribut sur le domaine temps:

```
type T_EMPLOYE : Entity
        name : string (20);
        birth_date: time
        end;
```

Le type "time" est défini par les périodes du calendrier Grégorien auxquelles nous ajoutons l'heure, sous la forme d'heures, minutes et secondes. La définition de <time> est donnée par:

<time> ::= < year> '/' <month> '/' <day> ' ' <hour> ':' <minute> ':' <second> ! 'present_time'

<year>, <month>, <day>, <hour>, <minute>, <second> ::= <unsigned_integer>

Les représentations externes de temps sont des chaines de caractères de longeur 19, telle que la suivante:

'1983/11/28 15:12:20'

Une instance d'un attribut défini sur le type temps est appelée une date et a la structure décrite par la syntaxe de <time>. Le mot "present_time" fait appel à une procédure qui retourne la chaine de caractères correspondante a l'instant présent.

Notre définition de calendrier est equivalente à celle employée par la langage "Conceptual Schema Langaget - CSL", <BFM_79>, et repriset ensuite par le "Temporal Hierarchic data Model - THM" <Sch_83>. Le modèle de données TERM, <KL_83>, emploi une représentation de temps similaire.

Dans cette classe de modélisation un système de calendrier est materialisé par une serie ordonnée d'intervalles, six dans notre modèle:

(I6,I5,I4,I3,I2,I1)

ou I1 ... I6 sont des entiers. A chaque intervalle est associé un nom:

(I6:année, I5:mois, I4:jour, I3:heure, I2:minute, I1:seconde)

Les valeurs des intervalles successifs satisfont les contraintes d'integrité particulières au calendrier. Pour le calendrier Grégorien, étendu jusqu'à la seconde, ces contraintes sont:

(second >= 0) AND (second < 60) AND

(minute >= 0) AND (minute < 60) AND

(heure >= 0) AND (heure < 24) AND

(jour >= 1) AND (jour <= 31) AND

((jour <> 31) OR (mois in (1, 3, 5, 7, 8, 10, 12)) AND

((jour <> 30) OR (mois <> 2)) AND

((jour <> 29) OR (mois <> 2) OR ((année MOD 4 <> 0) AND

((année MOD 100 <> 0) OR (année MOD 400 = 0)))) AND

(mois >= 1) AND (mois <= 12) AND

(année >= 1582)

Les types de base restreints et les types renommés.

Le niveau le plus bas, ici la seconde, est dit la granularité du système de temps. On voit bien qu'un point de temps, dans ce type de modélisation, est vraiment un intervalle. Differentes applications peuvent avoir besoin de granularités

différents de la seconde. Pour celles-ci on permet la définition de types restreints particuliers au temps. Ce types sont des restrictions sur l'existence d'intervalles moins significatifs. Ainsi il est possible de redéfinir des calendriers avec une granularité plus grande. Par exemple:

type t1 : time > hour;

cette declaration specifie un calendrier du type:

(année, mois, jour)

Les contraintes d'integrité prédéfinies pour le calendrier de base et applicables aux nouveaux calendriers sont maintenues. La syntaxe pour cette restriction est:

<time_rest_type> ::= 'time' '>' ('month' | 'day' | 'hour' |

'minute' | 'seconde')

Les opérateur de comparaison ne sont définis que sur des opérandes de même précision. Cette limitation empèche l'apparition d'une valeur indeterminée comme résultat d'une comparaison. L'usager peut employer les fonctions T_TRUNC et T_ROUND pour changer la précision d'un opérand, cettes fonctions sont similaires à TRUNC et ROUND définies pour les réels, nous reviendrons sur ce sujet dans la section 2.2.3 .

Un autre type restreint qui peut être défini sur le type temps est l'intervalle:

<time_interval> ::= '(' <time> '..' <time> ')'

L'intevalle de temps est défini comme une période entre deux dates. Par définition les deux extremes appartient à

l'intervalle. Un type intervalle peut avoir une des extremités définie comme 'present_time'.

Il est aussi possible de définir des types renommés sur un type de base simple où restreint. Par exemple:

type inscrp_int : (1983/10/10 8:00:00 .. 1983/12/9 17:00:00)

Les types périodiques.

La représentation des activités périodiques est une des plus importantes nécessités des systèmes de gestion. Quelques exemples d'activités périodiques sont: l'émission de relevés de comptes de chèques à chaque 1er jour du mois, l'émision des avis de paiement d'impôt chaque mois de septembre, l'impression du solde de chaque compte chèque à 18 heures ou l'affectation de l'horaire de 12:00-14:00 heures pour le dejeuner. Afin de permettre la réalisation de ce type de modélisation est défini le type périodique.

Exemples:

type t1 : month := 12 ;

t2 : (day := 10 .. day := 20) ;

La signification de chaque type périodique défini ci-dessus est respectivement: la définition de tous les mois de décembre et la définition d'une période entre le 10 et le 20 de chaque mois.

La syntaxe du type périodique est la suivante:

<periodic_time_type> ::= <periodic_date> ! <week_day> ! <periodic_interval>

<periodic_time> ::= ('year' | 'month' | 'day' | 'hour' | 'minute') '>' <time_rest_type> | <time_element>

<week_day> ::= 'su' | 'mo' | 'tu' | 'we' | 'th' | 'fr' | 'sa'

<periodic_interval> ::= '(' <periodic_date> '..' <periodic_date> ')'

<periodic_date> ::= <periodic_time> ':=' <time_value>

<time_value> ::= /* instance of <periodic_time> */

<time_element> ::= 'year' | 'month' | 'day' | 'hour' | 'minute' | 'seconde'

Les constantes de type temps.

Pour faciliter la définition du schéma et l'emploi de la langage de manipulation existe la possibilité de définir des constantes de type temps:

<time_const> ::= 'time_constant' <one_time_const_def> (';' <one_time_const_def>)*

<one_time_const_def> ::= <time_const_id> '=' <unsigned_integer> <time_element>

<time_const_id> ::= <identifier>

Exemples:

time_constant quinzaine = 15 days ;

semestre = 6 months ;

septennant = 7 years ;

2.2.2 Historiques

En plus des attributs définis sur les types de base simples ou restreints et sur les types renommés il existe une autre façon pour laquelle le temps intervient dans la modèlisation : la manipulation des anciennes valeurs d'attributs ou des faits appartenant à la base de données. Ce nouveau concept correspond à la définition d'historiques.

Pour faire la modélisation du comportement des valeurs contenues dans la base par rapport au temps nous avons fait une extension de la définition des types. Avec cette extension nous avons une description uniforme au niveau des attributs, des entités et des associations. La définition d'un type est étendue de manière à faire apparaitre deux parties composantes: la structure de données et la persistance. La structure de données correspond à la définition statique du type. La persistance définit le comportment des valeurs du type par rapport au temps. La persistance d'un type peut être statique ou dynamique. Un type statique est tel que la seule valeur disponible dans la base de données est la valeur présent. Une partie "persistance" vide indique un type statique. Un type dynamique mantient, dans la base de données, une sequence des valeurs anterieures: l'historique.

Exemples:

```
type revenus : dynamic
               each month
               last 12
               real ;
```

Le type "revenus" est défini comme un type dynamique dont on veut maintenir les valeurs avec une périodicité mensuelle. Les 12 valeurs les plus récentes sont maintenues. La structure du type est un réel. Chaque valeur de "revenus" est associée à la valeur de "present_time" du mois de la modification.

```
type echange : entity
               monnaie : string (12) ;
               taux : dynamic real
               end ;
```

Le type entité "echange" a l'attribut "taux" défini sur un domaine dynamique avec la structure "real". A cet attribut est associée une liste de longueur illimitée. Chaque noeud de cette liste a la structure: (temps,réel) ou "temps" est le "present_time" pris au moment de l'introduction dans la base de données du "réel".

```
type t_envoi : relationship
               dynamic each year
               between t_fournisseur and t_article
               quantité : integer
               end ;
```

Le type association "t_envoi" est défini comme dynamique et le système maintient une photographie ('snapshot') de chaque année.

La syntaxe de <one_type_def> étendue est:

<one_type_def> ::= <type_id> ':' <persistency_def> <structure_def>

<structure_def> ::= <basic_type> | <constructed_type> |

<class_type>

<persistency_def> ::= 'dynamic' /'each' (<periodic_time_type>
! <unsigned_integer>)/ /'last' <unsigned_integer>/
! <empty>

La clause "each" permet de définir la périodicité de l'echantillonage. A chaque période ou après le nombre de modifications spécifié une valeur est maintenue dans la base de données. La clause "last" indique combien de valeurs il faut maintenir, elle décrit la longeur de la liste de valeurs.

Un type de base peut toujours avoir une persistence dynamique. Si une entité ou une association ont des attributs définis sur des types dynamiques elles contient l'ensemble des historiques des attributs. Le même est vrai pour les types construits. Pour qui soit possible définir un type classe ou un type construit avec une persistence dynamique il faut, obligatoiremente, que les types constitutifs soient statiques. La raison pour cette restriction est le fait que il n'y a pas de sens définir un historique composé par des historiques. A chaque valeur de la classe ou du type construit maintenue dans la base de données tout un nouveau historique des types constitutifs dynamiques seraient inclus. Dans cette situation les valeurs anterieures à la dernière version déjà stockée seront repliées. La décision sur quels types doivent être dynamiques est une choix de la modélisation de la realité. Chaque application a des besoins qui doivent être pris en compte par l'usager du modéle de données.

Chapitre 3

LA MODELISATION DE LA REALITE

3. La modélisation de la réalité

Une des plus complexes activités dans le developpement des systèmes d'information est la modélisation de la réalité. A partir de la réalité notre perception permet d'avoir une certaine vision du système réel: cette vision est appelée la réalité perceptible. Entre cette réalité perceptible et la description de la base de données par le schéma conceptuel, nous devons executer une séquence d'activités qui est appelée analyse de systèmes. Dans ce cadre, trois activités sont bien distinctes: <Nij_76> :

- dénomination
- selection
- classification

La "dénomination" consiste à identifier par un nom unique chaque objet perceptible. La "selection" permet de reduire la réalité perceptible à un sous ensemble gérable. Cette activité est bien etudiée dans la methodologie des sciences physiques ou, en informatique, par la simulation. On l'appelle aussi l'abstraction. La "selection" a comme conséquence la définition de univers de discours ou ensemble des objets referencés. Finalement, la "classification", permet d'arriver à une description, grâce au modèle de données, de l'univers de discours, ce processus correspond à la définition du schéma.

L'activité de "denomination" correspond à la déscription informelle de l'application. Cette activité et la suivante, la "selection", doivent être menées par des personnes qui connaissent la sémantique de l'application. Cette partie de l'activité d'analyse prend en compte des grands volumes d'information. Elle prèsente, aussi, des problèmes de

communication entre les spécialistes en informatique et les usagers du système d'information. Il est tout à fait envisageable d'avoir un logiciel d'aide a l'analyse pour le bon déroulement de cette phase.

La "classification" consiste à faire le groupement en un certain nombre de classes des objets nommés et selectionés. Le modèle de données est un support pour cette activité car il permet l'emploi d'un ensemble de caractéristiques qui sont intrinsèques au modèle. De cette façon une certaine partie de la classification est déjà incluse dans la déscription du modèle de données et l'activité de spécifier le schéma est très simplifiée. Ces caractéristiques du modèle de données sont les contraintes d'integrité structurelles <Bro_78>.

Pour céer la description du schéma il existe le langage de définition de données, dans notre système, il s'agit de Lambda. Il établie la correspondance entre la réalité et le modèle de données.

Dans ce chapitre nous décrirons une méthodologie qui facilite le passage de l'univers de discours au schéma conceptuel. Cette methodologie sera illustrée par un exemple. Notre objetif a été d'établir une stratégie générale pour capturer la sémantique d'un problème de taille significative. Dans la suite de notre recherche nous envisageons de developper un système d'aide à l'analyse structurée de façon à intégrer la stratégie de définition proposée ici.

Dans le processus de création d'un schéma il est important de modéliser d'une façon naturelle la sémantique de l'application au moyen des caractéristiques structurelles du modèle de données. Différentes methodologies sont décrites dans <Che_76>, <SS_77>, <Bro_78>, <VN_82>, <ZM_82>, <Sak_83>, entre autres.

Les modèles de données avec une sémantique étendue permetent de répresenter une plus grande partie de la sémantique de la réalité. En contrepartie l'activité de classification est plus complexe car les possibilités de représentation sont plus nombreuses. Cette difficulté conduit presque nécessairement au developemment de méthodologies pour qu'il soit possible de maîtriser le processus de modélisation.

Modélisation: une "Working Conference IFIP"

La methodologie de modéllisation sera décrite et illstrée par l'analyse d'un problème. Le problème choisi a été la modélisation d'une "Working Conference IFIP" prèsentée dans <IFIP_82>. Le choix a été guidé par les éléments suivants:

- Un exemple avec des aspects bureautiques
- La possibilité de l'inclusion du concept de temps
- Un problème bien connu
- Une application de taille significative
- Une définition ouverte permettant de futures extensions

Nous essaierons d'employer un seul exemple pour chaque point important en évitant les répétitions et les détails qui n'apportent pas plus clarté au problème. Dans l'annexe A3 sont décrits l'énoncé complet du problème et le schéma de la base de données.

3.1 Les classes d'entités

3.1.1 Identification et dénomination

Dans le modéle ER, <Che_76>, deux types d'objets sont définis: les entités et les associations. Les objets qui ont une existence indépendante et qui peuvent avoir une signification propre sont considerés comme des entités et sont réunis dans des classes. Par exemple une personne est une entité qui appartient à la classe d'entités Personne. Une classe d'entités est décrite par ses propriétés. La classe Personne est décrite par : numéro IFIP, nom, addresse et invitation à la conférence. Dans cette interpretation de l'univers de discours l'adresse a été considerée comme une propriété de Personne. L'adresse pourrait être consideré, dans une autre interprétation, comme une entité indépendante et l'indication qu'une personne habite à un certain endroit serait faite par l'association entre Personne et Adresse. Pour définir cette association il faudrait créer la classe d'associations "Résidence".

La modélisation de l'univers de discours par un schéma est une décision relative et depend des besoins de l'application. Dans le chapitre "Data Models" de <TL_77> nous trouvons une introduction à la modélisation avec l'aide des modéles de données réseau et relationnel. Le problème de savoir qu'est ce qui est un objet et quels objets peuvent être représentés par des entités ou par des associations est traités par <Ken_77>.

Une classe d'entités non dérivées est une classe qui peut être définie d'une façon independante des autres classes d'entités contenues dans l'univers de discours. Une entité non dérivée correspond à une "inner kernel entity" de <Cod_79>. Chaque occurrence d'une entité appartient à une seule classe d'entités non dérivée, cela veut dire que les classes

d'entitéss non dérivées sont disjointes.

Pour qu'une entité appartienne à une classe, elle doit satisfaire un prédicat. Ce prédicat est le critère de classification qui permet passer de l'univers de discours au schéma conceptuel. Dans notre exemple nous avons choisi:

Personne, Comité_technique, Groupe_de_travail, Conference_de_travail, Session, Article

comme les classes entités non dérivées. Chaque classe a un critère de classification bien défini par lequel il est possible de savoir si une entité appartient ou non à cette classe.

Dans l'ensemble des objets qui appartient à l'univers de discours il existe des entités qui peuvent être décrites d'une façon plus naturelle comme ayant une existence dépendante d'autres entités. Ces entités correspondent à des "subtypes" dans RM/T, <Cod_79>, et au concept de généralisation dans <SS_77>. Par les opérateurs de généralisation, du modèle de données Tigre, nous formons ce qui est appellée une hiérarchié de généralisations, <TIGRE-2>. Dans une hiérarchié de généralisations le sommet est une classe non dérivée. L'opérateur de spécialisation a une fonction inverse de celui du concept de généralisation, il permet de passer d'une classe plus générale à une classe restreinte. Les opérateurs d'union et d'intersection ont un rôle d'agrégation car à partir de classes particulières ils créent une classe plus générale.

Exemple, la spécialisation:

Nous pouvons définir

Invité, Rep_national, Referee, Probable_auteur,

Président, Auteur

comme des spécialisations de Personne.

La caractéristique des entités dérivées par spécialisation est d'avoir des proprietés communes qui peuvent être représentées par une seule entité de niveau supérieur. La classification des entités en non dérivées et dérivées par spécialisation doit être faite en parallèle à la description des classes d'entités.

3.1.2 Description des classes entités

Le processus de modélisation des classes d'entités peut être decomposé en plusieurs étapes:

a) identification et dénomination des propriétés immédiates des classes d'entités. Dans cette étape nous identifions quelles propriétés pouront être représentées comme attributs. D'autres propriétés devront être représentées par d'autres moyens, par exemple, comme contraintes de cardinalité ou comme contraintes d'integrité plus générales.

b) pour chaque classe d'entités, trouver quelles sont les propriétés communes qui peuvent donner lieu à la création d'une classe d'entités spécialisée. Créer cette classe spécialisée.

c) pour chaque classe d'entités qui a comme propriété d'être composée par d'autres entités créer une classe d'entités composée par agrégation d'entités. Faire la spécification des contraintes de cardinalité pour chaque classe d'entités qui appartient à l'agrégation.

d) trouver quelles classes d'entités peuvent être associées pour répresenter une nouvelle classe. Définir les classes dérivées par union et intersection correspondantes.

Pour toutes les classes nouvelles définies il faut procéder a une répétition des activités de "a" jusqu'à "d". A la fin de ce processus toutes les classes d'entités non spécialisées et spécialisées ainsi que les classes composées par agrégation d'entités auront été crées. Après la conclusion de ce niveau de définition il est possible de définir la structure des attributs.

e) identifier les attributs définis sur les types simples et les attributs définis sur des types construits.

f) parmi les attributs définis sur des types simples identifier ceux qui appartiennent aux clés des entités, si elles existent.

g) dans l'ensemble des attributs définis sur des types simples trouver quelles contraintes d'integrité peuvent être représentées par des types restreints ou des constantes. Définir les constants nécessaires.

h) dans l'ensemble des attributs définis sur des types simples de base ou restreints trouver quels doivent être ceux définis sur des types renommés. L'emploi des types renommés a pour but d'accroitre la sémantique modélisée. Créer les types renommés nécessaires.

i) pour chaque attribut défini sur un type construit

faire la description du type construit. Cette activité est composée par l'execution , repetée pour chaqcun des types construits, des items "a" et de "e" a "i".

Les différentes étapes de définition décrites sont illustrées en suite:

a) Pour Personne: les propriétés de cette classe d'entités sont répresentées par les attributs suivants

   - IFIP_n: est le numéro de la personne dans le système de gestion de l'IFIP

   - nom: est le nom de la personne

   - adresse: est l'adresse de la personne

b) Pour Rep_national, Referee, Probable_auteur et Président: toutes ces classes d'entités ont des propriétés en commun qui sont généralisées par l'entité Personne. Ces quatre classes sont définies commes des spécialisations de Personne. Comme l'inclusion d'une entité dans chaque classe spécialisée sera faite par une décision des Comités, elles ont la condition d'inclusion "manual". La classe Personne a l'attribut "invitation", pour signaler que cette personne a été invitée. Pour chaque occurrence personne qui a la valeur de l'attribut "invitation" égal à TRUE une occurrence est incluse dans la classe Invité.

   Pour Invité: la spécialisation dans ce cas est une conséquence de la réception d'une réponse positive

d'acceptation de l'invitation. La condition d'inclusion dans la classe spécialisée Invité est définie par le prédicat "réponse = TRUE".

c) Pour Groupe_de_travail et Proceedings: créer les classes d'entités dérivées par agrégation d'entités correspondantes. La première est une agrégation d'entités qui appartient à la classe Personne et la seconde d'entités qui appartiennent à la classe Articles_par_session.

Pour Groupe_de_travail: un groupe doit être composé par plus d'une personne. Une contrainte de cardinalité est définie comme (2,*).

e) Pour Personne: "IFIP_n" et "nom" sont définis sur des types simples et "adresse" est défini sur un type construit enregistrement.

f) Pour Personne: l'attribut "IFIP_n" est unique dans le système de gestion et assure l'identification unique des membres, il a été choisi comme la clé.

g) Pour Conférence_de_travail: la durée de la conférence peut être indiquée par l'attribut "période" défini sur le type restreint intervalle de temps.

i) Pour Personne: l'attribut "adresse" n'est pas atomique il doit être représenté par le type construit "t_adresse".

```
type t_adresse : record

        n           : integer ;
        rue         : string (36) ;
        code_postal : integer ;
        ville       : string (20) ;
        pays        : string (36)
    end ;
```

Pour la définition du type construit enregistrement nous avons efectué les activités "a" et de "e" à "i". Cette procédure peut être executée plusieurs fois car un type construit peut être composé par autres types construits. Pendant cette execution l'activité "g" a donnée lieu à la modification suivante:

h) Pour le type t_adresse et pour Rep_national: les deux contiennent l'attribut "pays" et dans l'application nous voudrons envoyer, au répresentant national, une liste de tous les articles reçus, et ensuite une autre liste des articles acceptés. Nous avons choisi définir l'attribut "pays" sur le type renommé "nom_de_pays".

3.2 Les classes d'association

Une fois faite la déscription des classes d'entités, la hierarchie de classes dérivées, les classes composées par union et intersection et par agrégation d'entités il faut répresenter les relations entre ces classes. Les relations entre classes d'entités sont représentées par les classes d'association. Pour qu'une association existe il est obligatoire que les deux entités mises en relation existent aussi (le modéle Tigre ne supporte que des associations binaires). Aussi pour qu'il soit possible de définir une

classe d'associations les deux classes d'entités relationnées doivent être déjà definies.

Une classe d'association peut avoir des propriétés particulières. Par exemple, la classe d'association Attribution a comme propriétés: la date d'attribution, la date d'évaluation, quel a été l'évaluation et un commentaire.

En plus des entités en relation et des eventuelles propriétés, il faut définir le rôle que chaque entité joue dans l'association et ses contraines de cardinalité.

3.2.1 Description des classes d'association

L'identification, la dénomination et la déscription des classes d'association se font de la manière suivante:

a) identification de l'association. La relation entre les entités associées est identifiée et un nom est donné à la classe d'association.

b) nommer les rôles. Si le nom de la classe entité associée est significatif nous n'avons pas besoin de créer des nouveaux noms pour les rôles. En absence des noms des rôles ils seront identiques aux noms des entités.

c) définir les contraites de cardinalité sur les entités qui appartiennent à l'association.

d) pour chaque association trouvée identifier et nommer les propriétés. Pour chaque propriété executer les actions de "e" à "i" comme definies pour les entités.

e) trouver quelles associations peuvent être spécialisées. Pour chacune définir le nom de la classe spécialisée, les eventuelles contraintes de cardinalité et les propriétés particuliéres.

Exemples:

a) une association est identifiée entre les referees et les articles. Elle est désignée par le nom Attribution.

b) les noms des entités associées sont significatifs pour Attribution. Nous ne définissons pas les noms de rôles. Par contre pour la classe d'associations Composition, entre Comité_technique et Personne, nous avons choisi d'appeler le rôle joué par Personne dans l'association de "membre".

c) pour Organisation: une conférece de travail doit être proposée par au moins un Groupe_de_travail. La contrainte est définie comme (1,*).

d) pour Attribution: l'execution des activités de "e" à "i" définies comme pour les entités donne:

```
type Attribution : relationship
        between Referee
        and Article

        date_att     : jour ;
        evaluation   : boolean ;
        commentaire  : document
                          structure
                              comm : text
                       end
```

end ;

3.3 L'agrégation associative

L'opérateur d'agrégation associative permet de voir une classe d'associations et des classes d'entités liées comme une entité agrégée. Cet opérateur produit une classe d'entités à partir d'une classe d'associations et des classes d'entités associées. La classe d'entités resultant de l'opération d'agrégation peut avoir ses propres propriétés.

Dans la modélisation, l'agrégation associative permet une relativité de représentation car nous pouvons redéfinir une classe d'associations comme une classe d'entités. Cela veut dire qu'à un certain moment il est possible d'employer l'association comme une agrégation formant un nouvel objet. Ce emploi est analogue au "produit" défini en <CNF_80>. L'autre emploi permet de faire des asociations entre des asociations, comme comme en <SSW-80>. Dans ce cas l'agrégation permet d'abstraire la classe d'associations comme une entité qui peut être associée à d'autres entités ou être le composant d'une agrégation d'entités.

Dans l'exemple, la classe Art_sess est agrégée dans la classe d'entités Articles_par_session. Cette classe sera ultérieurement employée dans la construction de la classe Proceedings.

3.4 Les actions et le schéma

Les activités du Comité de Programme (CP-1 a CP-6) et celles du Comité d'Organisation (CO-1 a CO-7) sont liées à la manipulation des entités et des associations définies dans le schéma. Ensuite nous ferons une description du rapport entre les activités et les différents objets du schéma.

CP-1. Préparation de la liste de personnes pour lesquelles l'appel aux communications doit être envoyé.

Le Comité de Programme doit rediger l'appel aux communications et selectioner les personnes à qui il doit être envoyé. L'appel est modélisé par la classe d'entités Appel_aux_communications. Cette classe a deux attributs: la version et le contenu. La version est reprèsentée par l'attribut "version" défini sur un type scalaire qui a pour valeurs: préliminaire, définitive et rappel. A chaque valeur de cet attribut est associé un texte. L'envoi d'un appel correspond à l'insertion d'une occurrence dans la classe Envoi. Comme le texte de l'appel et les informations sur le destinataire sont stockés dans la base de données, l'émmision peut être faite par le système. La selection des personnes à qui l'appel préliminaire doit être envoyé est faite par le Comité. A chaque nouvelle version de l'appel un nouveau envoi est fait pour toutes les personnes déjà associées par la classe Envoi. En mêmme temps il fait la mise à jour de la date d'envoi.

CP-2. Enregistrement des lettres d'intention reçues en réponse aux appels aux communications.

Des personnes autres que celles qui ont reçu l'appel peuvent envoyer une lettre d'intention. La réception des lettres d'intention est par consequent indépendant de la récption d'un appel par le probable auteur. La réception d'une lettre d'intention entraine l'inclusion d'une occurrence dans la classe Probable_auteur. Si dans Personne il y a déjà une description de cette personne il y aura seulement la création d'une occurrence dans la

classe spécialisée Probable_auteur. Dans le cas contraire, il faudra inserer une occurrence dans la classe Personne et ensuite faire sa spécialisation dans la classe Probable_auteur.

CP-3. Enregistrement des articles à leur réception.

A la réception d'un article il recoit un numéro pour faciliter sa manipulation en dehors du système informatique et une occurrence est inserée dans la classe Article. Un article peut avoir plusieurs auteurs, lors de sa réception les co-auteurs sont inclus dans la classe Auteur et, s'il faut, dans la classe Personne. Ensuite est crée l'association Autorship entre Article et Auteur. Cette association a comme attribut "auteur_no", l'ordre du nom du co-auteur dans la liste des auteurs.

CP-4. Distribuition des articles aux referees.

Cette activité doit être décomposée en deux autres: la selection des referees et l'attribuition des articles aux referees. La selection a comme résultat l'insertion de occurrences dans la classe spécialisée Referee. Une personne spécialisée dans Referee est un referee potentiel, quelqu'un qui a accepté de collaborer au processus de selection des articles, si necessaire. La distribuition des articles aux referees est modélisée par des occurrences de la classe Attribution. Les contraintes de cardinalité sont: un referee peut recevoir au plus cinq articles et un article doit être auvoié à au moins deux referees et au plus à trois. La date d'envoi des articles au referee est indiquée par l'attribut "date_att".

CP-5. Reception des rapports des referees et sélection des articles pour inclusion dans le programme.

La reception de chaque rapport des referees produit la mise à jour des attributs "date_eval", "evaluation" et "commentaire" de l'occurrence appropriée de l'association Attribuition. Après la réception de tous les rapports le Comité de Progamme peut sélectionner les articles pour l'inclusion dans le programme. Les articles acceptés auront l'attribut "decision" égal à TRUE. Ces articles sont spécialisés dans Article_accepté. La classe spécialisée Article_accepté a un attribut défini sur le type construit document, de cette façon les textes révisés des articles seront saisis et l'édition des proceédings sera faite sur un original généré par l'éditeur de textes du système.

CP-6. Groupement des articles sélectionés en sessions et choix du président de chaque session.

La première activité est la définition des sessions. Pour chaque session une occurrence est inserée dans la classe Session. Ensuite le Comité de Programme doit faire la selection des présidents. Les personnes qui pourraient présider une session sont contactées et en cas de réponse positive de leur part, elles sont inclues dans la classe spécialisée Président. Chaque occurrence de Personne spécialisée en Président est un président potentiel. La decision d'attribuer la présidence d'une session à une personne est reprèsentée par l'association d'un président à une session via Présidence.

Les articles acceptés sont classés par sessions, à

chaque article est attribué un numéro d'ordre dans la session et l'heure de début de sa prèsentation. L'association Art_sess est agrégée dans l'entité Articles_par_session qui sera employée pour la définition des Proceedings.

CO-1. Préparation de la liste des personnes qui doivent recevoir une invitation pour participer à la Conférence de Travail.

Le changement de la valeur de l'attribut "invité" à TRUE produit l'inclusion d'une occurrence, spécialisation de Personne, dans la classe Invité.

CO-2. Envoyer une invitation prioritaire aux répresentants nationaux, membres du groupe de travail et des grouupes associés.

A partir du groupe de travail qui organise la Conférence trouver le comité téchnique à qui le groupe appartient. Une fois selectioné le comité il faut trouver l'ensemble des groupes de travail associés. Ces requêtes sont décrites au moyen de l'association Appartenance. Une foi selectionés tous les groupes du Comité Téchnique il faut selectioner les personnes intégrantes de ces groupes et faire l'union avec les reprèsentants nationaux. Pour chaque composante de l'ensemble resultant verifier si l'attribut "invité" à la valeur TRUE. Dans le cas contraire changer la valeur pour TRUE. Changer la valeur de l'attribut "priorité" pour TRUE.

CO-3. Assurer que chaque auteur d'un article accepté soit invité.

CO-4. Assurer que chaque auteur d'un article rejeté soit invité.

Cettes deux activités sont executées ensemble. Pour cela il suffit de verifier si pour chaque auteur l'attribut de Personne, "invité" a la valeur TRUE, le cas echeant la changer. Comme Auteur est une classe spécialisée de Personne elle hérite de tous les attributs de la classe supérieure.

CO-5. Eviter l'émission d'invitations doubles pour une personne

Par la structure du schéma, les invités possibles sont des spécialisations de la classe Personne, il y a donc une impossibilite d'invitations doubles.

CO-6. Production de la liste finale de participants.

Pour obtenir la liste finale de participants nous devons classer les invités, qui ont accepté de venir, en deux groupes de priorité. Dans chaque groupe la classification est faite par ordre croissante de la valeur de l'attribut "date_rec". A partir du groupe prioritaire faire l'incusion dans la classe spécialisée Participant des invités par ordre de classification jusqu'à arriver au nombre maximun de participants.

Chapitre 4

LE SERVEUR BASE DE DONNEES

4. Le serveur Base de Données

Ce chapitre sera consacré à la présentation des différentes architectures pour la réalisation d'un SGBD traditionnel et les solutions possibles pour l'extension du système à des nouveaux types de données généralisées. Ensuite nous ferons la description de l'architecture choisie pour le serveur TIGRE et du SGBD relationnel sous-jacent.

Le développement d'un SGBD basé sur un modèle de données avec une sémantique étendue et capable de fournir un interface d'usager de haut niveau est un problème de grande complexité. Pour maîtriser cette complexité la solution la plus acceptée est la définition d'une architecture construite en couches spécialisées. De cette façon les différents problèmes sont isolés et peuvent être résolus indépendament.

L'adition de types de données complexes tels que les documents, les formulaires, les tableaux et les images, rend le problème de la spécification de l'architecture plus difficile. Les applications bureautiques ont besoin de fonctions pour la manipulation et le stockage de documents généralisés avec les données formatées. Dans les systèmes traditionnels la gestion du texte a été faite indépendamment du SGBD sans qu'une intégration soit possible. La CAO a besoin de pouvoir manipuler des gros objets structurés à l'aide de fonctions particulières. L'association de ces différents types de données et la nécessité d'avoir accès en ligne obligent une ré-évaluation des solution déjà proposées.

La solution à ces nouveaux besoins passe par plusieurs étapes de developpement. Il y a des chercheurs qui pensent que les nouvelles fonctionalités doivent être séparées du SGBD et d'autres qui proposent leur intégration totale dans le SGBD, <LS_83>. Entre ces limites il existe un grand ensemble de

possibilités qui doivent être analysées avant qu'une architecture définitive puisse être établie.

La reusite du developpement d'un projet complexe, tel que TIGRE, est très dependant de la decomposition des objetifs à long terme en activités de réalisation immédiate. La decomposition bien faite et les interfaces bien définies sont essentiels pour le succès de la céation du système et pour permetre l'indépendance entre les personnes <AS_82>. Ce chapitre est un effort pour orienter le processus de developpement dans cette direction.

4.1 Différentes propositions d'architectures

La proposition ANSI/SPARC

Différentes architectures sont possibles pour la réalisation d'un SGBD. Le "American National Standards Committee on Computer and Information Processing" a développé des études pour déterminer les zones où une normalisation pourrait être utile dans la technologie des Bases des Données. Le résultat de ce travail a été la définition d'interfaces. L'autre produit de cet effort a été la définition d'une architecture générale qui, ensuite, fut employée comme référence par la plus grande partie des études sur les base de données. Les conclusions de ce groupe d'étude sont présentées dans <TK_78>. Cette proposition est faite à un haut niveau d'abstraction par rapport à la réalisation physique d'un SGBD. Plusieurs possibilités de réalisation peuvent être schématisées de façon à supporter les différents interfaces et schémas.

Les systèmes de gestion de bases de données actuels proposent une organisation à deux niveaux: les données du point de vue du système et les données du point de vue du programmeur. Pour uniformiser la terminologie le groupe d'étude a proposé

les mots "interne" et "externe" pour faire cette distinction. En plus, un troisième niveaux a été mis en évidence, le niveau "conceptuel". Il représente la déscription de la realité. Le mécanisme pour faire cette description est le schéma conceptuel qui est décrit explicitement dans un langage formel. Les deux autres visions des données, interne et externe, doivent être cohérentes avec la vision exprimé par le schéma conceptuel.

La correspondance entre les objets dans les niveaux externe, interne et conceptuel est établie par des transformations. Les font la liaison entre les descripteurs des différents schémas. Les transformations, sont des fonctions de manipulation, qui emploient l'information fournie par les mappings pour traduire les requêtes externes en requêtes internes, directement ou via le schéma conceptuel. La localisation du schéma conceptuel entre l'externe et l'interne est nécessaire pour fournir un niveau d'accès indirect essentiel à l'independence des données.

Dans la référence <TK_78> une description complète de l'architecture à trois niveaux est faite ainsi que celle des interfaces. Dans <Dat_82> le chapitre 1 fait une discussion didactique de cette architecture.

Interfaces usagers et architectures.

Les interfaces usagers peuvent être classées en trois catégories:

- Interface de haut niveau, non-procédurale,
- Interface de niveau intermediaire, navigationnelle,
- Interface de bas niveau, méthode d'accès.

Une interface de haute niveau , ou non-procedurale, se

caractérise par le fait qu'elle accepte des requêtes qui spécifient ce que l'usager désire extraire de la base sans indiquer comment cette opération doit être réalisée. Par exemple:

> "Selectioner tous les employés qui travaillent au departement d'informatique"

Une telle requête se caractérise aussi, par le fait qu'elle est soumise en bloc au SGBD qui répond en renvoyant l'ensemble des entités qui satisfont aux criteries de selection. Comme la langage de manipulation de données (le DML) travaille avec des ensembles il est possible de faire un découpage net entre le DML et le langage de programmation. Une fois qu'une réponse a été obtenue, le langage peut traiter les occurrences qui composent cette reponse de manière indépendente du SGBD.

Une interface de niveau intermediaire, ou navigationnelle, est orientée vers les occurrences des objets de la base. Les commandes du DML sont mélangés avec ceux du langage de programmation. Ceci est necessaire pour que soit possible la récuperation des ensembles de données voulues. Par exemple, dans la requête ci-dessus il faudrais envoyer d'abord une requête pour selectionner le département d'informattique. Ensuite, il fà cedrait emmettre une requête pour chaque employé attaché à ce departement. La verification des conditions, telles que l'existence d'un objet ou le dernier objet d'un ensemble, sont sous la responsabilité du programme écrit dans le langage hôte.

Au niveau le plus bas d'interface, l'usager ne voit pas les objets du niveau conceptuel mais tout simplesment des enregistrements. Si, par exemple, pour des soucis de performance la représentation d'un objet est décomposée en deux enregistrements, il faudra deux appels au SGBD - ou au

système de fichiers - pour la récupération de toutes les données correspondants. Par exemple, l'entité Personne peut être decomposée en:

identif(no-per, nom, date_naiss, ↑complem)
complem(adresse, adresse_prof, titulation, poste)

ou ↑ correspond a un pointer pour l'enregistrement associé du fichier "complem".

Pour récupérer toutes les informations sur une personne il faut trouver l'enregistrement concerné "identif" et après, par l'adresse ↑complem, faire un nouvel accès au fichier "complem" pour obtenir les données restantes.

Les différentes combinaisons de interfaces pour supporter une interface de haut niveau sont représentées par la figure suivante:

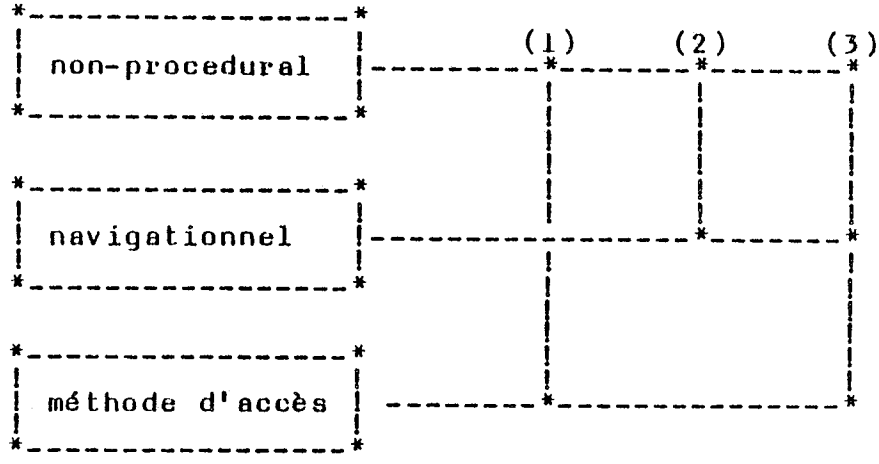


figure 4.1: Les différentes architectures.

Les possibilitées présentées sont: (1) l'interface de haut niveau supportée par un sytème offrant une interface au niveau de la méthode d'accès, (2) supportée par un système offrant une interface de niveau navigationnel ou pour (3) une

combinaison des deux possibilités.

Pour le développement d'un prototype il est intéressant de profitér de SGBDs déjà existants et d'écrire une couche pour la gestion des nouvelles fonctionalités. Ce processus de conception et developpement doit suivre les techniques de structuration bien établies pour que le logiciel soit maintenable. L'architecture (1) parait la plus adequate. Cette architecture est employée par le système R avec un grand effet sur ses performances, <CGI_81>.

En plus de l'architecture il faut décider quelles sont les interfaces accessibles aux usagers; seulement celui de haut niveaux ou bien aussi, les interfaces intermediaires? Une étude de différentes architectures par rapport à l'accès des usagers, aux performances, à la répartition, aux machines bases de données, à la complexité du logiciel est faite dans <RS_82>.

Les données généralisées

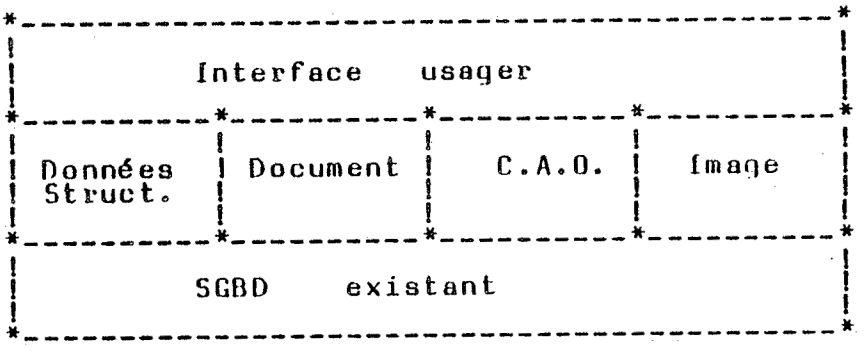
Les SGBDs traditionnellement employés pour des applications de gestion considérent des données structurées. Ces données sont représentées, au niveau physique, par des enregistrements composés d'un nombre fixe ou variable de champs définis sur de types simples. Pour les applications qui ont besoin de gérer des données moins structurées, comme les systèmes documentaires, la solution a été, dans la plus grand partie des cas, de développer des solutions particulières en employant des systèmes des gestion de fichiers, <Sal_68>, <Sal_71>.

Les données généralisées, au contraire des données classiques, ne sont pas atomiques et possèdent une structure interne qui leur sont propres. Par ailleurs, elles sont, en général, d'une

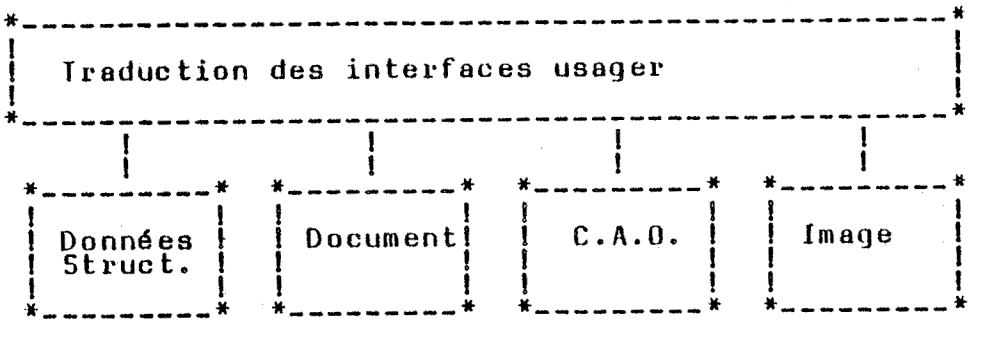
taille importante, ce qui pose de nouveaux problèmes de stockage et de manipulation, <HL_81>. Des exemples de données généralisées sont: les documents structurés (<TIGRE_3>, <TIGRE_6>), les formulaires <TIGRE_9>, les séries historiques <ABDR_83>, les images digitalisées, et les graphiques.

Les solutions possibles pour intégrer ce type de données à un SGBD sont, <LS_83>:

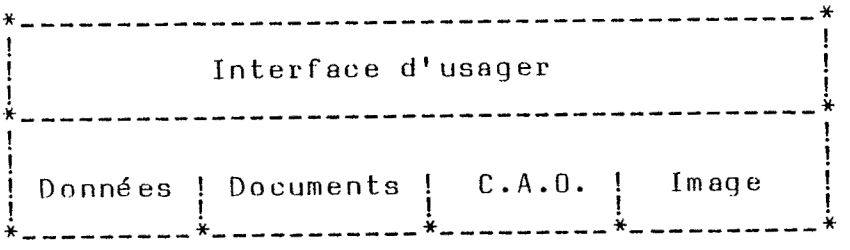
1) Extension d'un SGBD existant au moyen de niveaux supplémentaires construits sur les interfaces déjà existants pour supporter différentes applications.



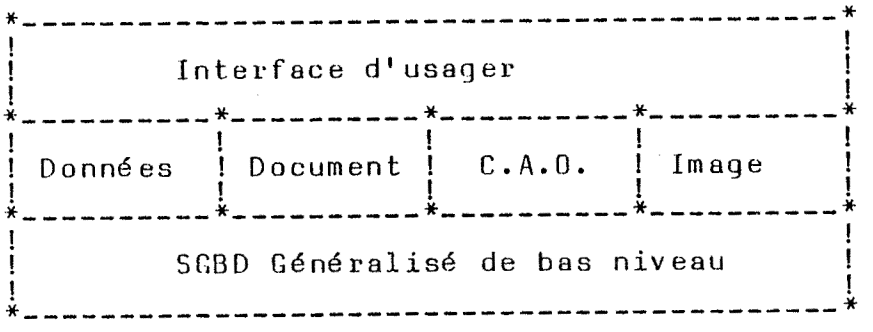
2) Combinaison d'un SGBD avec des systèmes indepéndents pour chaque type de donnée généralisée.



3) Développement d'un nouveau SGBD étendu capable de manipuler toutes sortes de données généralisées.



4) Développement d'un SGBD de bas niveau avec primitives générales et couches spécifiques pour chaque application.



Les différentes proposition d'architectures décrites dans cette section forment le cadre dans lequel les études pour la définition de l'architecture de serveur TIGRE ont été développés.

4.2 L'architecture du serveur TIGRE

Après l'étude des possibilités et en considerant la disponibilité d'equipement et de logiciel nous avons développé la architecture suivante pour le serveur TIGRE.

Pour la réalisation du prototype , nous employons un SGBD relationnel comme couche inférieure et nous créons un module independant pour la gestion des objets complexes. De cette façon nous pouvons limiter l'effort de développement à l'extension du modèle sémantique et à l'introduction des nouveaux types de données sans avoir besoin de modifier le SGBD déjà existent. La structure du prototype est la suivante:

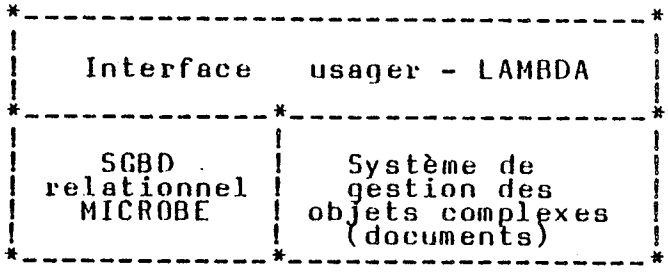


figure 4.2: L'intégration des objets complexes.

Avec une gestion independante des objets complexes il est possible d'essayer différentes approches pour la gestion des nouveax types de données sans interférir avec le développement des extensions concernant les autres concepts liés au modèle de données. Cette structure permetra aussi le developpement d'un module le plus efficace possible pour la gestion des documents.

L'interface usager est developé sur deux interfaces inférieures: un de haute niveau pour les données structurés et une au niveau du methode d'accès pour les objets complexes. La decision d'employer un interface relationnel a été prise pour permetre l'utilisation future d'une machine base de données,

<BS_80> et <BS_82>. De cette façon l'efficacité plus reduite sera largement compensée par les performances de la machine base de données.

A partir du schéma des interfaces présénté dans <TK_78> et reproduit ici à la figure 4.3 nous décrirons en plus détail le serveur TIGRE. La discussion ensuite est basé sur la déscription des interfaces comme présénté au chapitre III de la référence au-dessus. La structure fonctionnelle du serveur est présentée dans la figure 4.4 ou les interfaces décrits sont représentés entre les différents composantes.

(1) Déscription du schéma conceptuel en format source

Le format source du schéma conceptuel est l'interface par laquelle est faite la déclaration de ce schéma. Cette description inclue la spécification des objets conceptuels et leurs propriétés et associations.

Cette interface correspond aux énoncés de définition Lambda. Dans le prototype la définition de l'schéma est faite comme dans l'exemple de l'annexe A4. La définition consiste en la dèscription des objets et des contraintes structurelles. Des extensions sont souhaitées comme l'emploi du type de donnée Document pour faire la déscription des objets de la base.

(2) Déscription du schéma conceptuel en format objet

Le format objet du schéma conceptuel est l'interface par laquelle le schéma conceptuel est connue du reste du système, via le dictionnaire de données. Celui-ci est une méta-base de données qui contient au minimun la représentation du schéma et les

définition de correspondance. Il peut contenir aussi des statistiques sur l'utilisation des divers objets de la base et les déclarations sur la securité d'accès.

Dans le serveur TIGRE le format objet correspond d'une part à ce que nous appellons le catalogue TIGRE (chap 5) et d'autre part à la déscription des objets complexes. Le dictionnaire de données -DD- est la méta-base constituée par les relations du catalogue TIGRE, par celles du catalogue MICROBE et par la représentation des objets complexes stockés par le sous-système particulier.

(3) Description du schéma conceptuel en format de présentation

Le format de présentation est une transcription du schéma conceptuel préparé pour le personnel concerné par le systéme d'information. Il peut être employé pour vérifier la disponibilité et les caracthéristiques des données. L'accès à cette déscription peut être controlée par des contraintes de securité.

La correspondance entre le schéma conceptuel et le schéma interne peut optionnellement être associée à la description des objets conceptuels, interface (15), ainsi pour chaque objet au niveau conceptuel il y aura une déscription de sa représentation dans le catalogue. Le sub-système d'autorisation généralisée (SAGE, <TIGRE_14>) permet de controler la diffusion des informations aux usagers pour controle de la confidencialité.

(4) Description du schéma externe en format source

Le schéma externe spécifie les objets perçus par les différents usagers ainsi que la correspondance avec les objets du schéma conceptuel.

Dans la première version du prototype TIGRE les différents schémas externes corresponent seulement à des restrictions d'accès au schéma conceptuel. La définition de ses schémas est faite à l'aide des énoncés de définition des privileges d'accès définis pour SAGE.

(5) Description du schéma externe en format objet

Le format objet du schéma externe est constitué par le sous-ensemble des relations du dictionnaire de données qui représentent les utilisateurs, les groupes d'utilisateurs et les droits d'accès. La validation de ces énoncés est faite sur la définition du schéma conceptuel.

(6) Description du schéma externe en format compatible avec le langage hote

Pour qu'il soit possible de manipuler les objets de la base par un langage de programmation il faut rendre le schéma externe accessible à ce langage. Le "traducteur de schémas" est le logiciel responsable de cette transformation. Dans le prototype TIGRE cette transformation est accomplie par deux composants: la transformation des représentations relationnelles des objets dans la base de données en structures de données de Pascal et la transformation de la représentation des objets

complexes en un format pivot acceptable par l'editeur spécialisé.

(12) Langage de manipulation des données

Il correspond aux énnoncés de manipulation de Lambda, <TIGRE_11>, et au code intérmediaire résultant. Ce code est composé par des arboréscences relationnelles algébriques intérpretables par MIOPE.

(13) Description du schéma interne

Le schéma interne correspond à la description de l'ensemble des relations MICROBE et des structures de données employées pour le stockage des objets complexes. Le format source est constitué par les énoncés de MIOPE pour la création d'index et il sera ensuite developpée une extension de Lambda pour permettre le contrôle de la décomposition des objets conceptuels en relations MICROBE, (chap 5). Le format objet est l'équivalent au catalogue du SGBD MICROBE.

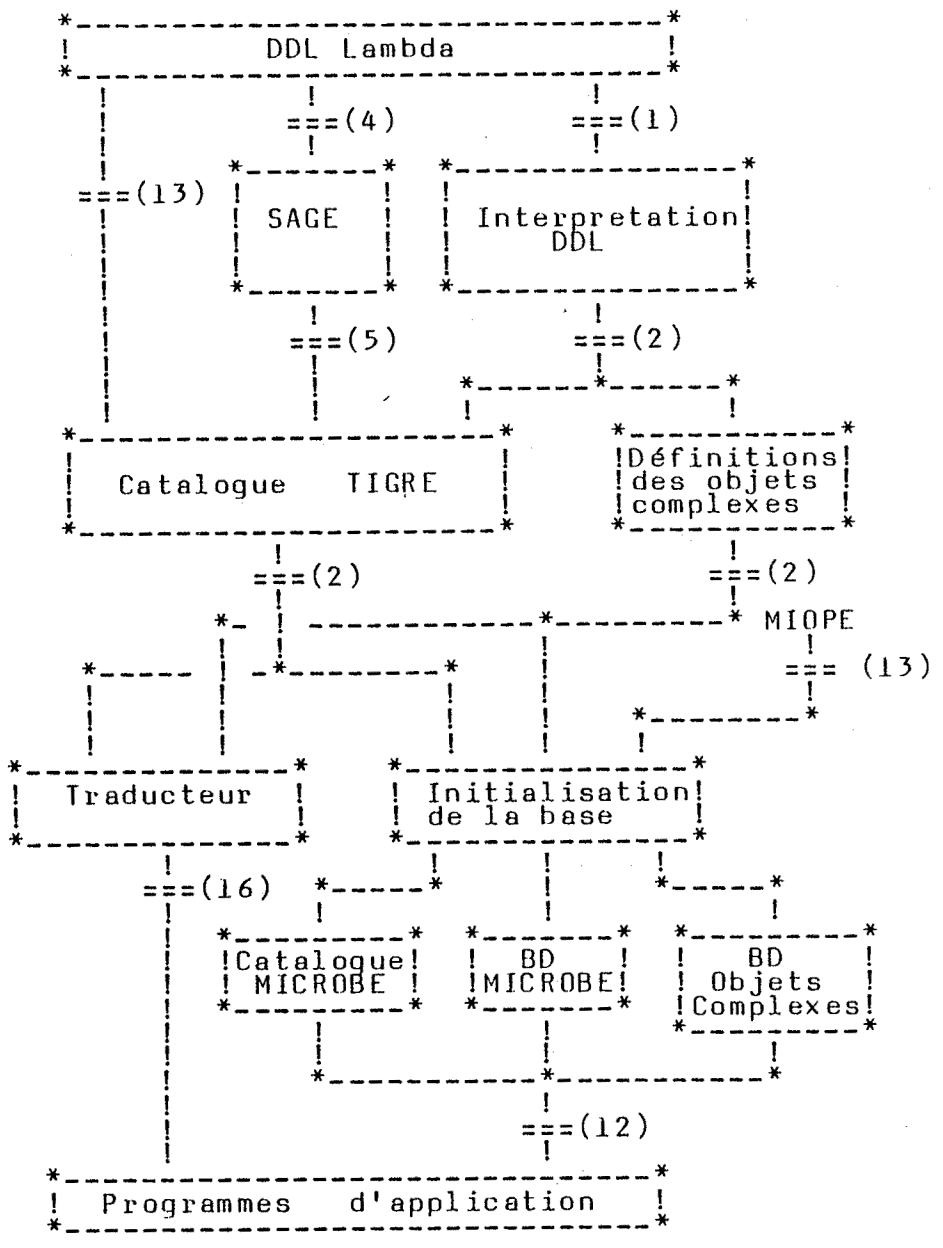


figure 4.4: L'architecture du serveur TIGRE.

4.3 La réalisation

Le prototype est en développement, le logiciel est écrit en Pascal sur le système d'explotationn MULTICS d'un ordinateur HB-68. Le systéme relationnel de base est MICROBE, section 4.4.

Nous avons choisi comme interface entre TIGRE et MICROBE les requêtes algébriques en forme arborescent. Cette interface est employée entre le traducteur MIQUEL et l'optimiseur MIRES, <Fert_81>. Les requêtes LAMBDA sont traduites en arborescences algébriques et ces arborescences sont optimisées et executées par MICROBE.

La routine d'accès au SGBD relationnel reoit les arborescences algébriques dans une forme parenthésée. Cette architecture permet une decomposition du logiciel en composants distribués. L'interpreteur peut être resident aux postes de travail et la communication avec le serveur de base de donnés est fait par l'appel remote de la routine d'accès à la base. Pour les objets complexes la structure de l'interface est similaire mais independante, car le système de gestion des objets complexes est separé du SGBD MICROBE.

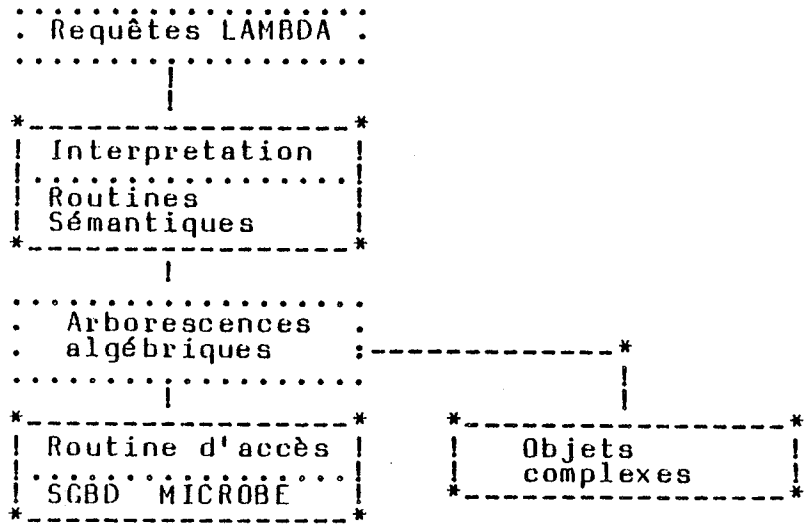


figure 4.5: Interface entre TIGRE et MICROBE.

Comme MICROBE n'accepte qu'une base ouvert par fois le catalogue et les bases TIGRE coexistent dans une seule base MICROBE. L'independance entre le cataloge et les différentes bases est assuré par le logiciel TIGRE. La création de la base MICROBE est faite par la primitive "initialize" qui réserve l'espace en disque et génére les relations du catalogue TIGRE.

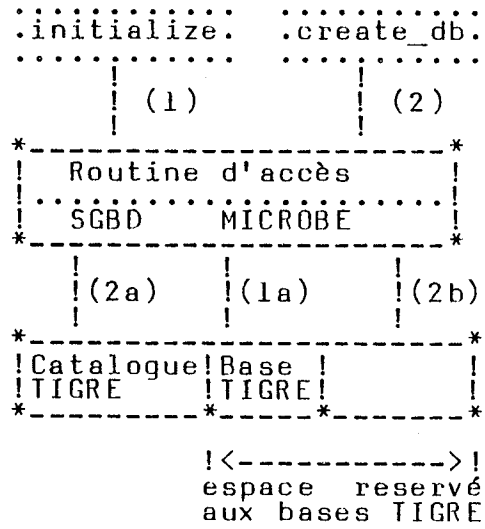


figure 4.6: Inicialisation du système.

Chaque base TIGRE est crée par un appel à la primitive "create_db <db_name>" et correspond a un ensemble de relations dans l'espace réservé pour TIGRE. Les bases TIGRE sont, aussi, composées par la description des objets complexes et par la représentation de ses occurrences.

Nous avons développé les modules d'analyse sintaxique des énoncés de définition, de création du catalogue TIGRE, d'interface avec le SGBD MICROBE et pour l'inicialisation de la base TIGRE. L'interprétation des énoncés de définition LAMBDA et sa transformation dans la représentation relationnelle est décrite dans le chapitre 5.

4.4 Le SGBD relationnel MICROBE

Le SGBD MICROBE est le resultat d'un projet développé au Laboratoire IMAG à partir de 1979. Le prototype est operationnel sur des micro-ordinateurs LSI-11/23 et sur un ordinateur HB-68 doté du sytème d'explotation Multics, <Fern_81>, <NFFL_82>. Le prototype TIGRE est en phase de réalisation sur cette denière version.

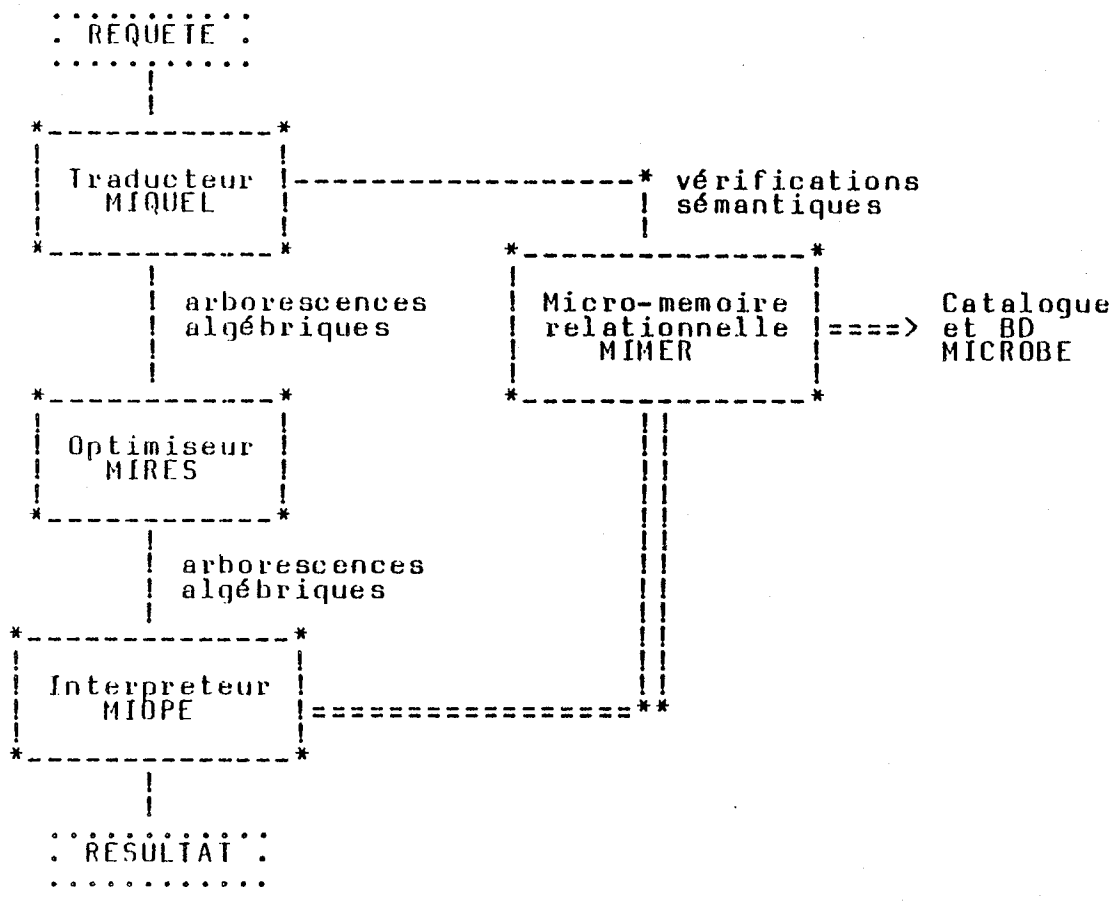


figure 4.7: L'architecture de MICROBE.

MICROBE est composé de quatre modules: le traducteur de requêtes MIQUEL, l'optimiseur d'arborescences algébriques MIRES, l'interpréteur d'arborescences MIOPE et la micro-memoire relationnelle MIMER. Ces modules sont reliés comme le montre la figure 4.7 .

Le langage de requête pour MIQUEL est une version réduite de SEQUEL, <CB_74>. Ce langage est interpretée par le traducteur MIQUEL qui fait des vérifications sémantiques en consultant le catalogue via MIMER. Le résultat de ce processus est une arborescence algébrique relationnelle. Ensuite, cette arborescence est optimisée statiquement par MIRES qui emploi des règles d'optimisation sur l'ordre d'execution des opérateurs algébriques. Finalement, cette arborescence optimisée est évaluée par MIOPE qui accede à la base via MIMER et renvoi le résultat à l'utilisateur.

Chapitre 5

LA REPRESENTATION DU SCHEMA CONCEPTUEL

5. La représentation du schéma conceptuel

Dans ce chapitre nous traitons du processus de compilation du schéma conceptuel pour sa transformation en format objet. La définition du format objet du schéma conceptuel est basée sur des concepts proposés par Codd, <Cod_79> et <Dat_83>, dans RM/T. Cette proposition est une extension du modèle relationnel originel, <Cod_70>.

La structure du catalogue que nous définissons est une extension de la proposition initialle de RM/T. Nous y avons inclus plusieurs nouvelles relations pour pouvoir décrire sous forme relationnelle les types définis dans un schéma conceptuel correspondant au modèle TIGRE. Ce cataloque est une base de données du point de vue du SGBD sous-jacent. A terme nous envisageons l'utilisation d'une machine base de données tel que VERSO, <BS_80> et <BS_82>, comme support relationnel.

La section 5.1 présente une description de RM/T, la section 5.2 décrit l'utilisation que nous faisons de certains concepts de ce modèle et, finalement, la section 5.3 montre, en détail, la correspondance entre les énoncés de définition de LAMBDA et le catalogue généré.

5.1 Le modèle relationnel étendu RM/T

Cette section fait une description du modéle relationnel étendu RM/T et est basée sur les références <Cod_79> et <Dat_83>.

5.1.1 Entités et types d'entités

Le concept de base d'un modèle est la possibilité de représenter la réalité au moyen d'entités. Les entités peuvent être décrites d'une façon informelle comme n'importe quel

objet perceptible. Il est supposé, aussi, que les entités peuvent être classées dans des types d'entités. Par exemple, une personne est considerée comme une instance du type Personne. Une entité peut avoir plusieurs types comme la personne qui appartient au type Personne et au type Employé. En plus un ensemble de rapports peuvent exister entre des entités, par exemple, deux ou plusieurs entités peuvent être liées par une association ou un type peut être un sous-type d'un autre type.

Les entités et les types d'entités sont classés dans trois catégories:

1) Caractéristiques: ces entités ont pour but décrire autres entités. Les entités caractéristiques sont dépendentes des entités qu'elles décrivent. La raison pour l'existence de ces entités est une conséquence d'une dépendence multivaluée ou d'une dépendence fonctionnelle transitive. Dans ce dernier cas une entité a une propriété immédiate qui a son tour a une autre propriété immédiate. Par exemple, une route a un certain nombre de matériels de surface qui à leur tour, ont des porosités différentes. Un type d'entité qui est caractéristique par rapport aux secteurs de la route est introduit pour représenter les types de matériel de surface de chaque sectéur et leur porosité.

2) Associatives: ces entités sont destinées à représenter les liens entre d'autres entités. Par exemple, une affectation représente une association entre un employé et un projet. Les entités liées sont indépendantes, n'est à dire qu'aucune entité liée n'est existentiellement dépendante des autres.

3) Noyau: une entité noyau est une entité qui n'est pas caractéristique ni associative. Ce sont des entités qui ont une existence indépendente.

Tous les types d'entités peuvent avoir un ou plusieurs sous-types qui a leur tour peuvent avoir des sous-types. Le type E2 est un sous-type du type E1 si toutes les entités qui appartiennent au type E2 sont obligatoirement des entités du type E1 et ont au moins une propriété particulière. Par exemple, Ingénieur est un sous-type d'Employé. Un sous-type appartient au même groupe que son super-type. Un sous-type d'une entité caractéristique est, aussi une entité caractéristique.

5.1.2 Surrogates

Les clés primaires définies par l'usager ont certains faiblesses telles que:

- Elles sont sujettes à des changements si l'environnement change

- Différentes clés peuvent identifier la même entité

- Il peut être nécessaire de mantenir des informations sur une entité avant qu'elle ait reçu une clé controlée par l'usager

Pour surmonter ces problèmes, RM/T emploie la notion de surrogates. Un surrogate est un identificateur unique généré par le système. L'operation "Créer_entité" entraine la génération d'une nouvelle valeur de surrogate, unique par rapport à tous les surrogates existants ou qui ont existé dans le système.

Dans la base de données toute identification d'une entité et toute référece à une entité est faite par un surrogate. Cela signifique que les clés primaires et les clés étrangères sont des surrogates. Les clés des usagers sont considerées comme des propriétés des entités. De cette façon les usageurs n'ont pas besoin de créer de clés artificielles dans des situations où des clés "naturelles" n'existent pas.

Le domaine-E est le domaine de toutes les valeurs possibles de surrogates. Par convention un attribut défini sur le domaine-E a pour suffixe le caractére "ć".

5.1.3 Relations-E et relations-P

Une base RM/T contient une relation-E par classe d'entité. Cette relation est une relation unaire qui contient tous les surrogates des entités appartenat à cette classe et existant actuellement dans la base. Le but des relation-E est d'indiquer l'existence des entités et de servir comme point central de référence pour toutes les entrées dans la base de données concernant des entités appartenant au type. Par convention le nom de la relation-E est le même que le nom de la classe d'entités qu'elle représente et le seul attribut de la relation a pour nom le nom de cette suivi par le caractére "ć".

Les propriétés monovaluées d'une entité sont représentées par des attributs d'une ou de plusieurs relations-P. Chaque relation-P a comme clé primaire un attribut défini sur le domaine-E qui lie les propriétés de chaque entité avec l'assertion de son existence définie par la relation-E. Par convention la clé primaire de chaque relation-P est représentée par un attribut qui a le même nom que l'attribut de la relation-E correspondante.

La manière précise dont les propriétés d'un type d'entités sont groupées en relations-P est sous la responsabilité du concepteur du schéma interne. Une solution extreme consiste à grouper toutes les propriétés dans une seule relation-P et une autre solution consiste à avoir une relation-P binaire pour chaque propriété.

Exemple: nous voulons créer une base de données RM/T contenant des données sur les employés d'une entreprise et les sites où ils travaillent. Pour chaque employé nous voulons maintenir le nom, l'adresse et l'historique des travaux qu'il a accompli dans l'entreprise. Employé est un type d'entités ayant comme propriétés monovaluées nom et adresse et comme propriété multivaluée l'historique des travaux. Travail est un type d'entités caractéristique qui décrit la propriété multivaluée d'Employé. Site est un type d'entités ayant les propriétés nom et code postal et, finalemant, Loc est un type d'entités associatives liant Employé et Site. Dans la figure suivante les identificateurs S1...S5 représentent les valeurs des surrogates.

Employé

employé_é
S1

relation-E

Employé_nom

employé_é	nom	prenom
S1	Lux	Jean

relation-P

Employé_addresse

employé_é	nº	rue	ville
S1	7	Foch	Lille

relation-P

Travail

travail_é
S2
S5

relation-E

Travail_employé

travail_é	employé_é
S2	S1
S5	S1

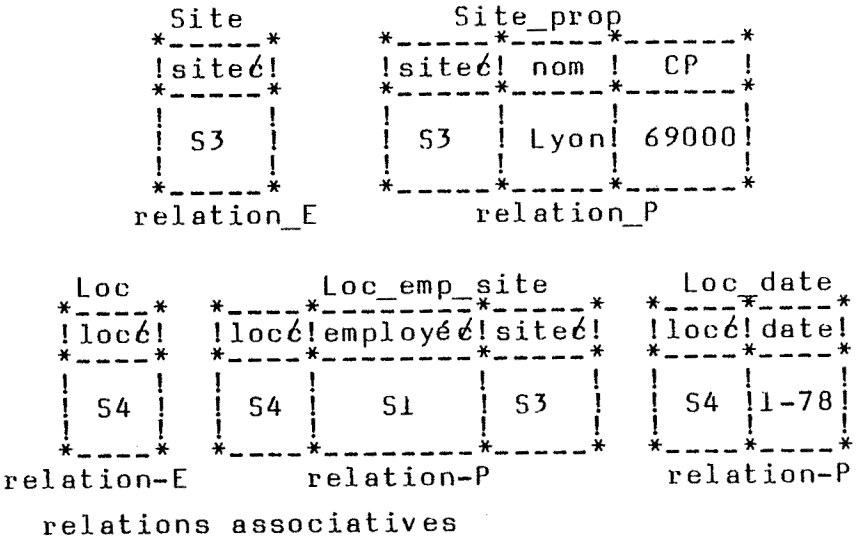
relation-P

Travail_prop

travail_é	date	nom_trav	salaire
S2	1-78	progrmm	4500
S5	2-79	analyste	6500

relation-P

relations caractéristiques



5.1.4 Généralisation et agrégation

Chaque entité appartient à un type d'entités mais elle peut aussi, appartenir à plusieurs types au même temps. Par exemple, un employé peut appartenir simultanement aux classes suivantes:

1) Employé

2) Ingénieur, un sous-type immédiat d'Employé

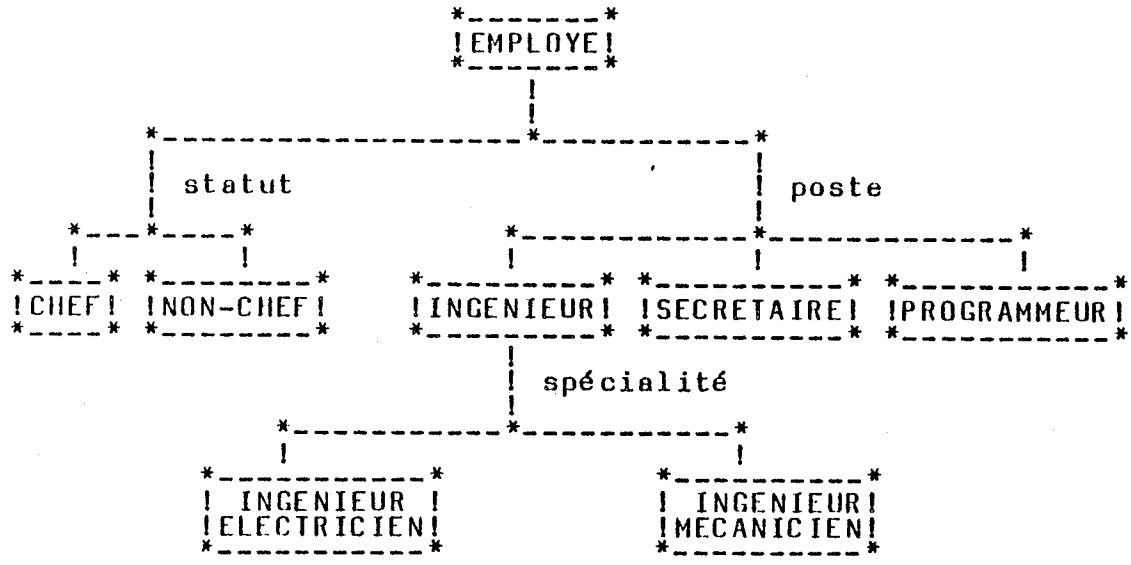
3) Ingénieur Electricien, un sous-type immédiat d'Ingénieur et sous-type de second ordre d'Employé

4) Chef, un autre sous-type immédiat d'Employé

Le concept de généralisation inclut les notions de sous-type et de super-type. Le terme généralisation est dérivé du fait que, par exemple, Ingénieur est une généralisation d'Ingénieur Electricien et d'Ingénieur Mécanicien. D'une façon équivalent ces deux deriers sont des spécialisations d'Ingenieur au sens que toutes les propriétés d'Ingénieur s'appliquent aux

Ingénieurs Electriciens et aux Ingénieurs Mécaniciens. L'inverse n'est pas vrai car il peut avoir des propriétés qui sont relatives aux Ingénieurs Electriciens mais non aux Ingénieurs en général. l'ensemble de sous-types et super-types est appelé une hiérarchie de types et permet au concepteur du schéma conceptuel d'eviter les occurrences de valeurs nulles de type "non applicable".

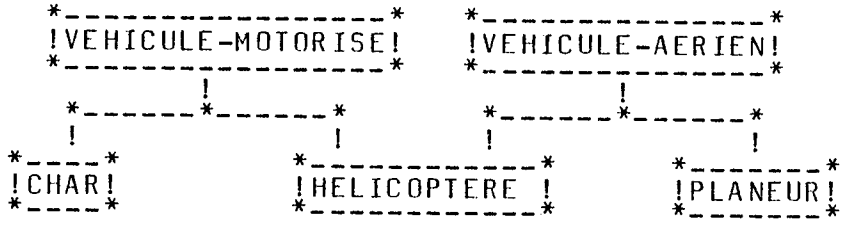
Exemple:



Cet exemple montre la division d'un super-type en catégories distinctes de sous-types. Les types Chef et Non-chef forment une catégorie du super-type Employé et Ingénieur, Secretaire et Programmeur une autre. Les noms des catégories sont indiqués à coté des arcs correspondents.

Il est aussi, possible qu'un sous-type soit spécialisé à partir de deux ou plusieurs super-types immédiats. Dans ce cas une instance du sous-type sera simultanement une occurrence de chaque super-type immédiat. Autrement dit, les hiérarchies de généralisation ne sont pas forcément disjointes.

Un exemple est :



La raison pour qu'un sous-type soit généralisé dans plusieurs super-types indépendants est de permetre que le sous-type hérite de toutes les propriétés des super-types en plus de ses propriétés particulières.

5.1.5 Règles d'intégrité, catalogue et opérateurs

En plus de deux règles d'intégrité du modèle relationnel des nouvelles règles existent pour RM/T, ce sont:

3) Integrité des entités

Comme conséquence de la définition d'identificateur unique pour les surrogates les relations-E acceptent des insertions et des suppressions mais non des modifications.

4) Intégrité des propriétés

Si un n-uplet (t) apparait dans un relation-P la clé primaire (surrogate) de t doit apparaitre dans la relation-E correspondante à P.

5) Intégrité caractéristique

Une entité caractéristique ne peut exister dans la base de données que si l'entité qu'elle décrit

existe aussi dans la base.

6) Intégrité associative

Une entité associative ne peut exister dans la base que si pour cette entité associative les références aux entités associées identifient une entité de type approprié ou ont un valeur nul.

7) Intégrité des sous-types

une relation-E, correspondant à une entité E1, doit appartenir à toutes les relations-E des entités qui sont des super-types de E1.

RM/T contient son propre catalogue pour maintenir les informations sur le schéma des bases de données. Les références bibliographiques citées au début du chapitre proposent la structure d'un catalogue minimal. A partir de cette description nous avons défini le catalogue du serveur TIGRE décrit dans l'annexe A3.

Des opérateurs sont décrits qui font partie intégrante du modèle pour permettre la manipulation de façon uniforme des schémas des bases et ses extensions. Comme nous n'avons employé que les concepts liés à la structure de RM/T pour développer le serveur TIGRE les opérateurs ne seront pas décrits ici.

5.2 La correspondance entre RM/T et TIGRE

En comparant le modèle RM/T (section 5.1) et le modèle TIGRE (chapitre 1) il est possible de constater les points communs entre les deux modèles. Ils appartiennent à la catégore des modèles sémantiques et leurs concepts de structures de données sont très proches. Pour développer un serveur base de données correspondant au modéle TIGRE sur un SGBD relationnel l'emploi de RM/T nous est apparu comme un outil commode pour faire la correspondance entre les schémas. La raison de cette décision est la position intermediaire de RM/T entre TIGRE et le relationnel puisqu'il est une extension du modèle relationnel qui va dans le sens de TIGRE. Nous employons les concepts de RM/T pour structurer la description d'un schéma TIGRE sous forme relationnelle.

La correspondance des concepts entre les modèles est décrite ainsi:

TIGRE	RM/T
Type structuré	Type d'entité
Classe d'entités ou d'associations	Type d'entité
Occurrence d'un objet appartenant à une classe d'entités	Entité noyau
Occurrence d'un objet appartenant à une classe d'associations	Entité associative
Type construit	Type d'entité caractéristique
Occurrence d'un objet de type construit	Entité caractéristique
Spécialisation	Généralisation inconditionnelle à un seul super-type
Intersection	Généralisation inconditionnelle à plusieurs super-types

TIGRE	RM/T
Union	Généralisation alternative
Agrégation d'entités	"Cover aggregation"
Agrégation associative	"Cover aggregation" d'un type d'entité associative et de ses entités participantes

Il faut noter que dans TIGRE, les deux emplois qui RM/T fait d'entités caractéristiques, la représentation des propriétés multivaluées et des propriétés indirectes, sont separés. Pour les multivaluées nous employons les types liste et pour les indirectes les types enregistrement.

5.3 La génération du catalogue TIGRE

La représentation relationnelle d'une base de données TIGRE contient une reletion_E par classe. Il y a aussi une relation_E par type construit Enregistrement, Liste et Document. La relation_E sert à indiquer l'existence des occurrences. Toutes les valeurs des attributs des occurrences sont maintenus par une relation_P. Pour les attributs définis sur des types construits dans la relation_P il y a un surrogate qui fait référence à la relation_E qui correspond au type construit. Les convention de nommage des relations et des attributs sont les mêmes que dans RM/T.

Dans le serveur, toutes les références internes sont faites via des surrogates. Les objets du catalogue identifiés des surrogates sont: base de données, domaines, attributs, listes parenthesées pour la description de documents, relations et prédicats. D'un point de vue conceptuel, les types, classes et occurrences d'entiés ainsi que les types construits et ses occurrences sont identifiés dans la base par ses surrogates. Les surrogates sont des identificateurs internes invisibles par les usageurs.

Dans le prototype, les surrogates sont une sequence monotonique croissante d'entiers. Les valeurs des surrogates associés aux objets detruits ne sont jamais re-employés. Comme l'ordinateur HB-68, sur lequel le prototype est en développement, a un limite de plus de 34 milliards pour les entiers cette approache est parfaitemente valable.

Montrons maintenant la correspondance entre chaque catégorie d'énoncé de définition LAMBDA et le catalogue TIGRE. La totalité de l'exemple est donnée à l'annexe A4.

5.3.1 Les types simples

Un type simples est complètement décrit par un n-uplet dans la relation CAT_D. Les types simples sont pré-définis au sens que les n-uplets qui les décrivent sont générés pendant la création de la base. Il n'y a pas de définition dans LAMBDA pour ces types. La définition de la longueur d'une chaine de caractères est vue comme une restriction du type "chaine de caractères" associée à la définition des attributs ou des types renommés.

CAT_D

D_c	dom_name	of_type	data_type
1	Integer	Integer	Integer
2	Real	Real	Real
3	Boolean	Boolean	Boolean
4	String	String	Char
5	E_domain	Integer	Integer
6	Time	String	Char

5.3.2 Types restreints

Les types restreints sont les intervalles et les scalaires auxquels nous pouvons ajouter la définition implicite d'une restriction sur la longeur d'une chaine de caractères. Pour représenter ces types il faut des relations supplémentaires pour maintenir les valeurs possibles des domaines, les restrictions. Ces relations sont: CAT_INTD pour les intervalles, CAT_SCAD pour les scalaires et CAT_STRING pour les chaines de caractères.

Exemple:

(1..10)
(préliminaire, definitive, rappel)
string (20)

CAT_D

D_c	dom_name	of_type	data_type
114	notnamed	interv	1
75	notnamed	scalar	4
35	notnamed	String	4

CAT_INTD

D_c	min	max
121	1	10

CAT_STRING

D_c	length
35	20

CAT_SCAD

D_c	element
75	préliminaire
75	definitive
75	rappel

La définition d'un attribut directement sur un domaine représenté par un type restreint donne lieu à la création d'un domaine identifié dans le catalogue par le mot "notnamed". ça signifique que ce domaine est uniquement référencé par son surrogate et que les opérations de comparaison sont valables entre attributs définis sur des domaines caractérisés par le même type de donnée.

5.3.3 Types renommés

Pour les types renommmés la différence dans la représentation sera le changement de valeur de "dom_name" pour le nom du type. Par exemple:

```
type nom_pays : string ;
     t_datenv : list (3) of string (8) ;
```

CAT_D

D_c	dom_name	of_type	data_type
8	nom_pays	String	4
9	t_datenv	List	5

5.3.4 Types enregistrement

Les types enregistrement correspondent à la définition de propriétés indirectes d'un type classe. Nous les représentons comme de domaines avec une structure plus complexe que les types simples. Par exemple:

```
type t_loc    : record
                  ville : string (20) ;
                  pays  : nom_pays
                end ;
```

La représentation du type t_loc est faite par:

CAT_D

D_c	dom_name	of_type	data_type
17	t_loc	record	5

La valeur "domaine_E" de l'attribut "data_type" signifie que les attributs définis sur ce domaine auront comme valeur un surrogate qui fait référence au n-uplet qui contient les valeurs des propriétés indirectes.

La réprésentation d'un type enregistrement en LAMBDA correspond à la création d'une relation_E et d'une relation_P dans la base. Les occurrences de ce type enregistrement sont référencées par des surrogates. Dans le catalogue ces relations sont représentées par tules de CAT_R.

CAT_R

R_c	rel_name	rel_type
19	t_loc	ER
18	t_loc_p	P

CAT_COMP

R_comp_c	RE_c
18	19

Pour lier les relation E et P nous avons défini la relation CAT_COMP qui mantient un n-uplet pour chaque relation-P.

Le domaine correspondant à un type enregistrement est représenté par l'ensemble des relations E et P. Pour faire cette liaison il existe la relation du catalogue CAT_STRUC. Cette relation est équivalente à la relation "CG-relation" du catalogue RM/T.

CAT_STRUC

D_c	R_c
17	19

La description des relations de la base est faite par CAT_A. Cette relation contient toutes les descriptions des attributs des relations E et P. Il y a un attribut dans CAT_A qui correspond à chaque attribut (champ) du type enregistrement.

CAT_A

A_c	R_c	D_c	att_name	E_ref	uk
20	19	5	t_loc_c	19	F
21	18	5	t_loc_c	19	F
22	18	23	ville	0	F
24	18	8	pays	0	F

Les valeurs de "D_c" 23 et 8 font référence à deux domaines, respectivement un domaine "notnamed" du type string(20) et un de nom "nom-pays" défini comme string(36). La valeur "0" de "E_ref" est l'equivalent d'une valeur nulle du type non applicable.

Les relation de la base décrit es par cet exemple sont:

t_loc

t_loc_c
.....

t_loc_p

t_loc_c	ville	pays
.....	.....	....

Dans ces relations les valeurs des occurrences des enregistrements décrits par le type t_loc sont maintenues.

5.3.5 Types liste

La définition d'un type liste en LAMBDA correspond aussi à la création d'une relation_E et d'une relation_P dans la base. Dans cette dernière relation chaque élément de la liste correspond à un n-uplet.

Exemple:

```
type t_datenv : list_of_string (8) ;
```

CAT_D

D_c	dom_name	of_type	data_type
9	t-datenv	L	5
16	notnamed	S	4

CAT_R

R_c	rel_name	rel_type
10	t_datenv	EL
11	t_datenv_p	P

CAT_COMP

R_comp_c	RE_c
11	10

CAT_A

A_c	R_c	D_c	att_name	E_ref	uk
12	10	5	t_datenv_c	10	F
13	11	5	t_datenv_c	10	F
14	11	1	order	0	F
15	11	16	value	0	F

CAT_LIST		CAT_STRUC		CAT_STRING	
D_c	no_elem	D_c	R_c	D_c	length
9	3	9	10	16	8

Les relations suivantes seront crées dans la base:

t_datenv
t_datenv_c
.....

t_datenv_p		
t_datenv_c	ordre	value
.....	.....	.....

5.3.6 Types document

La représentation des documents comprend deux niveaux: la définition et l'occurrence. La représentation des définitions est distribuée entre le catalogue TIGRE et le module de gestion des gros objets complexes. La gestion de l'espace physique de stockage ainsi que les primitives d'accès et les autres fonctions de manipulation de documents sont assurées par ce module.

Un type document correspond , dans le catalogue TIGRE, à un domaine document. La représentation des types Paragraphe, Section et Rapport, conforme à la définition faite à la page 1. est:

CAT_D			
D_c	dom_name	of_type	data_type
400	Paragraphe	Document	5
402	Section	Document	5
404	Rapport	Document	5

CAT_DOC	
D_c	list_c
400	401
402	403
404	405

La relation CAT_DOC fait référence au module des objets complexes où est stockée une liste parenthesée. Cette liste correspond à une description d'un document. L'accès à la liste

est fait par son surrogate "liste_c". Elle correspond à la représentation employée pour l'echange de documents entre l'éditeur et la base de données.

Les listes parenthesées définisant les schémas de documents exemplifiés à la page sont:

```
(Paragraphe DEFTYPE (STRUC (LIST 1 * STRUC (Unité)))
                    (CONST () ))

(Section DEFTYPE
             (STRUC (AGR (titre Texte)
                         (contenu LIST 1 * STRUC (Paragraphe))))
             (CONST () ))

(Rapport DEFTYPE
          (STRUC (AGR

          (page_garde AGR ((entête Text VAL (T0))
                           (titre Text)
                           (auteur REF_PAR VAL (nom_auteur))))

          (introduction AGR ((titr Text VAL (T1))
                             (contenu Text))

          (corps List 1 * STRUC (section))

          (conclusion STRUC (Paragraphe)))

          (CONST ((Text T0 'Laboratoire IMAG')
                  (Text T1 'Introduction')
                  (Paramètre nom_auteur (Text)))))
```

5.3.7 Entités

La représentation des types entité dans le catalogue est faite par un domaine de type "e" représenté par une relation_E et une relation_P. Pour la définition des attributs nous employons la structure de domaines presentée plus haut.

Exemple: type Personne : entity

```
        key IFIP_no : integer ;
            nom     : string (20) ;
            adresse : t_adresse ;
            invité  : boolean
        end ;
```

CAT_D

D_c	dom_name	of_type	data_type
25	tadresse	r	5
44	notnamed	S	4
39	Personne	e	5

CAT_STRING

D_c	length
44	20

CAT_R

R_c	rel_name	rel_type
37	Personne	EK
38	Personne_p	P

CAT_COMP

R_comp_c	RE_c
38	37

CAT_STRUC

D_c	R_c
39	37

CAT_A

A_c	R_c	D_c	Att_name	E_ref	uk
40	37	5	Personne_c	37	F
41	38	5	Personne_c	37	F
42	38	1	IFIP_no	0	T
43	38	44	nom	0	F
45	38	25	adresse	27	F
46	38	3	invité	0	F

Cette représentation dans le catalogue décrit les relations suivantes de la base:

Personne

Personne_c
.....

Personne_p

Personne_c	IF	IP_np	nom	adresse	invité
.....	.....	.....	...	.....	.....

5.3.8 Associations

La représentation des types associations suit les mêmes principes que celle des d'entités quis. Pour maintenir l'information sur les types d'entités qui particient à l'association, les rôles et les cardinalités, nous employons la relation du catalogue CAT_DESIG. La relation_E correspondante à une association a le type "EA". Les valeurs des propriétés sont maintenues dans la base par une relation_P. Les surrogates des occurrences appartenant à une occurrence de l'association sont stockées dans une relation_A.

Exemple: type Autship : relationship
between Personne : aut (1,10)
and Article : art ;

aut_no : integer
end ;

CAT_D

D_c	dom_name	of_type	data_type
61	Autship	a	5

CAT_STRUC

d_c	R_c
61	59

CAT_R

R_c	rel_name	rel_type
59	Autship	EA
60	Autship_p	P
64	Autship_d	A

CAT_COMP

R_comp_c	RE_c
60	59
64	59

CAT_A

A_c	R_c	D_c	att_name	E_ref	uk
62	59	5	Autship-c	59	F
63	60	5	Autship_c	59	F
65	64	5	Autship_c	59	F
66	64	5	Persone_c	37	F
67	64	5	Article_c	47	F
68	60	1	Auteur_no	0	F

CAT_DESIG

RR_c	RE_c	role_name	min	max
59	37	aut	1	10
59	47	art	0	999

Les relations suivantes seront crées dans la base:

Autship

Autship_c
.....

Autship_p

Autship_c	auteur_no
.....	.....

Autship_d

Autship_c	Personne_c	Article_c
.....	.....	.....

5.3.9 Agrégation d'entités

La relation CAT_EAGG mantient l'information sur la composition du type d'entité agrégée. Elle maintient une référence aux entités composantes de l'agrégation et les contraintes de cardinalité. La relation_E correspondante à l'agrégation est de type "EE".

Dans la base, en plus des relations E et P, il existe une relation_G qui a pour but de maintenir les valeurs des

surrogates des occurrences qui appartiennent à une occurrence d'entité agrégée.

Exemple: type Dossier : entity_agr
of Appel (1,3)
and Article ;

dossier_no : integer
end ;

CAT_D

D_c	dom_name	of_type	data_type
81	Dossier	e	5

CAT_STRUC

d_c	R_c
81	79

CAT_R

R_c	rel_name	rel_type
79	Dossier	EE
80	Dossier-p	P
84	Dossier_g	G

CAT_COMP

R_comp_c	RE_c
80	79
84	79

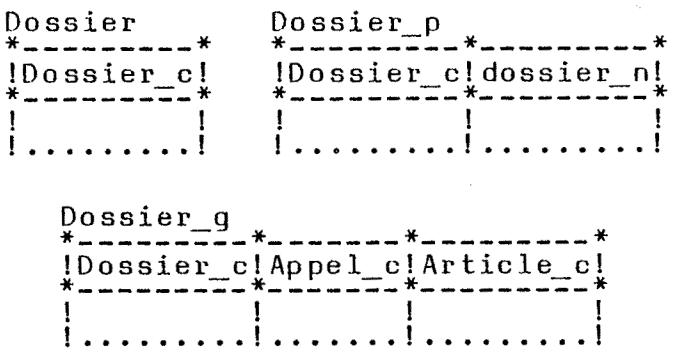
CAT_A

A_c	R_c	D_c	att_name	E_ref	uk
82	79	5	Dossier_c	79	F
83	80	5	Dossier_c	79	F
85	84	5	Dossier_c	79	F
86	84	5	Appel_c	69	F
87	84	5	Article_c	47	F
88	80	1	dossier_n	0	F

CAT_EAGG

R_agg_c	R_comp_c	min	max
79	69	1	3
79	47	0	999

Dans la base sont crées les relations suivantes:



5.3.10 Agrégation associative

Une occurrence d'une classe dérivée par agrégation associative a pour clé la concatenation des clés des entités appartenant à l'association. Comme conséquence de cette définition il est clair qu'il y a une correspondance 1:1 entre les occurrences de l'association et les occurrences de l'agrégation associative. La représentation d'un type agrégation associative est faite par une relation_E "virtuelle". Cette relation a le type "AA" et est une redéfinition de la relation_E correspondant à l'association. Celà est du au fait que toutes les occurrences de l'association sont aussi des occurrences de l'agrégation. Tout se passe comme si cette relation_E avait un deuxieme nom -"alias"- celui de l'agrégation.

Les propriétés de l'agrégation sont les propriétés des entités appartenant à l'association, celles qui sont propres à l'association et en plus des eventuelles propriétés particuliéres à l'agrégation.

Une agrégation associative a une relation_P pour maintenir la valeur de ses propriétés particuliéres. Pendant l'accès à l'ensemble des propriétés de l'agrégation on effectue le produit entre les relations_P qui correspondent aux entités composantes, à l'association et à l'agrégation.

Exemple type Chapitre : relshp_ag
of Art_sess ;

chap_no : (1..10)
end ;

CAT_D

D_c	dom_name	of_type	data_type
49	Article	e	5
91	Session	e	5
99	Art_sess	a	5
108	Chapitre	e	5
114	notnamed	i	1

CAT_STRUC

d_c	R_c
49	47
91	89
99	97
108	106

CAT_INTD

D_c	min	max
114	1	10

CAT_R

R_c	rel_name	rel_type
47	Article	EK
48	Article_p	P
91	Session	EK
92	Session_p	P
97	Art_sess	EA
106	Chap	AA
107	Chap_p	P

CAT_COMP

R_comp_c	RE_c
48	47
91	89
107	106

CAT_A

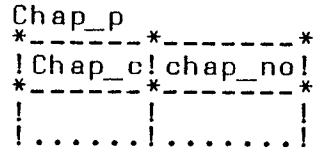
A_c	R_c	D_c	att_name	E_ref	uk
109	106	5	Chap-c	106	F
110	107	5	Chap_c	106	F
113	107	114	chap_no	0	F

CAT_AAGG

Ragg_c	R_comp_c
106	97

Comme la relation_E correspodant à l'entité Chap est virtuelle

seulement une relation_P est crée dans la base:



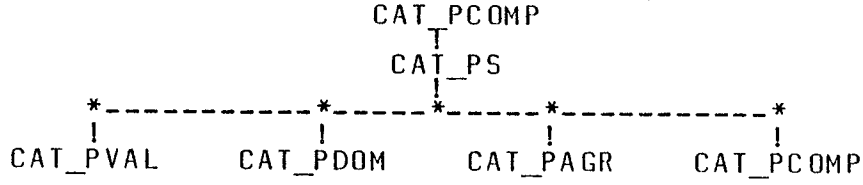
5.3.11 Généralisation

Pour représenter la généralisation nous employons la relation CAT_GEN. Cette relation lie le type classe résultat, le type classe opérande et le prédicat, s'il existe dans la définition du type dérivé. La relation-E correspondant à un type dérivé par généralisation a le type ES, EI ou EU qui sont associés respectivement à la spécialisation, à l'intersection et à l'union.

Les prédicats sont décrits sous la forme:

(P11 ET P12 ET ... P1n) OR
(P21 ET P22 ET ... P2m) OR
(Pj1 ET Pj2 ET ... Pjk)

où chaque prédicat simples est indiqué par Pjk et les groupes de prédicats simples par (...). Dans le catalogue, la représentation est faite par une famille de relations:



Le prédicat composé est représenté par un n-uplet de CAT_PCOMP et les (j) groupes de prédicats par autant (k) n-uplets de CAT_PS que necessaires. Les relations CAT-PVAL, CAT_PDOM, CAT_PAGR et CAT_PCOMP contiennent les informations particuliéres à chaque type de prédicat simples, elles son

décrites dans l'annexe A2.

Exemple:

```
type Invité : spec
        of Personne
        where invité = TRUE

        date_inv : time ;
        priorité : boolean
    end ;
```

CAT_D

D_c	dom_name	of_type	data_type
39	Personne	e	5
502	Invité	e	5

CAT_STRUC

D_c	R_c
39	37
502	500

CAT_R

R_c	rel_name	rel_type
37	Personne	EK
38	Personne_p	P
500	Invité	ES
501	Invité-p	P

CAT_COMP

R_comp_c	RE_c
38	37
501	500

CAT_A

A_c	R_c	D_c	att_name	E_ref	uk
46	38	3	invité	0	F
503	500	5	Invité_c	500	F
504	501	5	Invité_c	501	F
505	501	6	date_inv	0	F
506	501	3	priorité	0	F

CAT_PCOMP

PC_c	DC_c	manual
507	39	F

CAT_PS

PS_c	PC_c	refinement	grn
508	507	value	1

CAT_GEN

ID_result_c	ID_op_c	operator	PC_c
502	39	spec	507

CAT_PVAL

PS_c	A_c	operator	value
508	46	=	T

Pour les autres types de prédicats simples la représentation dans le catsalogue est analogue à l'exemple à l'exception des informations particuliéres à chaque type de prédicat simple.

5.3.12 Le catalogue et la base

A chaque définition d'un type classe correspond une classe dans la base de données. Le nom de cette classe est le même que le nom du type. A partir du type classe il est aussi possible définir de variables classe temporaires et de variables de programation comme décrit à la page 1. .

Aprés la génération de la représentation du schéma conceptuel d'une base sous forme de catalogue TIGRE il faut initialiser la base. Une procédure spécifique effectue la lecture des relations du catalogue TIGRE et produit les appels necessaires pour la création des relations et autres structures de données dans la base relationnelle. A ce moment les différentes zones de stockage pour les objets complexes sont crées. Tous les paramétres concernant le stockage des données sont fournis à cette procédure. A la fin de ce processus, une base correspondante au schéma est crée, toutes ses structures sont vides et les activités de manipulation de la base peuvent commencer.

L'inclusion des nouveaux types classe dans le schéma est possible. La modification dynamique d'un type classe pose des problèmes si cette modification entraine des repercussions sur les autres types classe . Aussi la modification ou destruction des types classe qui ont une extension dans la

base oblige tout an ensemble d'activités pour maintenir la cohèrence entre la base et le catalogue. Ces problèmes sont en train d'être modelisés en PROLOG. A partir des conclusions de ces travaux seront définies les procedures du serveur chargées de l'evolution dynamique du schéma.

Chapitre 6

CONCLUSION

6. Conclusion et futures extensions

Au long de cette thèse nous avons étudié la modélisation des applications bureautiques et le développement d'un modèle de données et d'un serveur de base de données capable de supporter ce genre d'application. L'étude des modèles de données avec une sémantique étendue et le developpement du modèle TIGRE nous a conduit aux conclusions suivantes:

1) L'inclusion de la sémantique explicite dans le modéle de données est un besoin pour le développement des application nouvelles. Cette sémantique peut être classée en deux groupes:

   - la sémantique associée aux nouveaux types de données,

   - la sémantique liée à la représentation de la realité

   la première correspond, dans le modèle TIGRE, aux types de données document et temps, la seconde aux concepts de généralisation et agrégation.

   L'extension des types permet l'intégration des nouvelles activités dans le système d'information automatisé, comme la manipulation des documents, ces applications étaient jusqu'ici indépendantes du système faute d'outils de modélisation et de manipulation adaptés. D'autre part ces applications étendues comprennent un ensemble majeur d'activités et sont, par conséquent, plus complexes que les applications traditionnelles. Cela se traduit par le besoin d'une meilleure représentation de la réalité. Cette représentation est assurée par l'emploi des types pour la définition d'un modèle Entité-Association et par l'inclusion des concepts de généralisation et d'agrégation.

Dans le cadre de la bureautique les nouveaux types de données permettent l'inclusion du traitement de texte et de la recherche bibliographique par l'emploi du concept de type document. La représentation explicite du concept de temps permet la modélistions des apllications de gestion où différentes activités sont définies par rapport à un calendrier.

2) Nous avons developpé un modèle de données généralisé à partir d'une étude approfondie de la bibliographie et en apportant notre experience personelle sur les besoins des applications bureautiques. Il faut utiliser une première version du prototype dans un environnement de travail et par des usagers autres que ceux qui ont conçu le modèle de données. Le but de cette utilisation doit être la verification de l'importance des concepts inclus dans le modèle pour la représntation de la réalité. En parallele il faut verifier quels sont les fonctions additionnelles qui peuvent avoir échappé à notre analyse. Cette validation est en plus necessaire car la richesse du modèle permet des solutions alternatives. Seulement la pratique peut indiquer quelles combinaisons de structures de données et d'outils de manipulation sont les plus adaptés aux besoins des applications.

L'architecture du serveur base de données a été conçue sur deux points principaux: la définition d'interfaces et l'emploi des concepts developpés dans RM/T pour faire la correspondance entre le modèle TIGRE et le modèle relationnel. La réalisation nous a permis de montrer qu'il est posible de réaliser des extensions sémantiques en utilisant un SGBD relationnel existant tout en maintenant l'indépendence entre les deux niveaux et en fournissant aux usagers un modèle de données homogéne. La pré-compilation des requêtes permet que l'execution des programmes soit faite directement sur la base

relationnelle sans la surcharge de multiples accés au catalogue TIGRE. De cette façon nous croyons que la performance de l'ensemble serveur/application sera acceptable.

Au-delà des propositions et de la réalisation actuelle il serait souhaitable de prolonger le travail dans les directions suivantes:

1) L'emploi du concept d'historique (section 2.2.2) dans le catalogue TIGRE pour permettre la modification dynamique du schéma de la base. De cette façon il serait possible de maintenir les différents états successifs du schéma de la base et traiter convenablement les requêtes. Pour cela il faut disposer dans le SGBD d'une logique ternaire.

2) L'emploi du type document pour faire la déscription des différents objets du catalogue. Une des utilités du schéma est la communication entre les specialistes en informatique et les usagers du système (fonction de dictionnaire de données). Cette extension permettra l'inclusion des descriptions textuelles et graphiques des objets composants le schéma. En plus de la simples enumeration des entités, associations et de leurs propriétés, il est possible d'inclure des informations sur l'origine et l'emploi des données, leur périodicité de mise à jour, les contraintes non représentables dans le langage de specification, entre autres. La pré-compilation des programes d'application doit génerer, dans le catalogue TIGRE, un ensemble d'informations sur les différents types d'accés aux objets de la base.

3) En plus de la définition des composants statiques du système d'information, il est envisageable d'inclure dans le catalogue les actions élémentaires réalisables sur

chaque entité ou association. Avec cette possibilité l'usager du SGBD écrira ses programmes d'application comme une composition d'actions pré-définies dans le modèle TIGRE. Cette application du concept de types abstraits de données aux types classe améliorera l'indépendance des données et assurera l'integrité des mise à jour par l'inclusion des restrictions d'intégrité.

4) L'extension sémantique proposée dans le modèle TIGRE permet une meilleur représentation de la réalité. Cette richesse sémantique permet l'intégration au schéma des informations qui dans les systèmes traditionnels sont implicites dans les programmes d'application; cependant la conception générale du système demeure complexe. Il nous parait important de mettre en évidence la necessité de l'emploi des outils de Conception Assisté par Ordinateur pour le projet de bases de données généralisées. La disponibilité de ces outils , ensemble avec les extensions du catalogue TIGRE proposées dans les items précedents, apporteront une contribuition significative au développement des applications généralisées.

Comme conclusion finale de notre travail, nous croyons que l'étude des modèles de données sémantiques, la proposition du modèle TIGRE, ses extensions et la réalisation du logiciel associé au catalogue a fourni un noyau important pour la suite de la recherche. A partir de ce noyau il sera possible l'acquerir de l'experience sur l'application bureautique et développer de nouvelles fonctions.

ANNEXES

ANNEXE A1

SYNTAXE DES ENONCEES DE DEFINITION LAMBDA

<schema_def> ::= 'Define' <db_name> <body_schema_def> 'end'

<body_schema_def> ::= <constant_def> <type_def> <var_def>

<constant_def> ::= 'constant' <one_cons_def> (';' <one_cons_def>)* ! <time_const> ! <empty>

<one_const_def> ::= <const_id> '=' <value>

<value> ::= <numeric_constant> ! / (+!-) / <unsigned_integer> ! <string>

<time_const> ::= 'time_constant' <one_time_const_def> (';' <one_time_const-def>)*

<one_time_const_def> ::= <time_const_id> '=' <unsigned_integer> <time_element>

<time_const_id> ::= <identifier>

<type_def> ::= 'type' <one_type_def> (';' /'type'/ <one_type_def>)*

<one_type_def> ::= <type_id> ':' <persistency_def> <structure_def>

<persistency_def> ::= <'dynamic' /'each' (<periodic_time_type> ! <unsigned_integer>)/ /'last' <unsigned_integer>/ ! <empty>

<structure_def> ::= <type_id> ':' (<basic_type> !

```
            <constructed_type> ! <class_type>)

<var_def> ::= <dbvar_def> <temp_var_def> <var_def> ! <empty>

<dbvar> ::= 'dbvar'   <one_var_def>     (    ';'     /'dbvar'/
            <one_var_def>  )*

<temp_var_def>  ::=  'var'  <one_var_def>   (   ';'    /'var'/
            <one_var_def>  )*

<one_var_def> ::= <var_id> ':'  <type_id>

<var_id> ::= <identifier>

<type_id> ::= <identifier>

<basic-type> ::= <basic_simple_type> ! <basic_restricted_type>

<basic_simple_type> ::=  'integer'  !  'real'  !  'boolean'  !
            'string' '(' <unsigned_integer> ')' ! 'time'

<basic_restricted_type> ::= <scalar_type> ! <interval_type>  !
            <time_interval_type>     !    <time_rest_type>    !
            <periodic_time_type>

<scalar_type> ::= '(' <scalar_id> ( ',' <scalar_id> )+ ')'

<scalar_id> ::= <identifier>

<interval_type>    ::=     '('     <unsigned_integer>     '..'
            <unsigned_integer> ')'

<time_interval_type> ::= '(' <time> '..' <time> ')'

<time_rest_type> ::= 'time' '>' ('month' ! 'day'  !  'hour'  !
```

```
            'minute' | 'second')

<periodic_time_type>  ::= <periodic_time>  |  <week_day>  |
            <periodic_interval>

<periodic_time> ::= ('year' | 'month' | 'day' | 'hour' |
            'minute') '>' <time_rest_type> | <time_element>

<week_day> ::= 'su' | 'mo' | 'tu' | 'we' | 'th' | 'fr' | 'sa'

<periodic_interval>   ::=   '('   <periodic_date>   '..'
            <periodic_date> ')'

<periodic_date> ::= <periodic_time_id> ':=' <time_value>

<time_value> ::= /* instance of <periodic_time> */

<time_element> ::= 'year' / 'month' / 'day' / 'hour' /
            'minute' / 'seconde'

<constructed_type>  ::=  <record_type>  |  <array_type>  |
            <document_type>

<record_type> ::= 'record' <field_list>

<field_list> ::= <field_def> (';' <field_def>)*

<field_def> ::= <identifier> ':' <basic_type>

<array_type> ::= 'array' '(' <integer> ')' 'of' <basic_type>

<document> ::= 'document' <document_def> 'end'

<document_def> ::= <structure_def> /<constant_def>/
```

```
<structure_def> ::= 'structure' <structure_constructor>  'end'
          ';'

<structure_constructor>  ::=  'begin'   <structured_elements>+
          'end'  ';'  !  'list'  '(' <min> ',' <max> ')' 'of'
          <structure_constructor>  !  'case'  <case_list>  !
          <document_type_id>  '.'  'structure'  ! <unit_type>
          /':=' (<const_id>)/

<choice_list>  ::=  ((<unsigned_integer>  !  <case_name>)  ':'
          (structure_constructor>)+

<case_name> ::= <identifier>

<structure_element> ::= <node_name> /<document_attributs>/ ':'
          <structure_constructor>

<document_attributs>   ::=   'attr'    <attribut_name>    (';'
          <attribut_name>)*

<attribut_name> ::= 'private' ! 'key_word'  !  'higlithing'  !
          'justification' ...

<document_type_id> ::= <identifier>

<parameter_id> ::= <identifier>

<unit_type> ::= 'text' ! 'graphics' ! 'image'  !  'formula'  !
          'table'  !  'voice'  !  'program'  !  'reference' !
          'unit'

<constant_def> ::= 'constant' <constant_decl>+ 'end' ';'

<constant_decl>          ::=          <unit_constant>          !
          <associated_element_constant>
```

```
<const_id> ::= <identifier>

<min> ::= <unsigned_integer>

<max> ::= <unsigned_integer>

<unit_constant> ::= 'unit' <contant_id> <unit_constant_decl>

<unit_contant_decl> ::= ';' <unity_type>  ':='  (<constant>  !
          <constant_ref>)

<constant> ::= <constant_string> ! <image> ! <graphics> !  ...

<constant_ref> ::= <lambda_selection_expression>

<associated_element_const>    ::=    <associeted_element_name>
          <contant-id> <composed_const_decl>

<composed_const_decl>    ::=    ':'    'begin'    (<node_name>
          !<compsed_const_decl>)*   'end'   !   <':'   'list'
          (<unsigned_integer> <composed_const_decl>)* 'end' !
          ':'     <case_name>     <composed_cons_decl>      !
          <unit_const_decl>      !      <unit_type>     /':='
          <function_name> /'(' <call_parameters> ')'//

<associeted_element> ::= 'footnote'  !  'bibliographic_ref'  !
          'illustration'  !  'page_header'  !  'parameter' !
          'function'

<constant_id> ::= <identifier>

<function_name> ::= <identifier>

<call_parameters> ::= <identifier> (';' <identifier>)*
```

<class_type> ::= <non_derived_class_type> !
<derived_class_type>

<non_derived_class_type> ::= <non_derived_entity_type> !
<non_derived_relationship_type>

<derived_class_type> ::= <generalization_type> !
<agregation_type>

<generalization_type> ::= <specialization_type> ! <union_type>
! <intersection_type>

<agregation_type> ::= <entity_agregation_type> !
<relationship_agregation_type>

<non_derived_entity_type> ::= 'entity' <attributes> 'end'

<attributs> ::= /<key_part>/ <attribut_list>

<key_part> ::= 'key' <attribut_list> 'end_key'

<attribut_list> ::= <attribut_def> /';' <attribut_list>/ !
empty

<attribut_def> ::= <attribut_id> ':' <attribut_type>

<attribut_id> ::= <identifier>

<attribut_type> ::= <type_id> ! <basic_type>

<non_derived_relationship_type> ::= 'relationship' <role_part>
/<attribut_list>/ 'end'

<role_part> ::= 'between' <role> 'and' <role>

```
<role> ::= <entity_type_id> /':' <role_id> <cardinality>/

<entity_type_id> ::= <identifier>

<role_id> ::= <identifier>

<cardinality> ::= '(' <unsigned_integer> '..' <upper-bound>
          ')' /'incremental'/

<upper-bound> ::= <unsigned_integer> | '*'

<specialization_type>    ::=    <specialized-entity_type>    |
          <specialized_relationship_type>

<specialized-entity_type>       ::=       'specialization-of'
          <entity_type_id>    <restriction_pred>   /'manual'/
          /<attribut_list>/

<restriction_pred> ::= 'where' <predicate> ('or' <predicate>)*

<predicate> ::= <simple_predicate> ('and' <simple_predicate>)*

<simple_predicate> ::= <attribut_id> (<scalar_comparator> |
          <set_comparator>) <expression> | <attribut_id> ":"
          (<scalar_type> | <interval_type>) | <role_id> 'of'
          <entity_type_id>             |             'part_of'
          <agregate_entity_type_id>

<aggregate_entity_type_id> ::= <identifier>

<specialized_relationship_type>    ::=    'specialization_of'
          <relationship_type_id> <retriction_pred> /'manual'/
          <role_part> /<attribut_list>/ 'end'

<relationship_type_id> ::= <identifier>
```

```
<union_type>    ::=    'union_of'     <union_entity_type_list>
          /<attribut_list>/ 'end'

<union_entity_type_list>     ::=      <restricted_entity_type>
          /'manual'/      ('and'     <restricted_entity_type>
          /'manual'/)+

<restricted_entity_type>          ::=          <entity_type_id>
          <restriction-pred>

<intersection_type>           ::=            'intersection_of'
          <intersection_entity_type_list>   /<attribut_list>/
          'end'

<intersection_entity_type_list>  ::=  <restricted_entity_type>
          ('and' <restricted_entity_type>)+ /manual/

<entity_agregation_type>      ::=       'entity_agregation_of'
          <entity_type_id>        <cardinality>        ('and'
          <entity_type_id> <cardinality>)*  /<attribut_list>/
          'end'

<relationship_agregation_type>  ::=   'relationship_agregation
          of'  <relationship_type_id> /<attribut_list>/ 'end'

<time_element> ::=  'year'  !  'month'  !  'day'  !  'hour'  !
          'minute' ! 'seconde'

<time> ::= <yar> '/' <month> '/' <day> ' ' <hour> ':' <minute>
          ':' <seconde>

<year> , <month> , <day> , <hour> , <minute> ,  <second>   ::=
          <unsigned_integer>

<scalar_comparator> ::= '=' ! '<>' ! '>' ! '>=' ! '<' ! '<='
```

```
<set_comparator> ::= '=' | '<>' | 'contains' | 'contained_in'

<expression> ::= <numeric_constant> | <constant_string> |
        <variable>

<numeric_constant> ::= /('+' | '-')/ <unsigned_number>

<unsigned_number> ::= <unsigned_integer> '.'
        /<unsigned_integer>/

<unsigned_integer> ::= <digit>+

<constant_string> ::= '"' <character>+ '"'

<identifier> ::= <letter> (<letter> | <digit>)*

<empty> ::=
```

ANNEXE A2

DESCRIPTION DES RELATIONS DU CATALOGUE TIGRE

I - Domaines

CAT_D (db_c, d_c, dom_name, of_type, data_type)

CAT_D est la relation principalle pour la représentation des domaines. Le premier accès au catalogue concernant un domaine est fait à cette relation. A partir des informations contenues dans CAT_D il est possible savoir s'il faut faire des nouveaux accès au catalogue pour compléter la description du domaine.

db_c — surrogate de la base de données

d_c — surrogate du domaine - clé de la relation

dom_name — nom du domaine - valeur "notnamed" si la définition du type de domaine est faite ensemble avec celle du attribut.

of_type — type du domaine - valeurs: integer, réal, boolean, string, scalar, interval, list, record et document ou le nom d'un operateur pour la construction des types_classe.

valeurs:

I integer	R real	B boolean
S string	L list	D document
s scalar	r record	i interval
e entity	a relationship	

data_type — est le type simples sur leqel le domaine est défini. Son contenu pet être un type reconue par le SGBD sousjacent, un code special pour les surrogates ou la spécification du type classe.

valeurs:

1 integer	2 real	3 boolean
4 string	5 E-domain	6 time

II - Domaine intervalle

CAT_INTD (db_c, d_c, min, max)

Cette relation sert à maintenir les limites de cardinalité des domaines du type intervalle. Après l'accès à la relation CAT_D, si le domaine est du type intervalle, il y faut accéder pour obtenir les limites supérieure et inferieure de l'intervalle.

db_c surrogate de la base de données

d_c surrogate du domaine - clé de la relation

min entier qui donne le limite inférieur du intervalle

max entier qui donne le limite supérieur du intervalle

III - Domaine scalaire

CAT_SCAD (db_c, d_c, element)

Cette relation est destinée à maintenir l'ensemble des valeurs possibles d'un domaine associé au type scalaire. Elle aura un tuple par élément - une chaine de caractères - du domaine.

db_c surrogate de la base de données

d_c surrogate du domaine - clé de la relation

element valeur d'un élément du domaine scalaire

IV - Relations

CAT_R (db_c, r_c, rel_name, rel_type)

Le rôle des tuples de CAT_R est la représentation dans le

catalogue des définitions des types construits liste, enregistrement et des types_classe. A chaque relation de la base correspond un tuple dans la relation CAT_R. Cette correspondence est décrite dans la section 5.3.X .

db_c surrogate de la base de données

r_c surrogate de la relation - clé de la relation

rel_name nom de la relation

rel_type décrit le type de la relation

valeurs:

E une relation E

EA un type association

EK un type entité independente - "inner kernel" en RMT

ER un type enregistrement

ED un type document

EL un type liste

EE un type entité derivé par derivation d'entités

ES un type entité derivé par spécialisation

EI un type entité derivé par intersection

EV un type entité derivé par union

AA un type entité derivé par aggregation associative

A une relation associative, c'est à dire qui lie des faites des types_entités liées semantiquement par un type association

P une relation qui a pour fonction la représentation des attributs d'un fait d'un type classe ou d'un fait d'un type liste ou enregistrement

G une relation qui a pour fonction mantenir l'information sur quels sont les types appartenant à une agrégation d'entités

V - Atributs

CAT_A (db_c, a_c, r_c, d_c, att_name, E_ref, user_key)

A chaque attribut d'une relation de la base correspond un tuple dans CAT_A.

db_c surrogate de la base de données

a_c surrogate de l'attribut - clé de la relation

r_c surrogate de la relation à laquelle l'attribut appartient

d_c surrogate du domaine sur lequel l'attribut est défini

att_name nom de l'attribut

E_ref indique le surrogate de la reletion_E qui doit contenir les valeurs de l'attribut si celui-ci est défini sur un domaine_E

user_key indique si le attribut appartient a la clé definie par l'usager

VI - Domaine défini sur le type chaine de caractères

CAT_STRING (db_c, d_c, length)

La relation CAT_STRING donne la longeur maximale admissible pour chaque élément du type chaine de caractères

db_c surrogate de la base de données

d_c surrogate du domaine - clé du la relation

length longeur maximale admissible

VII - Domaines construites et classes

CAT_STRUC (db_c, d_c, r_c)

Cette relation lie la représentation dans CAT_D d'un type classe ou construit, identifié par d_c, avec sa represéntation dans CAT_R, identifié par r_c. Ce dernier surrogate fait reférence à une relation_E.

db_c surrogate de la base de données

d_c surrogate du domaine - clé de la relation

r_c suorrogate de la relation_E correspondante

VIII - Liaison entre la relation_E et les relations_P ou A

CAT_COMP (db_c, r_comp_c, re_c)

Elle associe une relation_E avec une ou plusieurs relations_P (dans la premiére version du prototype existe seulemente une). Elle fait aussi la association avec les relations_A.

db_c surrogate de la base de données

r_comp_c surrogate de la relation P ou A - clé de la relation

re_c surrogate de la relation_E

IX - Type liste

CAT_LIST (db_c, d_c, n_of_elements)

Cette relation mantient la cardinalité de la liste.

db_c surrogate de la base de données

d_c surrogate du domaine - clé de la relation

n_of_elements cardinalité de la liste

X - Types documents

CAT_DOC (db_c, d_c, list_c)

Cette relation stocke la référence a la définition d'un document.

db_c surrogate de la base de données

d_c surrogate du domaine - clé de la relation

list_c référence à la liste parenthesé définissant le type document

XI - Associations

CAT_DESIG (db_c, dr_c, de_c, role, min, max)

Pour la réprésentation des types_associations, les informations sur les rôles, les types_entités associés et la cardinalité de l'association sont mantenues par cette relation.

db_c surrogate de la base de données

dr_c surrogate du type association

de_c surrogate d'un des types_entités associés

role nom du rôle du type entité identifié par de_c.

min limite inférieur de la cardinalité

max limite supérieur de la cardinalité

XII - Prédicats simples

CAT_PS (db_c, ps_c, gr_n, refinement, p_c)

Les prédicats simples sont représentés par cette relation. L'attribut "refinement" indique le type de raffinement associé a ce prédicat simples. Cette information indique la relation du catalogue à laquelle il faut accéder pour compléter la description du prédicat. Les différentes raffinements sont:

par valeur, par domaine, par association et par agrégation.

db_c surrogate de la base de données

ps_c surrogate du prédica simples - clé de la relation

p_c surrogate du prédicat composé auquel le prédicat simples appartient

refinement type de raffinement du prédicat simples

gr_n numéro du groupe dans la forme normale disjonctive du prédicat p_c auquel le prédicat simples ps-c appartient.

XIII - Raffinement par valeur

CAT_PVAL (db_c, ps_c, a_c, operator, value)

db_c surrogate de la base de données

ps_c surrogate du prédicat simples _ clé de la relation

a_c surrogate de l'attribut sur lequel opere le predicat référencé par le surrogate ps_c

operator un des opérateurs admissibles par ce type d'attribut

value chaine de caractères contenant un valeur compatible avec le type d'attribut

XIV - Raffinement par domaine

CAT_PDOM (db_c, ps_c, a_c, d_c)

db_c surrogate de la base de données

p_s surrogate du prédicat - clé de la relation

a_c surrogate de l'attribut sur lequel opere le prédicat simples référencé par ps_c

d_c surrogate du domaine intervalle ou scalaire

XV - Raffinement par aggrégation

CAT_PAGR (db_c, ps_c, d_c)

db_c surrogate de la base de données

ps_c surrogate du prédicat simples

d_c surrogate du domaine correspondante au type entité agregé

XVI - Raffinement par association

CAT_PASS (db_c, ps_c, d_c,role)

db_c surrogate de la base de données

ps_c surrogate du predicat simples

d-c surrogate du domaine correspondante au type entité

role nome du rôle

XVII - Les prédicats de restriction composés

CAT_PCOMP (db_c, p_c, d_c, manual)

db_c surrogate de la base de données

p_c surrogate du prédicat composé - clé de la relation

d_c surrogate du domaine correspondante au type classe sur lequel opére le prédicat

manual le valeur true indique qu'en plus de satisfaire le prédicat indiqué par la combinaison de prédicats simples il faut faire explicitemente l'inclusion des faites

XVIII - Dérivation par généralisation

CAT_GEN (db_c, d_result_c, d_op_c operator, p_c)

db_c surrogate de la base de données

d_result_c surrogate du domaine correspondante au type classe derivé

d_op_c surrogate du domaine correspondante au type classe opérand

operator l'opérateur employé pour faire la dérivation

p_c surrogate du prédicat de restriction composé du domaine correspondante au type dérivé ou valeur nul s'il n'existe pas

XIX - Higrarchie de dérivation

CAT-ANT (db_c, da_c, att_inc)

Cette relation facilite la recherche des ancêtres d'un type

classe dérivée donnée lors de la traduction de la requête Lambda. Elle faite la réprésentation du graphe de dérivation.

db_c surrogate de la base de données

d_c surrogate du domaine identifiant le type casse

ant_c surrogate du domaine identifiant le type classe pére

att_inc indique si les attributs de ce ancêtre doivent être inclus dans l'ensemble des attributs du domaine correspondante au type classe derivé. Les régles d'inclusion sont fournies par la définition des operateurs de specialisation , union et intersection.

XX - Agrégation d'entités

CAT_EAGG (db_c, r_agg_c, d_comp_c, min, max)

Cette relation a pour but stocker les types d'entités qui peuvent appartenir à l'aggregation et ses contraintes de cardinalité.

db_c surrogate de la base de données

d_agg_c surrogate qui identifie le type entité agrégé - clé de la relation

d_comp_c surrogate qui identifie un type entité appartenant à l'aggregation

min limite inférieur de la cardinalité

max limite supérieur de la cardinalité

XXI - Agrégation associative

CAT_AAGG (db_c, d_agg_c, d_comp_c)

cette relation associe un type entité derivé par agrégation associative et le type association sousjacent

db_c surrogate de la base de données

d_agg_c surrogate du domaine correspondante au type entité derivé - clé de la relation

d_comp_c surrogate du domaine correspondante au type association

ANNEXE A3

DEFINITION D'UN EXEMPLE

IFIP

"COMPARATIVE REVIEW OF INFORMATION SYSTEMS DESIGN METHODOLOGIES"

Définition du problème

1. Introduction

Une "IFIP Working Conference" est une conférence internationale qui a pour but de mettre ensemble des spécialistes de tous les pays affiliés à l'IFIP pour discuter d'un certain sujet technique. Ce sujet interesse specifiquement un ou plusieurs Groupes de Travail de L'IFIP. L'organisation, qui doit être prise, est de faire une conférence par invitations et non ouverte au public en général. Pour une conférence de ce type il faut, d'une certaine façon, assurer que tous les membres des Groupes de Travail et des Comités Techniques de l'IFIP soient invités. D'autre part, il est important d'assurer un nombre suffisant de participants sans dépasser la limite maximale permise par les moyens disponibles.

La politique de l'IFIP pour une Conférence de Travail repose sur deux comités: le Comité de Programme et le Comité d'Organisation. Le Comité de Programme gére le contenu technique de la conférence. Le Comité d'Organisation gére les aspects financiers, la gestion des lieux de la conférence , les invitations et la publicité. Les comités doivent nécessairemment travailler ensemble et avoir des informations communes maintenues de façon cohérente.

2. Le système d'informations qui doit être conçu

Le système d'informations à concevoir doit fournir le support pour le Comité de Programme et pour le Comité d'Organisation.

Les activités suivantes des comités doivent être supportées:
Comité de Programme

CP-1. Préparation de la liste de personnes à qui envoyer l'appel aux communications

CP-2. Enregistrement des lettres d'intention reçues en réponse aux appels aux communications.

CP-3. Enregistrement des articles dès leur reception.

CP-4. Distribution des articles aux referees.

CP-5. Reception des rapports des referees et sélection des articles pour inclusion dans le programme.

CP-6. Groupement des articles sélectionnés en sessions et choix du président pour chaque session.

Comité d'Organisation

CO-1. Préparation de la liste des personnes invitées à la conférence.

CO-2. Envoie d'une invitation prioritaire aux répresentants nationaux, membres du Groupe de Travail et des Groupes associés.

CO-3. Assurer que chaque auteur d'un article accepté soit invité.

CO-4. Assurer que chaque auteur d'un article rejeté soit invité.

CO-5. Eviter l'emission d'invitations doubles.

CO-6. Enregistrer l'acceptation des invitations.

CO-7. Production de la liste finale de participants.

3. Limites du système

Il faut noter que les aspects financiers du travail du Comité d'Organisation, la planification des réunions des groupes, la reservation d'hotel pour les participants et la préparation des proceedings on été omis de cet exercise.

SCHEMA POUR L'EXEMPLE

```
define exemple

type jour : time > hour ;

type nom_de_pays : string (36) ;

type t_local : record

          ville : string (20) ;
          pays  : nom_de_pays
    end ;

type t_adresse : record

          n           : integer ;
          rue         : string (36) ;
          code_postal : integer ;
          ville       : string (20) ;
          pays        : nom_de_pays
    end ;

type Personne : entity

      key IFIP_n   : integer ;
          nom      : string (20) ;
          adresse  : t_adresse;
          invité   : boolean
    end ;

type Invité : specialization_of Personne
              where invité = TRUE ;

          date_inv : jour ;
```

```
        reponse   : record
                        decision : boolean ;
                        date_rec : jour
                    end ;
        priorité : boolean
    end ;

type Participant : specialization_of Invité
                where reponse = TRUE and manual

        inscription : record
                          date  : jour ;
                          value : real
                      end
    end ;

type Referee : specialization_of Personne
               manual ;

type Prob_auteur : specialization_of Personne
                   manual ;

        date_rec_int : jour ;
        titre        : string (40) ;
        subjet       : string (20) ;
        date_rec_art : jour
    end ;

type Auteur : specialization_of Prob_auteur
              where date_rec_art <> NULL

type Président : specialization_of Personne
                 manual ;
```

```
type Rep_national : specialization_of Personne
                    manual

        pays : nom_de_pays
   end ;

type Article : entity

        numéro          : integer ;
        titre           : string (40) ;
        date_reception  : jour ;
        date_decision   : jour ;
        date_reponse    : jour ;
        decision        : boolean
   end;

type Article_accepté : specialization_of Article
                       where decision = TRUE ;

        contenu : structure
                      .
                      .
                      .
   end ;

type Session : entity

        session-n : integer ;
        titre     : string (40) ;
        lieu      : string (20) ;
        horaire   : time
   end ;

type Groupe_de_travail : entity_aggregation_of Personne (2,*)
```

```
      key GT_numéro : real ;
          GT_nom    : string (80)
   end;

type Comité_technique : entity

      key CT_numéro : integer ;
          CT_nom    : string (80)
   end ;

type Conference_de_travail : entity

          nom      : string (80) ;
          ville    : record
                        nom  : string (20) ;
                        pays : nom_de_pays
                     end ;
          période : (jour .. jour)
   end ;

type Appel_aux_communications : entity

          version : (préliminaire, définitive, rappel) ;
          contenu : document
                        structure
                            corps : text
                        end
   end ;

type Envoi : relationship
          between Personne (1,1) and
          and Appel_aux_communications (1,3) ;

          date_env : list_of (3) jour
   end ;
```

```
type Composition : relationship
          between Comité_technique
          and     Personne : membre ;

type Appartenance : relationship
        between Comité_technique
        and     Groupe_de_travail (1,*) ;

type Organisation : relationship
        between Groupe_de_travail (1,*)
        and     Conference_de_travail ;

type Composition : relationship
        between Conférence_de_travail
        and Session ;

type Art-session : relationship
        between Session
        and Article_accepté ;

        ordre : integer ;
        heure : time
   end ;

type Authorship : relationship
        between Auteur (1,*)
        and Article (1,*) ;

        auteur_no : integer
   end ;

type Présidence : relationship
        between Chairman (1,1)
        and Session (1,1) ;
```

```
type Attribution : relationship
        between Referee (2,3)
        and Article (0,5) ;

        date_att     : jour ;
        evaluation   : boolean ;
        commentaire : document
                          structure
                              comm : text
                      end
   end ;

type Articles_par_session : relationship_aggregation_of
                            Art_sess ;

type Proceedings : entity_aggregation_of
        Articles_par_session

        contenu : documment
                      structure
                          .
                          .
                          .
                  end
   end ;

end.
```

ANNEXE A4

LE CATALOGUE GENERE PAR UN EXEMPLE

```
***** define exemple
CAT_Db( Dbc=7 dbname=exemple )
***** type nom_pays : string (36) ;
CAT_D( Dbc=7 Dc=8 domname=nom_pays oftype=s datatype=4)
CAT_STRING( Dbc=7 Dc=8 length=36)
***** t_datenv : list_of (3) string (8) ;
CAT_D( Dbc=7 Dc=9 domname=t_datenv oftype=L datatype=5)
CAT_LIST( Dbc=7 Dc=9 noofelements=3)
CAT_R( Dbc=7 Rc=10 relname=t_datenv reltype=EL)
CAT_R( Dbc=7 Rc=11 relname=t_datenv_p reltype=P )
CAT_COMP( Dbc=7 Rcompc=11 REc=10)
CAT_A( Bdc=7 Ac=12 Rc=10 Dc=5 atn=t_datenv_c Eref=10 uk=F)
CAT_A( Bdc=7 Ac=13 Rc=11 Dc=5 atn=t_datenv_c Eref=10 uk=F)
CAT_STRUC( Dbc=7 Dc=9 Rc=10)
CAT_A( Bdc=7 Ac=14 Rc=11 Dc=1 atn=order Eref=0 uk=F)
CAT_D( Dbc=7 Dc=16 domname=notnamed oftype=S datatype=4)
CAT_A( Bdc=7 Ac=15 Rc=11 Dc=16 atn=value Eref=0 uk=F)
CAT_STRING( Dbc=7 Dc=16 length=8)
***** t_loc : record
CAT_D( Dbc=7 Dc=17 domname=t_loc oftype=R datatype=5)
***** ville : string (20) ;
CAT_STRUC( Dbc=7 Dc=17 Rc=19)
CAT_R( Dbc=7 Rc=19 relname=t_loc reltype=ER)
CAT_R( Dbc=7 Rc=18 relname=t_loc_p reltype=P )
CAT_COMP( Dbc=7 Rcompc=18 REc=19)
CAT_A( Bdc=7 Ac=20 Rc=19 Dc=5 atn=t_loc_c Eref=19 uk=F)
CAT_A( Bdc=7 Ac=21 Rc=18 Dc=5 atn=t_loc_c Eref=19 uk=F)
CAT_D( Dbc=7 Dc=23 domname=notnamed oftype=S datatype=4)
CAT_A( Bdc=7 Ac=22 Rc=18 Dc=23 atn=ville Eref=0 uk=F)
CAT_STRING( Dbc=7 Dc=23 length=20)
***** pays : nom_pays
CAT_A( Bdc=7 Ac=24 Rc=18 Dc=8 atn=pays Eref=0 uk=F)
```

```
***** end ;
***** t_adresse : record
CAT_D( Dbc=7 Dc=25 domname=t_adresse oftype=R datatype=5)
***** n : integer ;
CAT_STRUC( Dbc=7 Dc=25 Rc=27)
CAT_R( Dbc=7 Rc=27 relname=t_adresse reltype=ER)
CAT_R( Dbc=7 Rc=26 relname=t_adresse_p reltype=P )
CAT_COMP( Dbc=7 Rcompc=26 REc=27)
CAT_A( Bdc=7 Ac=28 Rc=27 Dc=5 atn=t_adresse_c Eref=27 uk=F)
CAT_A( Bdc=7 Ac=29 Rc=26 Dc=5 atn=t_adresse_c Eref=27 uk=F)
CAT_A( Bdc=7 Ac=30 Rc=26 Dc=1 atn=n Eref=0 uk=F)
***** rue : string (36) ;
CAT_D( Dbc=7 Dc=32 domname=notnamed oftype=S datatype=4)
CAT_A( Bdc=7 Ac=31 Rc=26 Dc=32 atn=rue Eref=0 uk=F)
CAT_STRING( Dbc=7 Dc=32 length=36)
***** cp : integer ;
CAT_A( Bdc=7 Ac=33 Rc=26 Dc=1 atn=cp Eref=0 uk=F)
***** ville : string (20) ;
CAT_D( Dbc=7 Dc=35 domname=notnamed oftype=S datatype=4)
CAT_A( Bdc=7 Ac=34 Rc=26 Dc=35 atn=ville Eref=0 uk=F)
CAT_STRING( Dbc=7 Dc=35 length=20)
***** pays : nom_pays
CAT_A( Bdc=7 Ac=36 Rc=26 Dc=8 atn=pays Eref=0 uk=F)
***** end ;
***** Personne : entity
CAT_R( Dbc=7 Rc=37 relname=Personne reltype=EK)
CAT_D( Dbc=7 Dc=39 domname=Personne oftype=e datatype=5)
CAT_STRUC( Dbc=7 Dc=39 Rc=37)
CAT_R( Dbc=7 Rc=38 relname=Personne_p reltype=P )
CAT_COMP( Dbc=7 Rcompc=38 REc=37)
CAT_A( Bdc=7 Ac=40 Rc=37 Dc=5 atn=Personne_c Eref=37 uk=F)
CAT_A( Bdc=7 Ac=41 Rc=38 Dc=5 atn=Personne_c Eref=37 uk=F)
***** key IFIP_n : integer ;
CAT_A( Bdc=7 Ac=42 Rc=38 Dc=1 atn=IFIP_n Eref=0 uk=T )
***** nom : string (20) ;
```

```
CAT_D( Dbc=7 Dc=44 domname=notnamed oftype=S datatype=4)
CAT_A( Bdc=7 Ac=43 Rc=38 Dc=44 atn=nom Eref=0 uk=F)
CAT_STRING( Dbc=7 Dc=44 length=20)
***** adresse : t_adresse ;
CAT_A( Bdc=7 Ac=45 Rc=38 Dc=25 atn=adresse Eref=27 uk=F)
***** invite : boolean
CAT_A( Bdc=7 Ac=46 Rc=38 Dc=3 atn=invite Eref=0 uk=F)
***** end ;
***** Article : entity
CAT_R( Dbc=7 Rc=47 relname=Article reltype=EK)
CAT_D( Dbc=7 Dc=49 domname=Article oftype=e datatype=5)
CAT_STRUC( Dbc=7 Dc=49 Rc=47)
CAT_R( Dbc=7 Rc=48 relname=Article_p reltype=P )
CAT_COMP( Dbc=7 Rcompc=48 REc=47)
CAT_A( Bdc=7 Ac=50 Rc=47 Dc=5 atn=Article_c Eref=47 uk=F)
CAT_A( Bdc=7 Ac=51 Rc=48 Dc=5 atn=Article_c Eref=47 uk=F)
***** numero : integer ;
CAT_A( Bdc=7 Ac=52 Rc=48 Dc=1 atn=numero Eref=0 uk=F)
***** titre : string (40) ;
CAT_D( Dbc=7 Dc=54 domname=notnamed oftype=S datatype=4)
CAT_A( Bdc=7 Ac=53 Rc=48 Dc=54 atn=titre Eref=0 uk=F)
CAT_STRING( Dbc=7 Dc=54 length=40)
***** date_rec : time ;
CAT_A( Bdc=7 Ac=55 Rc=48 Dc=6 atn=date_rec Eref=0 uk=F)
***** date_dec : time ;
CAT_A( Bdc=7 Ac=56 Rc=48 Dc=6 atn=date_dec Eref=0 uk=F)
***** date_rep : time ;
CAT_A( Bdc=7 Ac=57 Rc=48 Dc=6 atn=date_rep Eref=0 uk=F)
***** decision : boolean
CAT_A( Bdc=7 Ac=58 Rc=48 Dc=3 atn=decision Eref=0 uk=F)
***** end ;
***** Authship : relship
CAT_R( Dbc=7 Rc=59 relname=Authship reltype=EA)
CAT_D( Dbc=7 Dc=61 domname=Authship oftype=a datatype=5)
CAT_STRUC( Dbc=7 Dc=61 Rc=59)
CAT_R( Dbc=7 Rc=60 relname=Authship_p reltype=P )
```

```
CAT_COMP( Dbc=7 Rcompc=60 REc=59)
CAT_A( Bdc=7 Ac=62 Rc=59 Dc=5 atn=Authship_c Eref=59 uk=F)
CAT_A( Bdc=7 Ac=63 Rc=60 Dc=5 atn=Authship_c Eref=59 uk=F)
***** between Personne : aut (1,10)
CAT_R( Dbc=7 Rc=64 relname=Authship_d reltype=A )
CAT_COMP( Dbc=7 Rcompc=64 REc=59)
CAT_A( Bdc=7 Ac=65 Rc=64 Dc=5 atn=Authship_c Eref=59 uk=F)
CAT_A( Bdc=7 Ac=66 Rc=64 Dc=5 atn=Personne_c Eref=37 uk=F)
CAT_DESIG( Dbc=7 RRc=59 REc=37 rolename=aut min=1 max=10)
***** and Article : art ;
CAT_A( Bdc=7 Ac=67 Rc=64 Dc=5 atn=Article_c Eref=47 uk=F)
CAT_DESIG( Dbc=7 RRc=59 REc=47 rolename=art min=1 max=999)
***** auteur_no : integer
CAT_A( Bdc=7 Ac=68 Rc=60 Dc=1 atn=auteur_no Eref=0 uk=F)
***** end ;
***** Appel : entity
CAT_R( Dbc=7 Rc=69 relname=Appel reltype=EK)
CAT_D( Dbc=7 Dc=71 domname=Appel oftype=e datatype=5)
CAT_STRUC( Dbc=7 Dc=71 Rc=69)
CAT_R( Dbc=7 Rc=70 relname=Appel_p reltype=P )
CAT_COMP( Dbc=7 Rcompc=70 REc=69)
CAT_A( Bdc=7 Ac=72 Rc=69 Dc=5 atn=Appel_c Eref=69 uk=F)
CAT_A( Bdc=7 Ac=73 Rc=70 Dc=5 atn=Appel_c Eref=69 uk=F)
***** version : (prelim, defin, rappel) ;
CAT_SCAD( Dbc=7 Dc=75 element=prelim )
CAT_A( Bdc=7 Ac=74 Rc=70 Dc=75 atn=version Eref=0 uk=F)
CAT_D( Dbc=7 Dc=75 domname=notnamed oftype=s datatype=4)
CAT_SCAD( Dbc=7 Dc=75 element=defin )
CAT_SCAD( Dbc=7 Dc=75 element=rappel )
***** dat_env : t_datenv ;
CAT_A( Bdc=7 Ac=76 Rc=70 Dc=9 atn=dat_env Eref=10 uk=F)
***** text : string (500)
CAT_D( Dbc=7 Dc=78 domname=notnamed oftype=S datatype=4)
CAT_A( Bdc=7 Ac=77 Rc=70 Dc=78 atn=text Eref=0 uk=F)
CAT_STRING( Dbc=7 Dc=78 length=500)
```

```
***** end ;
***** Dossier : entity_ag of
CAT_R( Dbc=7 Rc=79 relname=Dossier reltype=EE)
CAT_D( Dbc=7 Dc=81 domname=Dossier oftype=e datatype=5)
CAT_STRUC( Dbc=7 Dc=81 Rc=79)
CAT_R( Dbc=7 Rc=80 relname=Dossier_p reltype=P )
CAT_COMP( Dbc=7 Rcompc=80 REc=79)
CAT_A( Bdc=7 Ac=82 Rc=79 Dc=5 atn=Dossier_c Eref=79 uk=F)
CAT_A( Bdc=7 Ac=83 Rc=80 Dc=5 atn=Dossier_c Eref=79 uk=F)
CAT_R( Dbc=7 Rc=84 relname=Dossier_g reltype=G )
CAT_COMP( Dbc=7 Rcompc=84 REc=79)
CAT_A( Bdc=7 Ac=85 Rc=84 Dc=5 atn=Dossier_c Eref=79 uk=F)
***** Appel (1,3) and
CAT_EAGG( Dbc=7 Raggc=79 Rcompc=69 min=1 max=3)
CAT_A( Bdc=7 Ac=86 Rc=84 Dc=5 atn=Appel_c Eref=69 uk=F)
***** Article ;
CAT_EAGG( Dbc=7 Raggc=79 Rcompc=47 min=0 max=999)
CAT_A( Bdc=7 Ac=87 Rc=84 Dc=5 atn=Article_c Eref=47 uk=F)
***** dossier_n : integer
CAT_A( Bdc=7 Ac=88 Rc=80 Dc=1 atn=dossier_n Eref=0 uk=F)
***** end ;
***** Session : entity
CAT_R( Dbc=7 Rc=89 relname=Session reltype=EK)
CAT_D( Dbc=7 Dc=91 domname=Session oftype=e datatype=5)
CAT_STRUC( Dbc=7 Dc=91 Rc=89)
CAT_R( Dbc=7 Rc=90 relname=Session_p reltype=P )
CAT_COMP( Dbc=7 Rcompc=90 REc=89)
CAT_A( Bdc=7 Ac=92 Rc=89 Dc=5 atn=Session_c Eref=89 uk=F)
CAT_A( Bdc=7 Ac=93 Rc=90 Dc=5 atn=Session_c Eref=89 uk=F)
***** session_no : integer ;
CAT_A( Bdc=7 Ac=94 Rc=90 Dc=1 atn=session_no Eref=0 uk=F)
***** titre : string (40)
CAT_D( Dbc=7 Dc=96 domname=notnamed oftype=S datatype=4)
CAT_A( Bdc=7 Ac=95 Rc=90 Dc=96 atn=titre Eref=0 uk=F)
CAT_STRING( Dbc=7 Dc=96 length=40)
***** end ;
```

```
***** Art_sess : relship
CAT_R( Dbc=7 Rc=97 relname=Art_sess reltype=EA)
CAT_D( Dbc=7 Dc=99 domname=Art_sess oftype=a datatype=5)
CAT_STRUC( Dbc=7 Dc=99 Rc=97)
CAT_R( Dbc=7 Rc=98 relname=Art_sess_p reltype=P )
CAT_COMP( Dbc=7 Rcompc=98 REc=97)
CAT_A( Bdc=7 Ac=100 Rc=97 Dc=5 atn=Art_sess_c Eref=97 uk=F)
CAT_A( Bdc=7 Ac=101 Rc=98 Dc=5 atn=Art_sess_c Eref=97 uk=F)
***** between Article : art (1,10)
CAT_R( Dbc=7 Rc=102 relname=Art_sess_d reltype=A )
CAT_COMP( Dbc=7 Rcompc=102 REc=97)
CAT_A( Bdc=7 Ac=103 Rc=102 Dc=5 atn=Art_sess_c Eref=97 uk=F)
CAT_A( Bdc=7 Ac=104 Rc=102 Dc=5 atn=Article_c Eref=47 uk=F)
CAT_DESIG( Dbc=7 RRc=97 REc=47 rolename=art min=1 max=10)
***** and Session : sess
***** end ;
CAT_A( Bdc=7 Ac=105 Rc=102 Dc=5 atn=Session_c Eref=89 uk=F)
CAT_DESIG( Dbc=7 RRc=97 REc=89 rolename=sess min=1 max=999)
***** Chap : relshp_ag
CAT_R( Dbc=7 Rc=106 relname=Chap reltype=AA)
CAT_D( Dbc=7 Dc=108 domname=Chap oftype=e datatype=5)
CAT_STRUC( Dbc=7 Dc=108 Rc=106)
CAT_R( Dbc=7 Rc=107 relname=Chap_p reltype=P )
CAT_COMP( Dbc=7 Rcompc=107 REc=106)
CAT_A( Bdc=7 Ac=109 Rc=106 Dc=5 atn=Chap_c Eref=106 uk=F)
CAT_A( Bdc=7 Ac=110 Rc=107 Dc=5 atn=Chap_c Eref=106 uk=F)
***** of Art_sess
CAT_AAGG( Dbc=7 Raggc=106 Rcompc=97)
***** Chap_no : (1..10)
CAT_D( Dbc=7 Dc=112 domname=notnamed oftype=i datatype=1)
CAT_INTD( Dbc=7 DC=112 min=1 max=10)
CAT_A( Bdc=7 Ac=111 Rc=107 Dc=112 atn=Chap_no Eref=0 uk=F)
***** end ;
***** type tci : document
CAT_D( Dbc=7 Dc=113 domname=tci oftype=D datatype=5)
```

```
***** structure
***** corps : text
***** end ;
***** Proceed : entity_ag
CAT_R( Dbc=7 Rc=114 relname=Proceed reltype=EE)
CAT_D( Dbc=7 Dc=116 domname=Proceed oftype=e datatype=5)
CAT_STRUC( Dbc=7 Dc=116 Rc=114)
CAT_R( Dbc=7 Rc=115 relname=Proceed_p reltype=P )
CAT_COMP( Dbc=7 Rcompc=115 REc=114)
CAT_A( Bdc=7 Ac=117 Rc=114 Dc=5 atn=Proceed_c Eref=114 uk=F)
CAT_A( Bdc=7 Ac=118 Rc=115 Dc=5 atn=Proceed_c Eref=114 uk=F)
CAT_R( Dbc=7 Rc=119 relname=Proceed_g reltype=G )
CAT_COMP( Dbc=7 Rcompc=119 REc=114)
CAT_A( Bdc=7 Ac=120 Rc=119 Dc=5 atn=Proceed_c Eref=114 uk=F)
***** of Chap ;
CAT_EAGG( Dbc=7 Raggc=114 Rcompc=106 min=0 max=999)
CAT_A( Bdc=7 Ac=121 Rc=119 Dc=5 atn=Chap_c Eref=106 uk=F)
***** chap_init : tci
CAT_A( Bdc=7 Ac=122 Rc=115 Dc=113 atn=chap_init Eref=0 uk=F)
***** end ;
***** reset
***** quit
```

BIBLIOGRAPHIE

Générale

<Abr_74> JR Abrial, "Data Semantics.", dans Data Base Management., Eds. Klimbie & Koffman, North Holland, 1974.

<Abr_78> JR Abrial, "Manuel du Langage Z.", notes Z/1 à Z/13, notes de recherche non publiées.

<Adi_83> M Adiba, "La Gestion du Temps dans les SGBD.", in Base de Données Nouvelles Perspectives, rapport du groupe BD3, pag 145, INRIA, jan 83.

<And_82> TL Anderson, "Modelling Time at the Conceptual Level." , in Proceedings of 2° International Conference on Databases, Academic Press, Israel, 1982.

<And_83> TL Anderson, "Modelling Events ang Processes at the Conceptual Level." , dans Proceedings of International Conference On Databases, Israel, sep 83.

<AS_82> E Allman & M Stonebraker, "Observations on the Evolution of a Software System.", Memorandun n° UCB/ERL M82/59, jun 82.

<ACJ_83> G Ariav, J Clifford & M Jarke, "Panel on Time and Databases." in Proceedings of SIGMOD Conference 83, pag 243.

<ABDR_83> M Adiba et al. , "Bases de Données et Nouvelles Applications.", rapport de recherche n° 349, IMAG, Grenoble, fev 83.

<Bro_78> ML Brodie, "Specification and Verification of Data Base Semantic Integrity.", PhD Thesis, Computer Systems Research Group, Université de Toronto, CSRG-91, avr 78.

<Bro_80> ML Brodie, "The Application of Data Types to Database Semantic Integrity.", Information Systems, v 5, pag 287, 1980.

<BS_80> F Bancilhon & M Scholl, "Design of a Backend Processor of a Database Machine.", dans Proceedings of ACM SIGMOD Conference, Los Angeles, Ca, 1980.

<BS_82> F Bancilhon & M Scholl, "On-line Processing of Compacted Relations.", dans Proceedings of the 8th VLDB, Mexique, 1982.

<BD3_83> INRIA, "Bases de Données: Nouvelles perspectives.", rapport du groupe BD3, publication INRIA, jan 83.

<BFM_79> B Breutman, E Falkenberg & R Maurer, "CSL: A Language for Defining Conceptual Schemas.", dans Database Architecture, Bracchi and Nijssen eds. , North Holland, 1979.

<BRV_83> G Bogo, H Richy & I Vatton, "Un Modèle de Représentation de Documents Généralisés.", Actes des Journées Francophones sur la Manipulation des Documents, Rennes, mai 83.

<BADW_82> A Bolour et al. , "The Role of Time dans Information Processing: a survey.", ACM SIGMOD Record, v 11, nº 3, pag 27, avr 82.

<Che_76> PPS Chen, "The Entity Relationship Model: Toward a Unified View of Data.", ACM TODS, V 1, nº 1, pag 9, 1976.

<Cod_70> EF Codd, "A Relational Model of Data for Large Shared Data Banks.", Communications of ACM, V 13, nº 6, pag 377, 1970.

<Cod_79> EF Codd, "Extending the Database Relational Model to Capture More Meaning.", ACM TODS, v 4, nº 4, dec 79.

<CB_74> DD Chamberlin & RE Boyce, "SEQUEL: A Structured English Query Language.", dans Proceedings of ACM SIGFIDET Workshop, pag 249, Ann Arbor, Mich, mai 74.

<CW_82> J Clifford & DS Warren, "Formal Semantics for Databases.", ACM TODS, v 8, nº 2, pag 214, jun 83.

<CGI_81> DD Chamberlin, AM Gilbert & RA Yost, "A History of System R and SQL/Data System.", dans Proceedings of VLDB 81, pag 456.

<Dat_82> CJ Date, "An Introduction to Data Base Systems.", Addison-Wesley Systems Programming Series, 3rd edition, 1982.

<Dat_83> CJ Date, "An Introduction to Data Base Systems, II Volume.", Addison-Wesley Systems Programming Series, 1983.

<EN_79> C Ellis & G Nutt, "Computer Science and Office Information Systems.", Report SSL_79_6, XEROX, Palo Alto, Californie, jun 79.

<Fal_74> EF Falkenberg, "Time-handling dans Data Base Systems.", rapport 07/1974, Université de Stuttgart, juillet 74.

<Fern_81> F Fernandez, "La Micro Memoire Relationnelle (MIMER): Un Outil pour la Construction de SGBD Relationnels. Projet Microbe.", Thèse Docteur Ingenieur, IMAG, nov 81.

<Fert_81> L Ferrat, "Projet Microbe: Guide d'Utilisateur du Langage MIQUEL.", rapport interne, IMAG, jan 81.

<Gra_81> J Gray, "The Transaction Concept: Virtues and Limitations.", in Proceedings of VLDB 81, pag 144.

<Hame_81> J Hameon, "Indexation et Classement en Bureautique.", Thèse de Doctorat de 3ème Cycle, INPG, Grenoble, 1981.

<Hamm_79> H Hammer, "Laboratory for Computer Sciences: Progress Report/Office Automation Groupe.", MIT, Cambridge, 79.

<HL_81> L Haskin & RA Lorie, "On Extending the Functions of a Relational Database System.", rapport de recherche RJ 3182 (38988). IBM Research Laboratory, San Jose, Ca, nov 81.

<HM_81> M Hammer & D McLeod, "Database Description with SMD: a Semantic Database Model.", ACM TODS, V 6, nº 3, pag 351, sep 81.

<IFIP_82> "IFIP WG 8.1 Working Conference on Comparative Review of Information Systems Design Methodologies." , eds TW Olle, HG Sol & AA Verrijn_Stuart, Noordwijkerhout, The Netherlands, mai 82.

<Jol_82> V Joloboff, "UBIK: Un Formalisme de Modélisation.", dans Seminaire Base de Données, pag 175, Toulouse, nov 82.

<JW_78> K Jensen & N Wirth, "PASCAL User Manual and Report.", Second Corrected Reprint of the Second Edition, Springer-Verlang,1978.

<JKL_81> V Joloboff, I Kowarski & M Lopez, "Projet Base de Données Textuelles.", rapport de recherche IMAG nº 273, nov 81.

<JLV_83> V Joloboff, M Lopez & F Velez, "A Generalized Data Base Model.", rapport interne, Centre de recherche BULL, Grenoble, jan83.

<Kle_77> W Kent, "Entities and Relationships dans Information.", in International IFIP-TC-2 Working Conference, Nice, pag 1, jan 77.

<Klo_81> MR Klopprogge,"TERM : An Approach to include the Time Dimension in the Entity Relationship Model.", dans Proceedings of 2nd International Conference on Entiy Relationship Approach, pag 477, 1981.

<KL_82> I Kowalski & M Lopez, "The Document Concept dans a Database.", in Proceedings of the ACM SIGMOD Conference, Orlando, Florida, 1982.

<KL_83> MR Klopprogge & PC Lockemann, "Modelling Information Preserving Databases: Consequences of the Concept of TIME. " , dans Proceedings VLDB 83, Florence, 83.

<LP_82> R Lorie & W Plouffe, "Complex Objects and Their Use dans Design Transactions." , rapport de recherche RJ 3706 (42922), IBM Research Laboratory, San Jose, Ca, dec 82.

<LS_83> V Lum & HJ Schek (chairpersons), "Panel on Complex Data Objects : Text, Voice, Images. Can DBMS Manage them?", VLDB 83, Florence.

<Nij_76> GM Nijssen, "A Gross Architecture for the Next Generation Database Management Systems.", dans Modelling dans Database Management Systems, IFIP TC-2, Freundenstadt, Allemagne, jan 76.

<NFFL_82> GT Nguyen et al. "MICROBE: Manuel de Référence.", IMAG, jan 82.

<OS_82> R Overmeyer & M Stonebraker, "Implementation of a Time Expert dans a Data Base System. " , ACM SIGMOD Record, V 11, n° 3, pag 51, avr 82.

<Ria_83> D Rialhe, "Système BD-CAO pour la Modélisation et Integration des Données dans un Environnement CAO.", rapport de recherche IMAG n° 365, mars 83.

<RS_82> LA Rowe & M Stonebraker, "Architecture of Future Data Base Systems.", ACM SIGMOD Record, v 11, n° 1, jan 81.

<RSh_82> LA Rowe & K Shoens, "A Form Application Development System.", dans Proceedings of the ACM SIGMOD Conference, Orlando, USA, mai 82.

<Sal_68> G Salton, "Automatic Information Organisation and Retrieval.", Mc Graw Hill Book Co, 1968.

<Sal_71> G Salton, "The Smart Project.", Prentice Hall, 1971.

<Sak_83> H Sakai, "Entity-Relationship Approach to Logical Database Design.", dans Proceedings of 3rd International Conference on Entity-Relatinship Approach, Anaheim, Cal, Us,, pag 133, North Holland, oct 83.

<Sch_82> U Schiel, "The Temporal Hierarchic Data Model - THM.", Université de Stuttgart, bericht 10/82.

<Sch_83> U Schiel, "An Abstract Introduction to the Temporal Hierarchic Data Model (THM).", dans Proceedings of VLDB 83, Florence, pag 322.

<Schm_77> HA Schmid, "An Analysis of some Constructs for Conceptual Models.", in International IFIP-TC-2 Working Conference, Nice, pag 67, 1977.

<Sun_75> B Sundgren, "Theory of Databases.", New York, 1975.

<SK_80> M Stonebraker & K Keller, "Embedding Expert Knowledge and Hypotetical Data Bases into a Data Base System.", dans Proceedings of ACM SIGMOD Conferece on Management of Data, Santa Monica, Ca, pag 58, mai 80.

<SS_77> JM Smith & DCP Smith, "Database Abstractions: Aggregation and Generalization.", ACM TODS, v 2, nº 2, pag 105, jun 77.

<SNF_80> CS Santos, EJ Newhold & AL Furtado, "A Data-type Approach to the Entity-Relationship Model.", dans Entity-Relationship Approach to Systems Analysis and Design, Ed PPS Chen, pag 103, North Holland, 1980.

<SSW-80> P Scheuermann, G Schiffner & H Weber, "Abstractions Capabilities and Invariants Properties Modelling whitin the ER Approach.", in Entity-Relationship Approach to System Analysis and Design, Ed PPS Chen, North Holland, 1980.

<SSKG_76> M stonebraker et al. , "Document Processing in a Relational Data Base System.", Memorandum UCL/ERL M82/32, University of California, Berkeley, mai 82.

<SWKH_76> M Stonebraker et al. , "The Design and Implementation of INGRES" , ACM TODS, V 1, nº 3, pag 189, sep 76.

<Tan_81> G Tang, "A Management System for an Integrated Data Base of Pictures and Alphanumeric Data.", Computer Graphics and Image Processing, nº 16, pag 270, 1981.

<Tsi_80> D Tsichritzis, "OFS: an Integrated Form Management System.", dans "A Panache of DBMS Ideas III", Computer System Research Group, University of Toronto, 1980.

<Tsi_81> D Tsichritzis, "Integrating Data Base and Office Systems.", dans Proceedings of the 7th VLDB Conference, Cannes, 1981.

<TL_77> DC Tsichritzis & FH Lochovsky, "Data Base Management Systems.", Academic Press Inc, 1977.

<TK_78> DC Tsichritzis & A Klug (eds), "The ANSI/X3/SPARC DBMS Framework: Report of the Study Group on Data Base Management Systems.", Information Systems, v 3, nº 3, 1978.

<Vel_82> F Velez, "LAMBDA: un Langage de Définition et de Manipulation de Base de Données Généralisées.", dans Seminaire Base de Données, pag 203, Toulouse, nov 82.

<VN_82> D Vermier & GM Nijssen, "A Procedure to Define the Object Type Structure of a Conceptual Schema.", Information Systems, V 7, nº 4, pag 329, 1982.

<WFW_75> G Wiederhold et al. , "Structured Organization of Clinical Database. " , dans Proceedings of the AFIPS National Conference, Anhein, 1975.

<Zlo_79> M Zloof, "A Language for Office and Business Automation.", dans Proceedings of the Conference on Data Base Engineering, IBM Japan, nov 79.

<ZM_82> C Zaniolo & MA Melkanoff, "A Formal Approach to the Definition and the Design of Conceptual Schemata for Database Systems.", ACM TODS, V 7, n1, pag 24, mar 82.

Rapports de Recherche TIGRE

<TIGRE_1> Présentation Générale du Projet TIGRE.
Janvier 83.

<TIGRE_2> The TIGRE Data Model.
M Lopez, J Palazzo Oliveira & F Velez.
Novembre 83.

<TIGRE_3> Proposition de Modèle pour la Normalisation des Documents.
G Bogo, H Richy, I Vatton.
Mars 83.

<TIGRE_4> Recommandations pour l'Ergonomie d'un Poste de Travail Attaché a un Système Bureautique.
I Forestier.
Mars 83.

<TIGRE_5> La Correspondance de Schemas entre les Modèles TIGRE et Relationnel.
J Palazzo Oliveira, F Velez.
Novembre 83.

<TIGRE_6> Déscription des éléments du Modèle Document.
G Bogo, H Richy, I Vatton.
Septembre 83.

<TIGRE_7> Programmation en Logique pour une Base de Données Géneralisées.
N Gia Toan, J Olivares, P Winninger.
Novembre 83.

<TIGRE_8> Machines Bases de Données: Etat de l'Art et Perspectives.
M Burnier.
Novembre 83.

<TIGRE_9> Caractérisation des Formulaires pour une Base de Données Généralisée.
C Collet.
Novembre 83.

<TIGRE_10> Accés Concurrents aux Documents.
S Baltas.
Janvier 84.

<TIGRE_11> Définition du Langage Lambda.
F Velez.
Fevrier 84.

<TIGRE_12> Logic Programming for a Generalized Data Base Management System.
M Adiba & GT Nguyen.
Janvier 84.

<TIGRE_13> Modèle et Fonctionnalités pour un Système Integrant Outil BD et Outils de Conception.
D Rialhe
Janvier 84.

<TIGRE_14> SAGE: Un système d'Autorisation Généralisé.
M Adiba & F Azrou
Janvier 84

<TIGRE_15> Cooperation de PROLOG et d'un SGBD Généralisé:
Principes et Applications.
Nuguyen GT, J Olivares & P Winniger.
Avril 84.

AUTORISATION de SOUTENANCE

VU les dispositions de l'article 3 de l'arrêté du 16 avril 1974,

VU les rapports de présentation de Messieurs
. M. ADIBA, Professeur
. M. LEONARD, Professeur

Monsieur José PALAZZO MOREIRA DE OLIVEIRA

est autorisé à présenter une thèse en soutenance pour l'obtention du diplôme de DOCTEUR-INGENIEUR, spécialité "Informatique".

Fait à Grenoble, le 05 juin 1984

Le Président de l'I.N.P.-G

D. BLOCH
Président
de l'Institut National Polytechnique
de Grenoble

P.O. le Vice-Président,