86/37

Sc N 86/172 A

Université de Nancy I U.E.R. Sciences Mathématiques

Centre de Recherche en Informatique de Nancy

ETUDES SUR L'APPROCHE BASE DE DONNEES POUR LA GESTION D'OBJETS ISSUS DU DEVELOPPEMENT DE LOGICIEL

THESE

BIBLIOTHEQUE * UNIVERSITE NANCY THEQUE-SCIENCES *

soutenue publiquement le 6 Mai 1986

A L'UNIVERSITE DE NANCY I

pour l'obtention du grade de
DOCTEUR DE L'UNIVERSITE DE NANCY I en INFORMATIQUE

par
Myung-Joon KIM

devant la commission d'examen

Président :.....Jean Jacques CHABRIER
Rapporteurs :..... André FLORY
Serge MIRANDA
Examinateurs :.....Jean Claude DERNIAME
Jean Pierre FINANCE
Odile FOUCAUT

BIBLIOTHEQUE SCIENCES NANCY 1
D 095 147521 0

Université de Nancy I U.E.R. Sciences Mathématiques

Centre de Recherche en Informatique de Nancy

ETUDES SUR L'APPROCHE BASE DE DONNEES POUR LA GESTION D'OBJETS ISSUS DU DEVELOPPEMENT DE LOGICIEL

THESE

soutenue publiquement le 6 Mai 1986

A L'UNIVERSITE DE NANCY I

pour l'obtention du grade de
DOCTEUR DE L'UNIVERSITE DE NANCY I en INFORMATIQUE

par
Myung-Joon KIM

devant la commission d'examen

Président :.....Jean Jacques CHABRIER
Rapporteurs :..... André FLORY
Serge MIRANDA
Examinateurs :.....Jean Claude DERNIAME
Jean Pierre FINANCE
Odile FOUCAUT

A MES PARENTS (NAI POUMONIM KE)

REMERCIEMENTS

Le moment est venu de tourner une page de ma vie et de m'ouvrir au nouvel horizon qu'elle me destine. Cela a été possible grâce non seulement à la force que j'y ai mise, mais aussi grâce aux échanges qui se sont établis avec tout ce qui m'entoure: le milieu naturel dans lequel j'aimerai toujours être, les relations humaines qui sont souvent si subtiles ! et le temps qui passe ... vite.

L'expression de mes sentiments, à l'occasion de cette préface, touche vraiment à trop de choses et je pense qu'il est simplement plus convenable, dans ce cas, d'avoir l'esprit de prière.

Pourtant, je ne peux pas m'empêcher de penser à ma famille, surtout à mes parents qui m'ont aidé de loin, et aussi de dire un grand merci aux professeurs qui m'ont guidé, aux personnes qui ont accepté de participer au jury et aussi aux collègues avec qui j'ai eu d'abondantes conversations utiles.

Plus précisément, je tiens à remercier ici:

Monsieur Jean-Claude DERNIAME, Professeur à l'Université de Nancy I, mon Directeur de Recherche, à qui je m'adresse toujours aisément. Ainsi, j'ai eu d'etroits rapports avec lui pendant ces quatre dernières années.

Monsieur Jean Jacques CHABRIER, Professeur à l'Université de Dijon, qui m'a tendu la main depuis mon arrivée au laboratiore et enfin qui a accepté d'être le président de jury.

Monsieur Jean Pierre FINANCE, Professeur à l'université de Nancy I, qui m'a inculqué la rigueur scientifique.

Monsieur André FLORY, Professeur à l'Université de Lyon III, qui m'a fait l'honneur d'être rapporteur de ma thèse.

Madame Odile FOUCAUT, Maître de conférence à l'Université de Nancy I, qui m'a aidé dans mes travaux avec son exactitude et sa délicatesse

habituelles. C'était toujours un plaisir de discuter avec elle.

Monsieur Serge MIRANDA, Professeur à l'Université de Nice, qui, non seulement, a accepté de faire un rapport sur le présent travail mais aussi m'a donné l'exemple d'une personnalité ouverte à tout le monde.

Je suis également reconnaisant envers mes collègues du Centre de Recherche en Informatique de Nancy, grâce auxquels j'ai eu une merveilleuse ambiance de recherche, plus particulièment à Messieurs N. TOUNSI et M. S. OUERGHI. Je dois remercier à Mademoiselle F. PIERRE et à Monsieur M. ESSAIDI qui m'ont réservé de leur temps précieux pour soigner la présentation écrite du travail.

J'adresse mes remerciements aux gouvernements français et coréen qui m'ont offert l'occasion d'étudier en France et notamment à Monsieur Yves TANGUY, le directeur de CESELEC, qui m'a assisté tout au long de mon séjour en France.

Je termine l'expression de mes sentiments en pensant à ma chère Sun-Yung, dont la simple présence près de moi m'a été bénéfique.

*
***
*****

TABLE DES MATIERES

	PAGE
INTRODUCTION	
1. Avant-propos	10
2. Sujets de la thèse	12
3. Plan de la thèse	14
CHAPITRE I - SPECIFICATION DE R-LTR: BASE DE DONNEES RELATIONNELLE POUR LE LANGAGE LTR-3	
1. Introduction	16
2. Modèle relationnel de données (Aspect syntaxique)	21
2.1 Définition des structures d'objets	21
2.1.1. Objet, rubrique et relation	21
2.1.2. Relation, tuple, attribut, domaine et clé	22
2.1.3. Contraintes d'intégrité inhérentes	23
2.2 Schéma relationnel	25
2.3 Expression relationnelle	25
2.3.1. Opérandes	26
2.3.2. Opérations choisies	27
3. Contraintes d'intégrité (Aspect sémantique)	29
3.1 Contraintes d'intégrité structurelles et normalisation des relations (Sémantique formelle)	29
3.1.1. Dépendances fonctionnelles	29
3.1.2. Troisième forme normale	30
3.1.3. Dépendances d'inclusion inter-relations	30
3.2 Contraintes d'intégrité internes (Sémantique informelle)	31
4. Conclusion	32
CHAPITRE II - LANGAGE DE BASE DE DONNEES RELATIONNELLE: R-LTR	
1. Introduction	36
2. Langage de définition de données avec contraintes d'intégrité	39

2.1. Type de TUPLE 39
1.2.1. Syntaxe 39
1.2.2. Sémantique 39
1.2.3. Exemple 40
1.2.4. Fonctions prédéfinies pour les contraintes d'intégrité 40
2.2 Déclaration de relations 41
2.2.1. Syntaxe 41
2.2.2. Sémantique 42
2.2.3. Exemple 42
2.2.4. Fonctions prédéfinies pour les contraintes d'intégrité 43
2.3. Déclaration de base de données 44
2.3.1. Syntaxe 44
2.3.2. Sémantique 44
2.3.3. Exemple 45
3. Langage de manipulation de données 47
3.1. Opérations de mise à jour 47
3.2. Interrogations de données par une expression relationnelle 48
3.2.1. Syntaxe 48
3.2.2. Sémantique 49
3.3.3. Exemple 50
3.3.4. Fonctions prédéfinies de calcul 51
4. Structure d'une application 52
5. Réalisation du préprocesseur R-LTR 56
6. Conclusion 58

CHAPITRE III - LES ETUDES SUR LE MODELE ENTITE-ASSOCIATION

1. Introduction 62
2. Concepts et notions 65
2.1. Modèle Entité-Association 65
2.2. Diagramme Entité-Association 68
3. Elimination de certaines ambiguités sur les propriétés de cardinalité 70
3.1. Cardinalité d'un type d'entité dans un type d'association 70
3.1.1. Fonctionalité 70

3.1.2. Cardinalité 73
3.2. Dépendance entre entités 75
3.2.1. Dépendance d'existence et dépendance d'identification 75
3.3.2. Dépendance fonctionnelle entre entités 78
3.3. Interprétation d'un type d'association n-aire par dépendance fonctionnelle entre entités 80
3.4. Comparaison 82
4. Un guide d'application du modèle Entité-Association 86
4.1. Le premier pas vers la normalisation 87
4.1.1. Type d'entité bien défini 88
4.1.2. Type d'association bien défini 90
4.1.3. Diagramme Entité-Association bien défini 92
4.2. Règles de mise à jour des instances 95
5. Quelques types génériques d'association basés sur les notions d'abstraction de données 96
5.1. Classification - Instanciation 97
5.2. Généralisation - Spécialisation 99
5.3. Agrégation - Décomposition 101
5.4. Regroupement - Partition 103
5.5. Remarques concluantes 104
6. Conclusion 107

CHAPITRE IV - CONSTRUCTION DE SCHEMA RELATIONNEL EN TROISIEME FORME NORMALE PAR TRANSFORMATION DE SCHEMA ENTITE-ASSOCIATION

1. Introduction 109
2. Transformation d'un schéma Entité-Association en schéma relationnel 112
2.1. Etape-1 : Construction des schémas de relation pour chaque type d'entité et chaque type d'association sans se préoccuper des informations de l'obligation de type d'entité (Transformation) 112
2.2. Etape-2 : Modification du schéma relationnel obtenu à l'Etape-1 à partir de l'information indiquant des types d'entité obligatoires (Optimisation) 117

3. Analyse des clés dans l'algorithme de transformation 120
4. Réalisation 124
4.1. Système interactif de construction des schémas Entité-Association 124
4.2. Système de transformation d'un schéma Entité-Association en schéma relationnel 128
5. Conclusion 131

CHAPITRE V - PROPOSITION D'UN SYSTEME DE GESTION D'OBJETS

1. Introduction 133
2. Une structure d'accueil: Système de gestion d'objets 137
2.1. Choix du modèle conceptuel Entité-Association 138
2.1.1. Critères de choix 138
2.1.2. Concepts du modèle dans le cadre du système de gestion d'objets 139
2.2. Fonctions du système de gestion d'objets 142
2.2.1. Fonction de description 142
2.2.2. Fonction de manipulation 142
2.2.3. Fonction de maintien de l'intégrité 143
2.3.4. Fonction de stockage 143
2.4.5. Fonction de définition de vue 144
2.4.6. Fonction d'interfaçage 144
2.3. Caractéristiques du système de gestion d'objets 144
3. Présentation des travaux en cours 147
3.1. Descritption des connaissances d'un atelier logiciel 147
3.2. Etude sur le maintien de la cohérence 150
3.3. Interfaçage utilisateur 150
4. Conclusion 152

CONCLUSION GENERALE 153

REFERENCES BIBLIOGRAPHIQUES 157

INDEX 175

INTRODUCTION

Résumé

Dans cette introduction générale, nous présentons d'abord une partie des travaux menés au sein de l'équipe Génie Logiciel du Centre de Recherche en Informatique de Nancy (CRIN), ensuite nous citons les sujets traités par cette thèse, et enfin nous donnons le plan d'exposé.

Sommaire

1. Avant-propos
2. Sujets de la thèse
3. Plan de la thèse

INTRODUCTION

1. Avant-propos

Au sein de l'équipe Génie Logiciel du Centre de Recherche en Informatique de Nancy, une maquette d'un système de manipulation de multi-bases de données (système d'accès simultanés à plusieurs bases) [GODART81] [LITWIN81], appelée TYP-R, a été devéloppée [TOUNSI82,83]. TYP-R est écrit en TYP qui est un système intégré de programmation modulaire à l'aide de la notion de type abstrait de données [CHABRIER82-a,b]. TYP-R nous a permis de dégager certains difficultés qui se posent lors de la conception et de l'implantation d'un grand logiciel tel que, par exemple, un Système de Gestion de Bases de Données (SGBD) relationnel, en s'appuyant sur les méthodologies, type abstrait de données et modularité.

Une nouvelle version de TYP-R a été réalisée en CLU [LISKOV81] (un langage diffusé dans les laboratoires et s'appuyant sur les mêmes méthodologies), permettant de tenir compte des contraintes d'intégrité et de sécurité au niveau des multi-bases de données relationnelles. Dans le projet ADONIS [GODART84] [NASSIF83], un système d'intégrité est étudié et réalisé pour permettre de poser des contraintes d'intégrité sur des multi-bases et de les vérifier au cours de la mise à jour de données stockées. La confidentialité des bases est assurée par l'utilisation des relations virtuelles et des prédicats de sécurité. Tout utilisateur peut avoir le droit de donner aux autres utilisateurs des autorisations d'accès aux données, et de les révoquer. Trois types abstraits de données ont éte introduits et réalisés:

- RELATION (intermédiaire) qui permet d'effectuer les opétations de l'algèbre relationnelle sur les objets d'un tel type;
- MULTIBASE pour permettre de définir et de manipuler des schémas de la base;
- CI (Contrainte d'Intégrité) pour gérer des contraintes d'integrité.

TYP-R et ADONIS sont des prototypes réalisés avec des langages peu ou pas utilisés en milieu industriel. Les langages de Base de Données (BD) proposés dans les travaux précédents sont de mode interactif, c'est à dire que les commandes sont interprétées. Cela nous a amené à vouloir montrer que nos idées et nos exprériences peuvent être exploitées pour réaliser, de façon économique, un système diffusable en s'appuyant sur un langage hôte. Nous avons choisi le langage LTR-3 (Langage de Temps Réel, version 3), langage modulaire destiné à la programmation de systèmes temps réel.

Pour fournir aux utilisateurs de LTR-3 un moyen qui leur permet de définir et de manipuler des BDs relationnelles, nous proposons d'abord une extension compatible du langage, appelée R-LTR (LTR Relationnel). Ensuite, nous proposons une solution de réalisation par un préprocesseur qui traduit des programmes R-LTR en LTR-3. Après avoir compilé ce dernier, le code traduit sera exécuté en interaction avec un système support contenant des fonctions de SGBD.

Pour les applications de BD, il n'est pas suffisant de fournir un langage de BD traditionnel, parce qu'il est important de prendre en considération l'étape de conception de BD. Nous choisissons le modèle Entité-Association (E-A) comme modèle conceptuel pour faire l'analyse préalable par abstraction d'un problème. Nous étudions de manière approfondie le modèle E-A et ensuite nous proposons une méthode de construction systématique d'une BD relationnelle par transformation d'un schéma E-A, représenté par un diagramme, en un schéma relationnel.

Enfin, nos acquis dans l'approche BD pourront aller plus loin dans le cadre du génie logiciel, en abordant des problèmes plus généraux tels que ceux issus du développement de logiciels et de la gestion de projets. Un logiciel peut être défini comme une collection d'objets de types différents issus de toutes les étapes du cycle de vie. Il est généralement difficile d'avoir une vue globale d'un logiciel, notamment pour suivre son évolution. Nous proposons de structurer et stocker ces objets en une base de données, appelée Base d'Objets. Les objets de cette base seront gérés (décrits, instanciés,

modifiés, interrogés, etc.) par un Système de Gestion d'Objets. qui sera le noyau central d'un atelier logiciel. Nous utilisons le modèle Entité-Association pour le schéma conceptuel dans le système de gestion d'objets.

2. Sujets de la thèse

Le but final de la thèse concerne la mise en valeur de l'approche base de données et notamment la technique de conception et d'implantation de schéma conceptuel de données, pour proposer une solution aux problèmes du développement organisé de logiciels. Il s'agit de proposer un système de gestion d'objets qui permet de représenter et manipuler tous les objets (concrets ou abstraits) apparus au cours du développement de logiciels. Nous visons un système qui soit un noyau d'une structure d'accueil et qui soit un support à la construction d'un atelier logiciel, indépendamment de toute méthode de conception et de tout langage de programmation.

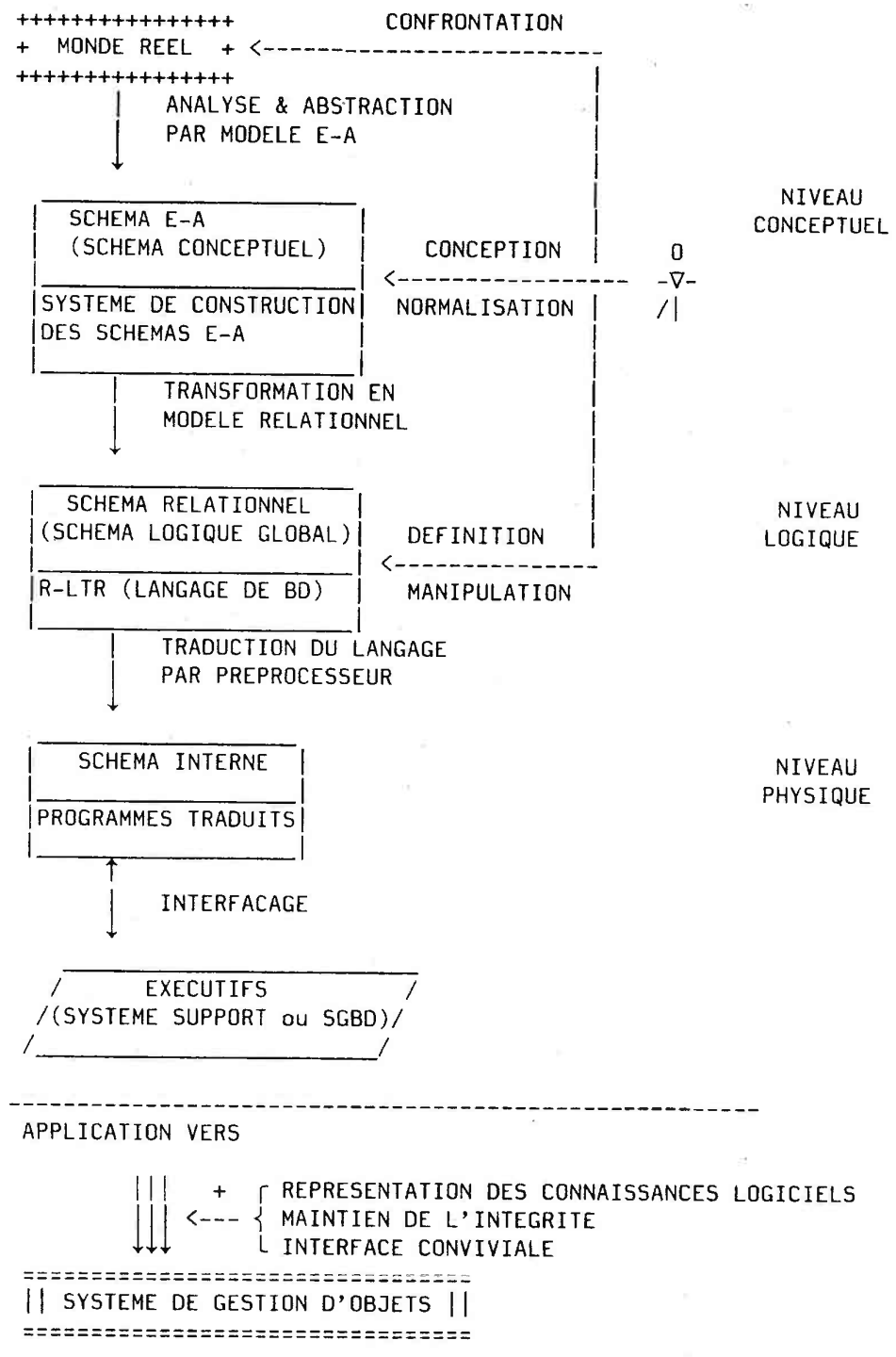
Le travail abordé dans la présente thèse se divise en trois sujets selon les trois aspects de l'apporche BD: le langage de BD, la conception de BD et l'application de BD en génie logiciel.

- L'extension d'un langage de programmation diffusé industriellement (en l'occurrence, le langage LTR-3) pour les applications de bases de données relationnelles;

- La mise en évidence de certaines propriétés du modèle Entité-Association (modèle qui permet une analyse conceptuelle et une abstraction) et le développement d'une méthode de construction systématique d'un schéma relationnel par la transformation d'un schéma E-A;

- La proposition d'un système de gestion d'objets pour représenter et manipuler des objets issus du développement de logiciels et de la gestion de projets.

Les sujets sont schématisés dans la Fig.0-1.



(Fig.0-1) Intégration des applications de l'approche BD
vers un système de gestion d'objets

3. Plan de la thèse

La suite de cet exposé est organisée en cinq chapitres.

Le premier chapitre donne une vue spécifique du modèle relationnel de données en vue de spécifier le langage de BD proposé R-LTR. Dans le deuxième chapitre, nous présentons le langage R-LTR destiné à l'intégration des applications de BD relationnelle dans le langage LTR-3, ainsi que sa réalisation.

Le troisième chapitre est consacré aux études sur le modèle Entité-Association afin de mettre au point certains concepts du modèle, en particulier:

- élimination des ambiguités pour les propriétés de cardinalité;
- proposition d'un guide d'application du modèle E-A;
- extension du modèle E-A en proposant quelques types génériques d'association.

Dans le quatrième chapitre, nous nous intéressons à une méthode et un outil informatique de construction systématique d'un schéma relationnel, par transformation d'un schéma E-A en schéma relationnel.

Au cinquième chapitre, nous décrivons une proposition de la représentation des connaissances et des objets issus du développement de logiciels par un système de gestion d'objets qui utilise le modèle Entité-Association.

Enfin, nous terminons notre exposé par une conclusion générale.

*
***
*****

CHAPITRE I - SPECIFICATION DE R-LTR: BASE DE DONNEES RELATIONNELLE POUR LE LANGAGE LTR-3

Résumé

Le but de ce chapitre est de présenter le modèle relationnel de données de façon à spécifier le langage R-LTR (LTR Relationnel). Les bases théoriques du modèle relationnel seront utilisées dans les chapitres suivants.

Sommaire

1. Introduction
2. Modèle relationnel de données (Aspect syntaxique)
   - 2.1 Définition des structures d'objets
     - 2.1.1. Objet, rubrique et relation
     - 2.1.2. Relation, tuple, attribut, domaine et clé
     - 2.1.3. Contraintes d'intégrité inhérentes
   - 2.2 Schéma relationnel
   - 2.3 Expression relationnelle
     - 2.3.1. Opérandes
     - 2.3.2. Opérations choisies
3. Contraintes d'intégrité (Aspect sémantique)
   - 3.1 Contraintes d'intégrité structurelles et normalisation des relations (Sémantique formelle)
     - 3.1.1. Dépendances fonctionnelles
     - 3.1.2. Troisième forme normale
     - 3.1.3. Dépendances d'inclusion inter-relations
   - 3.2 Contraintes d'intégrité internes (Sémantique informelle)
4. Conclusion

CHAPITRE I

SPECIFICATION DE R-LTR:
BASE DE DONNEES RELATIONNELLE POUR LE LANGAGE LTR-3

1. Introduction

Dans ce premier chapitre, nous présentons le modèle relationnel de données qui est à la fois une spécification du langage R-LTR (LTR Relationnel) et aussi une base théorique pour les chapitres qui suivent.

Base de données

Dans les années soixante, les systèmes de fichiers permettant le stockage de données sur les mémoires secondaires ont évolué pour répondre aux besoins des traitements de données en grande masse. Dans ces systèmes, des fichiers sont utilisés indépendamment les uns des autres (de façon "non intégrée").

L'intérêt de l'approche Base de Données (notée BD) est, d'une part, l'intégration structurée des données et d'autre part, la possibilité de partager leur utilisation par de nombreuses personnes. Citons à cet égard deux définitions qui ont mis l'accent sur l'aspect de l'outil informatique.

> Une base de données est une collection de données opérationnelles†, stockées et utilisées par les systèmes d'application, pour une certaine organisation. Ces données stockées doivent être intégrées et partagées [DATE81].

> Une base de données est un ensemble structuré de données enregistrées sur des supports accessibles par l'ordinateur pour satisfaire simultanément plusieurs utilisateurs de façon sélective et en un temps opportun [DELOBEL82].

(†) données nécessaires aux opérations d'une application.

Par ailleurs, nous pouvons considérer une BD comme une représentation particulière du monde réel des organisations.

> Une base de données est une collection structurée d'objets et de relations entre ces objets qui décrit le monde réel, l'univers d'une organisation [GODART81].

L'approche BD fournit aux utilisateurs des moyens pour organiser et gérer des informations et en plus des aides à la conception par la représentation du monde réel. La performance d'un outil informatique est souvent prise en compte dans les discours quotidiens, mais il y a à peine une quinzaine d'années que l'on a commencé à réfléchir sur l'outil informatique du point de vue méthodologique [BAUER80]. Pour insister sur l'aspect conceptuel, citons une phrase du Professeur DIJKSTRA:

> Un des aspects les plus importants d'un outil informatique est son influence sur l'habitude de penser de ceux qui essaient de l'utiliser [DIJKSTRA72].

Système de gestion des bases de données

Pour exploiter des BDs, il est nécessaire de développer des outils informatiques qui permettent de:

- définir et structurer des données,
- partager des données,
- exprimer et assurer des contraintes d'intégrité,
- consulter et manipuler des données,
- assurer la protection des données,
- etc.

Un logiciel dans lequel ces outils sont regroupés s'appelle un **Système de Gestion de Bases de Données** (noté SGBD). Une réflexion importante à propos des SGBDs est que, quelque soit la technologie de stockage des données, un SGBD doit permettre de construire une application qui ne connait qu'une vue logique des données exprimée dans un

modèle de données. Il s'agit de l'indépendance entre les structures logiques de données et leurs structures physiques d'implantation.

Modèle de données, Schéma de données

L'organisation de données peut être appréhendée à l'aide d'un modèle de données. Un **modèle de données** est défini par un ensemble de structures de données (ou un ensemble de règles de génération) et un ensemble d'opérations définies sur ces structures [MIRANDA84] [TSICHRITZIS82]. Un modèle de données est, d'une part, un outil intellectuel utilisé pour organiser des données, d'autre part, une méthode conceptuelle utilisée pour analyser et interpréter le monde réel.

Le résultat de l'application particulière d'un modèle de données s'appelle un **schéma de données** (cf. l'intention de BD [DATE81]). Un schéma de données décrit simplement l'organisation abstraite de données dans une BD. Le contenu d'une BD à un moment donné est appelé un **état de base de données** (cf. l'extension de BD [DATE81], instance de BD [ULLMAN82]).

Langage de base de données

Un langage de BD est un langage spécifique supportant la gestion de BD. En principe, chaque langage de BD est une combinaison de deux parties: un langage de définition des structures de données et un langage de manipulation des données.

Un **langage de définition des données** permet de définir (ou décrire) la structure de données d'une base, perçue par l'utilisateur selon un modèle de données. Un **langage de manipulation des données** est un ensemble d'opérations sur des objets décrits par le langage de définition des données. Ces opérations se divisent en deux catégories: mise à jour et interrogation. Les opérations de mise à jour concernent les changements d'état d'une BD (ex. insertion, suppression et modification). Les opérations d'interrogation donnent à l'utilisateur la possibilité de consulter les données de la base.

[Note] Dans les discours courants, "Langage de requête" est souvent employé comme synonyme d'un langage de manipulation des données restreint à l'interrogation.

Contraintes d'intégrité

Des restrictions logiques sur des données sont appelées contraintes. Généralement, les contraintes sont exigées pour des raisons d'intégrité de la BD vis-à-vis des lois sémantiques du monde réel, tant au niveau du schéma de données qu'au niveau des états de la BD. Sinha a proposé de distinguer entre contrainte d'intégrité et celle de consistance [SINHA83]. D'après lui, une contrainte d'intégrité spécifie les caractéristiques d'une donnée de la base indépendamment des autres; par contre, une contrainte de consistance spécifie les associations entre deux ou plusieurs données de la base. Nous utilisons simplement le terme **Contrainte d'Intégrité** (notée CI) au sens de correction, exactitude ou validité des données de la base, ce qui exprime l'intégrité et la consistance. Finalement, on peut voir une CI comme l'expression d'un prédicat sur les données de la base qui, lorsque ce prédicat n'est pas respecté, conduit au rejet d'une mise à jour de données [NICOLAS83].

Nous proposons de classer des CIs en trois catégories. D'abord les CIs strictement liées au modèle seront appelées CIs **inhérentes** [TSICHRITZIS82] [BRODIE84-a]. Ces CIs font partie intégrante du modèle et peuvent être considérées comme les sous entendus du modèle. Pour les CIs restantes, nous les appelons CIs **explicites** puisque elles doivent être explicitement exprimées d'une façon ou d'une autre dans le schéma, lors de la définition d'une BD (si le langage de définition des données le permet). En d'autres termes, les CIs inhérentes sont les contraintes induites par le modèle qui se retrouvent donc dans toute BD décrite selon ce modèle. Alors que les CIs explicites sont toutes celles qui peuvent varier selon l'application.

Les CIs explicites qui font intervenir la définition des structures de données représentées par un schéma seront appelées CIs **structurelles**. D'ores et déjà, nous étendons la définition d'un schéma de

données en y associant un ensemble de CIs structurelles. Néanmoins, avec les schémas étendus, on ne peut pas saisir toutes les propriétés des données. Une partie de ces propriétés de données peut être exprimée par des contraintes supplémentaires sur les instances du schéma de la base, c'est à dire sur les états de la BD. Dans ce dernier cas, nous appelons CIs **internes** les CIs qui peuvent être décrites, séparément de la description de la structure de données, par un langage au moment de la définition de BD [DELOBEL82] [FLORY82] [MIRANDA78]. Ces CIs permettent de traiter l'aspect sémantique d'un problème dans l'approche BD.

Parmi ces CIs, nous pourrons distinguer à nouveau deux sortes de CIs selon leur lien avec le temps: une CI statique est une contrainte qui définit un état valide de la base tandis qu'une CI dynamique définit la validité des changements d'états de la base, par référence aux états antérieurs de la base [FOUCAUT82] (cf. [DELOBEL82] [FLORY82]).

Enfin pour compléter la classification des CIs, nous appelons CIs **implicites**, des propriétés qui sont les conséquences logiques d'autres CIs, par exemple, des dépendances fonctionnelles déduites d'un ensemble de celles-ci et des inférences logiques déduites d'un ensemble d'assertions [BRODIE84-a].

Après avoir introduit certaines notions de BD, nous allons présenter le modèle relationnel en vue d'en intégrer les concepts dans le langage LTR-3 [CIMSA83]. La présentation qui suit est orientée vers la spécification du langage proposé R-LTR, au lieu d'être une présentation générale du modèle relationnel.

2. Modèle relationnel de données (Aspect syntaxique)

2.1. Définitions des structures d'objet

2.1.1. Objet, rubrique et relation

Lorsqu'on veut décrire un monde réel dans un système d'informations, il faut d'abord commencer par examiner les objets à traiter. "Qu'est-ce qu'un objet ?" Cette question est primordiale pour toute démarche scientifique mais la réponse à cette question dépend totalement du contexte dans lequel on se place. Une définition possible en est: "un découpage commode" en le personnifiant et en le regardant comme un tout [BOLL80]. Cela ne signifie nullement qu'il n'est pas décomposable en parties distinctes. Bien spécifier le point de vue duquel on se place et indiquer jusqu'à quel degré l'analyse sera effectuée, sont importants.

Pour montrer le fait que l'emploi du mot "objet" est dépendant du contexte, prenons deux exemples. Dans le modèle conceptuel OLE (Objet-Lien-Entité, défini à titre pédagogique), un objet est considéré comme la plus petite abstraction qui a une signification et qui n'existe que comme composant d'une entité [MIRANDA84]. En revanche, dans le modèle E-A (Entite-Association), un objet est considéré comme une entité qui est quelque chose perceptible par rapport à une sémantique du monde reel et dont l'existence peut être identifiée. Les propriétés d'une entité sont représentées par un ensemble d'attributs [CHEN76].

Nous proposons une définition d'un objet comme l'être-objet ou l'élément d'intérêt (sur lequel porte l'étude) à chaque niveau d'analyse qui constitue la préoccupation principale pour ce niveau. Ici ce sera une représentation uniforme de l'information à traiter à chaque niveau d'analyse.

En ce qui concerne le modèle relationnel, il y a deux niveaux d'analyse: domaines et relations entre eux. Un domaine est un ensemble d'objets élémentaires, non-décomposables et homogènes sur lequel

est basée la définition de relation étant l'objet plus complexe.

Dans [FLORY82], l'auteur introduit le concept de "rubrique" comme l'élément de base qui rejoint celui d'Objet du modèle OLE.

> Une **rubrique** représente le plus petit lot d'informations qui puisse être utilisé d'une façon autonome et qui a un sens indépendamment des autres lots [FLORY82,pp.37-43].

L'intérêt de ce concept face à celui de domaine est qu'une rubrique peut être a priori définie indépendamment de la relation à laquelle elle appartient. Une rubrique est représentée par un couple: un nom et un ensemble de valeurs possibles. A partir de ces objets élémentaires, on peut définir un objet complexe, "relation" sur laquelle est basée la modélisation du monde réel. Dans la suite, le concept de rubrique sera substitué par ceux d'attribut d'une relation et domaine de valeurs. En fait, "relation" est l'objet sur lequel notre analyse sera effectuée.

2.1.2. Relation, tuple, domaine, attribut et clé

Le modèle relationnel de données, proposé par [CODD70], consiste à représenter des données par la notion mathématique de relation. Les motivations importantes de la recherche sur le modèle relationnel sont les suivantes [CODD82]:

- permettre une indépendance entre l'aspect logique et l'aspect physique de la gestion d'une BD;
- fournir des schémas de données simples à utiliser;
- introduire des langages de haut niveau permettant une manipulation des informations de manière non-procédurale.

Un **domaine** est un ensemble de valeurs de même type. Le modèle relationnel est basé sur la notion mathématique d'une **relation** de degré n qui est définie comme un sous-ensemble du produit cartésien $D_1 x D_2 x \ldots x D_n$ de n (n>0) domaines $D_1, D_2, \ldots, D_n$ non nécessairement distincts. Un **attribut** A_i est un identificateur par lequel on représente l'usage (appelé aussi rôle) d'un domaine D_i dans une relation R. Nous

notons une relation ainsi: $R[A_1:D_1,A_2:D_2,\ldots,A_n:D_n]$ où R est le nom de relation et A_i est un attribut associé à un domaine D_i pour i=1,2,...,n. Par convention, nous ne spécifions pas le domaine d'un attribut lorsqu'il n'y a pas d'ambiguité de compréhension: $R[A_1,A_2,\ldots,A_n]$.

Un élément de la relation de degré n est appelé un **tuple** noté $<A_1:v_1,A_2:v_2,\ldots,A_n:v_n>$ où A_i est un attribut et v_i est une valeur appartenant à D_i pour i=1,2,...,n. Autrement dit, un tuple est un n-uplet dont les composants sont étiquetés.

Une **clé de relation**† est un attribut ou un ensemble d'attributs dont les valeurs sont discriminatoires. Elle est utilisée pour identifier chaque tuple de manière unique dans une relation. Une relation comporte toujours au moins une clé, mais il peut y en avoir plusieurs. Dans ce cas, elles sont appelées **clés candidates**. On laissera à l'utilisateur le choix d'une clé sémantiquement plus significative comme **clé principale**‡ dont la valeur ne peut être modifiée au cours des applications.

2.1.3. Contraintes d'intégrité inhérentes

Dans l'introduction, nous avons défini des CIs inhérentes comme des propriétés intégrantes du modèle relationnel à respecter lors de l'application du modèle à une organisation. Nous donnons une liste de CIs inhérentes; d'abord, nous commençons par donner une CI inhérente sur le type des valeurs d'un attribut.

(1) Chaque domaine est simple dans le sens où chaque élément du domaine est atomique (non-décomposable vis-à-vis du modèle de données). Autrement dit, un élément d'un domaine simple est toujours considéré comme une valeur, non pas comme un ensemble de valeurs ou comme une relation. Cette contrainte permet de représenter une

(†) "index" [FLORY82] et "identifiant" [TARDIEU85] sont utilisés comme synonymes de clé.
(‡) On emploie aussi le terme "clé primaire" ("primary key") à la place de "clé principale" [CODD79] [DATE82].

rubrique du monde réel par un attribut associé à un domaine simple. (Première forme normale)

(2) L'ordre des attributs dans une définition de relation n'est pas significatif, car chaque attribut d'une relation a un nom distinct.

(3) L'ordre des tuples d'une relation n'est pas significatif et il n'y a pas de duplication des tuples dans une relation (par définition). Cette propriété a directement pour conséquence l'existence de clé dont la valeur ne peut être indéfinie (ou "null" en anglais). (cf. CIs (4) et (6))

(4) Identification unique: Pour chaque tuple d'une relation, la valeur de clé identifie uniquement le tuple concerné; c'est à dire qu'il n'y a pas deux tuples qui partagent la même valeur de clé.

(5) Non-redondance de clé: Aucun attribut qui fait partie de la clé ne peut être supprimé sans détruire la propriété (4); c'est à dire que la clé est minimale. (Deuxième forme normale)

Les propriétés des clés seront détaillées en termes de dépendances fonctionnelles dans le paragraphe 3.1.. Dans [CODD79], l'auteur propose deux CIs posées sur la clé principale comme règles de mise à jour de données: CI d'entité et CI de référence.

(6) CI d'entité: Tous les attributs composant la clé principale d'une relation de base doivent avoir des valeurs bien déterminées (non pas indéfinies).

(7) CI de référence: Dans une relation R_1, s'il existe un attribut A qui est déjà défini comme clé principale d'une relation R_2, alors chaque valeur de A dans R_1 doit figurer comme valeur de la clé principale de R_2 (R_1 et R_2 ne sont pas nécessairement distinctes).

Dans la propositon de [CODD79], l'attribut A fait partie de la clé principale de R_1. [DATE82] a étendu cette CI comme ci-dessus avec la possibilité d'avoir la valeur nulle pour l'attribut A de la relation R_1.

2.2. Schéma relationnel

Un **schéma de relation** est un couple $(R[A_1,A_2,\ldots,A_m],\Sigma)$ où $R[A_1,A_2,\ldots,A_m]$ désigne le type de relation nommé R et construite sur l'ensemble d'attributs $A_1,A_2,\ldots,A_m$ et où Σ désigne un ensemble de contraintes d'intégrité. Comme ensemble de CIs, nous prenons celui de dépendances fonctionnelles entre attributs parce que ce concept intervient particulièrement dans le choix des clés de relation. Nous le décrivons plus précisément dans le paragraphe 3.1.. L'état d'un schéma de relation (l'instance ou la valeur d'un schéma de relation [PIROTTE82]) est un ensemble de tuples qui satisfont la spécification de schéma de relation. Dans la suite nous utilisons le terme "relation" pour désigner l'état d'un schéma de relation, s'il n'y a pas d'ambiguité.

Un **schéma relationnel** est un ensemble de schémas de relation avec un ensemble de contraintes d'intégrité inter-relations, noté $(\{R_i[U_i,\Sigma_i]|i=1,2,\ldots,n\}, SIGMA)$ où $U_i=A_{i_1},A_{i_2},\ldots,A_{i_{mi}}$. En fait un schéma relationnel est l'expression du schéma conceptuel d'une BD relationnelle.

2.3. Expression relationnelle (valeurs indéfinies exclues)

La partie "interrogation de données" est généralement plus importante dans un langage de manipulation de données puisque la partie "mise à jour" est souvent directement définie par un ensemble d'opérations concernant l'insertion, la suppression et la modification des données.

Une algèbre relationnelle [CODD70,72-b] comprend un ensemble d'opérations qui prennent une ou deux relations comme opérandes et produisent une relation comme résultat. Par exemple, {PRODUIT-CARTESIEN, UNION, DIFFERENCE, PROJECTION, SELECTION} est un ensemble d'opérateurs primitifs et il est relationnellement complet [CODD72-b] [DATE82] dans le sens où toutes les informations qu'on souhaite

extraire peuvent s'exprimer par la combinaison de ces opérateurs. L'algèbre relationnelle permet de spécifier des requêtes sur une BD relationnelle à l'aide des **expressions relationnelles** qui sont des enchaînements des opérations relationnelles appliquées à des relations.

D'autres langages concernent le calcul relationnel. Le calcul relationnel orienté à tuple utilise des variables de type tuple avec des prédicats définissant les tuples d'intérêt [CODD72-b], alors que le calcul relationnel orienté à domaine remplace les variables de type tuple par celles de type domaine [LACROIX77].

Ici nous considérons un langage d'interrogation basé sur l'algèbre relationnelle, qui nous paraît facile de l'introduire dans un langage de programmation à l'aide de la notion de type abstrait de données. Une interrogation sur BD est effectuée sous forme d'une affectation d'expression relationnelle à une relation temporaire. Une expression relationnelle se compose d'opérandes et opérateurs.

2.3.1. Opérandes

Nous pouvons distinguer trois sortes d'opérandes: relation de base, relation temporaire et sous-expression (relation intermédiaire).

Une **relation de base** est une relation qui fait partie du schéma de la BD et dont l'existence est permanente tant que la base existe. Chaque relation de base est spécifiée par un schéma de relation dont son état peut être modifié par des opérations de mise à jour.

Une **relation temporaire** ne fait pas partie du schéma de la base mais elle est destinée à (i) stocker le résultat obtenu à partir d'une expression relationnelle et à (ii) renommer ou copier une relation de base (cf. "marking" [WASSERMAN79], "relation dérivée" ou "vue" [DELOBEL82,p.179] [PIROTTE82]).

Une **relation intermédiaire** représente le résultat interne de chaque étape de l'évaluation d'une expression relationnelle qui est spécifié par une liste d'attributs et par un ensemble de tuples (cf.

relation dénotée par requêtes [PIROTTE82]). Dans le contexte du langage de programmation, ce type de relation sera remplacé par une sous-expression relationnelle.

2.3.2. Opérations choisies

Pour l'explication détaillée de la sémantique de chaque opérateur, rappelons que $R[A_1:D_1,A_2:D_2,\ldots,A_n:D_n]$ désigne une relation où $A_1,A_2,\ldots,A_n$ sont des attributs et $D_1,D_2,\ldots,D_n$ sont les domaines pour chaque attribut correspondant. Un tuple $<A_1:a_1,A_2:a_2,\ldots,A_n:a_n>$ est défini si chaque a_i est une valeur du domaine D_i pour A_i.

Opérations ensemblistes: UNION, INTERSECT et DIFFER

Etant données deux relations $R[A_1:D_1,A_2:D_2,\ldots,A_n:D_n]$ et $S[B_1:D'_1,B_2:D'_2,\ldots,B_n:D'_n]$. On dira que R et S sont union-compatibles par rapport à la spécification de correspondance $C=\{(A_i,B_j)|\ i,j\in\{1,2,\ldots,n\}\}$ si, quelque soit h et k tels que $h\neq k$, $A_h\neq A_k$ et $B_h\neq B_k$ (c'est à dire un attribut de chaque relation est mentionné une seule fois pour toute dans C), et que pour chaque couple (A_i,B_j), les attributs A_i et B_j ont le même domaine. Alors chaque opération UNION(R,S,C), INTERSECT(R,S,C) et DIFFER(R,S,C) effectue respectivement l'opération union, intersection et différence de deux relations R et S, exactement de la même manière que les opérations ensemblistes "$\cup$", "$\cap$" et "-".

Expliquons, par exemple, le cas de UNION. Le résultat de UNION(R,S,C) est une relation intermédiaire qui satisfait les deux conditions suivantes:

(1) La liste d'attributs du résultat est $(U_1:D_1,U_2:D_2,\ldots,U_n:D_n)$ où $U_i=A_i$ pour $i=1,2,\ldots,n$;

(2) Le tuple $<U_1:u_1,U_2:u_2,\ldots,U_n:u_n>$ appartient à l'ensemble de tuples de UNION(R,S,C), si et seulement si, $<A_1:u_1,A_2:u_2,\ldots,A_n:u_n>$ appartient à R ou $<B_{i_1}:u_1,B_{i_2}:u_2,\ldots,B_{i_n}:u_n>$ appartient à S et $(A_k, B_{i_k})\in C$.

La liste d'attributs du résultat est un choix de préférence. U_i peut être un des A_i et B_i, ou bien peut être renommé. Dans [DATE82], l'auteur propose de choisir celui de A_i ou B_i qui précéde l'autre dans l'ordre alphabétique. Nous choisissons A_i de la première relation R pour U_i.

Opérations relationnelles: SELECT, PROJECT et JOIN

Si θ est un des opérateurs arithmétiques de comparaison et si c est un attribut A_j ($j \neq i$) ou une valeur du domaine D_i, alors le résultat de SELECT(R,$A_i \theta c$) est une relation intermédiaire satisfaisant les conditions suivantes:

(1) La liste d'attributs du résultat est égale à celle de R;

(2) Le tuple $<A_1:a_1, A_2:a_2, \ldots, A_n:a_n>$ est inclus dans le résultat si et seulement si lorsque $<A_1:a_1, A_2:a_2, \ldots, A_n:a_n>$ appartient à R, "$a_i \theta c$" est vrai dans le sens $a_i \theta c$ si c est une valeur sinon $a_i \theta a_j$.

Etant donné qu'une séquence d'attributs $<A_{i_1}, A_{i_2}, \ldots, A_{i_j}>$ est un sous-ensemble de $<A_1, A_2, \ldots, A_n>$ de R, le résultat de PROJECT(R,$<A_{i_1}, A_{i_2}, \ldots, A_{i_j}>$) est une relation intermédiaire qui satisfait les deux conditions suivantes:

(1) La liste d'attributs du résultat est $(A_{i_1}:D_{i_1}, A_{i_2}:D_{i_2}, \ldots, A_{i_j}:D_{i_j})$ où chaque D_k pour $k=i_1, i_2, \ldots, i_j$ correspond à D_k de $(A_1:D_1, A_2:D_2, \ldots, A_n:D_n)$ de R;

(2) Le tuple $<A_{i_1}:a_{i_1}, A_{i_2}:a_{i_2}, \ldots, A_{i_j}:a_{i_j}>$ est inclus dans le résultat si et seulement si $<A_1:a_1, \ldots, A_{i_1}:a_{i_1}, A_{i_2}:a_{i_2}, \ldots, A_{i_j}:a_{i_j}, \ldots, A_n:a_n>$ appartient à R.

En ce qui concerne l'opération JOIN, il s'agit de la jointure naturelle, c'est à dire d'une équi-jointure avec l'élimination d'un des deux attributs dupliqués [CODD70] [DATE81]. Soit

$R[A_1:D_1,A_2:D_2,\ldots,A_m:D_m]$ et $S[B_1:D'_1,B_2:D'_2,\ldots,B_n:D'_n]$ deux relations. Le résultat de $JOIN(R,S,A_i=B_j)$, où D_i et D'_j sont les mêmes domaines, est une relation intermédiaire satisfaisant les conditions suivantes:

(1) La liste d'attributs du résultat est $(A_1:D_1,\ldots,A_m:D_m,B_1:D'_1,\ldots,B_{j-1}:D'_{j-1},B_{j+1}:D'_{j+1},\ldots,B_n:D'_n)$;

(2) Le tuple résultant $<A_1:a_1,\ldots,A_m:a_m,B_1:b_1,\ldots,B_{j-1}:b_{j-1},B_{j+1}:b_{j+1},\ldots,B_n:b_n>$ appartient au résultat de $JOIN(R,S,A_i=B_j)$ si et seulement si $<A_1:a_1,\ldots,A_m:a_m>\in R$, $<B_1:b_1,\ldots,B_n:b_n>\in S$ et $a_i=b_j$.

3. Contraintes d'intégrité (Aspect sémantique)

3.1. Contraintes d'intégrité structurelles et Normalisation des relations (Sémantique formelle)

Rappelons qu'un schéma de relation est $(R[U],\Sigma)$ où $U=\{A_1,A_2,\ldots,A_n\}$ dont Σ est l'ensemble de CIs structurelles, décrit par les dépendances fonctionnelles définies sur U. La troisième forme normale d'un schéma relationnel est liée à la notion de dépendance fonctionnelle. Parmi d'autre CIs structurelles proposées, on trouve aussi dépendance multivaluée [FAGIN77] et dépendance de jointure [AHO79] [FAGIN79]. Chacune permet d'aboutir respectivement à la quatrième et à la cinquième forme normale d'un schéma relationnel. Ici nous nous limitons à la dépendance fonctionnelle.

3.1.1. Dépendances fonctionnelles

Etant donné une relation $R[U]$, on dit qu'il existe une **Dépendence Fonctionnelle** (notée DF) de X à Y, noté $X\rightarrow Y$ où $X\subseteq U$ et $Y\subseteq U$, si et seulement si, quelques soient les tuples t_1 et t_2 de R, si $t_1[X]=t_2[X]$ alors $t_1[Y]=t_2[Y]$ où $t[X]$ indique la projection du tuple t sur l'ensemble d'attributs X. Une DF $X\rightarrow Y$ est élémentaire s'il n'existe pas $X'\subset X$ tel que $X'\rightarrow Y$.

Dans un schéma de relation $(R[U],\Sigma)$, un attribut A est **transitivement dépendant** d'un ensemble d'attributs $X \subseteq U$ s'il existe $Y \subseteq U$ ne contenant pas A tel que (i) $X \rightarrow Y$ (ii) $Y \rightarrow A$ (iii) $Y \not\rightarrow X$ (iv) $Y \not\subset X$ [DELOBEL82,p.407] (cf. [BERNSTEIN76]).

3.1.2. Troisième forme normale

Dans un schéma de relation $(R[U],\Sigma)$, un ensemble d'attributs $K \subseteq U$ est défini comme clé de relation si la DF élémentaire $K \rightarrow U$ est une conséquence logique de Σ.

Un schéma de relation $(R[U],\Sigma)$ sera en **Troisième Forme Normale** (notée 3FN) si tous les attributs non-clés sont non transitivement dépendants (on dit aussi directement dépendants) de la clé.

Un schéma de relation $(R[U],\Sigma)$ est en **Troisième Forme Normale de Boyce-Codd** (notée 3FNBC) si, chaque fois que $X \rightarrow A$, $A \notin X$ est vérifiée dans $(R[U],\Sigma)$ alors X contient une clé de $(R[U],\Sigma)$. Dans ce cas on peut avoir une autre notation d'un schéma de relation identique à $(R[U],\Sigma)$: $(R[U],\Omega)$ où $\Omega=\{K_1,K_2,\ldots,K_m\}$, pour $m \geq$, est un ensemble de clés candidates.

Un schéma relationnel est dit en 3FN (respectivement 3FNBC) si chaque schéma de relation $(R_i[U_i],\Sigma_i)$, $i=1,2,\ldots,n$, est en 3FN (respectivement 3FNBC)

3.1.3. Dépendances d'inclusion inter-relations

Etant donnés deux schémas de relation (non nécessairement distincts), $(R[A_1,A_2,\ldots,A_m],\Sigma_r)$ et $(S[B_1,B_2,\ldots,B_n],\Sigma_s)$. Si X est une séquence de k attributs distincts de $A_1,A_2,\ldots,A_m$ et si Y est une séquence de k attributs distincts de $B_1,B_2,\ldots,B_n$ et que X et Y sont union-compatibles, alors une **Dépendance d'Inclusion** (en abrégé DI), notée $R[X] \subset S[Y]$, est définie comme suit: un état consistant d'un schéma relationnel satisfait $R[X] \subset S[Y]$ si et seulement si pour tout tuple t de R, il existe un tuple u de S tel que $t[X]=u[Y]$ où $t[X]$ et $u[Y]$ sont les projections de t et u sur X et Y respectivement

[CASANOVA84-b] [NASSIF83]. Cette contrainte d'intégrité de DI est plus générale que la CI de référence expliquée dans le paragraphe 2.1.. Si, dans la définition de DI ci-dessus, Y est une clé principale de S alors la DI R[X]⊂S[Y] exprime la CI de référence entre R et S.

Pour un schéma relationnel $(\{(R_i[U_i],\Sigma_i)|i=1,2,,...,n\},SIGMA)$ nous choisissons un ensemble de DIs pour SIGMA. Si un schéma relationnel est en 3FNBC, nous pouvons le noter comme $(\{(R_i[U_i],\Omega_i)|i=1,2,...,n\},SIGMA)$ où $\Omega_i=\{K_1,K_2,...,K_{m_i}\}$.

3.2. Contraintes d'intégrité internes (Sémantique informelle)

La modélisation du monde réel par le modèle de données relationnel, comme tout modèle, comporte l'abstraction de la réalité sous la forme de structure de données avec des opérations associées. La forme de l'abstraction implique des limites à la représentation des réalités. Ces deux points, abstraction et limite à la représentation, sont les aspects opposés de la modélisation. Sans prendre trop de risque, on peut dire que le modèle relationnel est un modèle sain en égard à l'abstraction grâce à sa simplicité de structuration de données (relation) et à son langage complet (algèbre, calcul relationnel). En général, plus un modèle de données est élaboré en terme de structure, plus les CIs inhérentes sont incorporées au modèle, et moins les CIs explicites peuvent être spécifiées [TSICHRITZIS82].

Puisque toutes les CIs, notamment CIs internes, ne peuvent être intégrées au modèle, il est nécessaire de compléter le schéma de données par un ensemble de CIs qui sont gérées par un système d'intégrité (une partie du SGBD). Les rôles d'un système d'intégrités consistent, d'abord, à assurer la fiabilité des informations de la base, ensuite, à apporter un complément de l'ordre sémantique au schéma de données [FLORY82] [FOUCAUT82]. Afin d'accomplir ces rôles, un système d'intégrités doit assumer les responsabilités suivantes:

- surveiller les opérations de mise à jour (transactions) et détecter les violations des contraintes d'intégrité;
- en cas de violation, déclencher des actions adéquates [DATE83].

Nous proposons d'associer des CIs internes au modèle relationnel selon les niveaux de la structure de données: attribut, tuple, relation et base:

(1) Contraintes portant sur le domaine de valeurs d'un attribut;

(2) Contraintes sur un tuple d'une relation, autrement dit, celles portant sur les domaines de plusieurs attributs d'un tuple;

(3) Contraintes sur une relation (un ensemble de tuples), par exemple, la cardinalité d'une relation et la moyenne des valeurs d'un attribut d'une relation);

(4) Contraintes entre attributs de plusieurs relations, par exemple, la dépendance d'inclusion.

4. Conclusion

Rappelons que l'intérêt et le succès du modèle relationnel sont dûs:

- à sa structure simple et aux fondements théoriques qui rendent le modèle sain, et fournissent une méthode de normalisation pour l'organisation de la BD;
- aux langages de manipulation simples et puissants basés sur le calcul relationnel ou sur l'algèbre relationnelle;
- aux nombreux SGBDs relationnels, déjà développés, et qui valorisent le modèle.

Dans [CODD82], l'auteur cite quatre directions de travail à mener à court terme sur les systèmes relationnels: (i) améliorer l'aspect système d'intégrité dans un SGBD; (ii) fournir un support spécial pour l'ingénierie et les applications scientifiques; (iii) enrichir les catalogues de bases pour permettre des contrôles supplémentaires (description de schéma de données, description d'utilisateur, autorisation, etc.) et plus particulièrement pour utiliser le catalogue par

un même langage d'interrogation; (iv) aider à la conception de la BD aux niveaux logique et physique.

A long terme, il espère l'extension du modèle relationnel dans les directions suivantes:

- BD relationnelle distribuée sur un réseau de communication;
- extension du modèle pour appréhender plus de sémantique de données;
- développement de la théorie pour traiter les valeurs incomplètes et imprécises;
- traitement des données hétérogènes (ou multi-média) telles que l'image, le son, le texte, etc.;
- rapprochement entre langages de BD et langages de programmation, par exemple, en incorporant des types abstraits de données dans les langages de BD et des traitements relationnels dans les langages de programmation.

Cette liste peut être complétée par d'autre directions [GARDARIN85]:

- des BDs déductives où des règles permettant de dériver des informations à partir des relations de base sont ajoutées à la base;
- une meilleure convivialité des interfaces et une meilleure ergonomie des systèmes.

Toutes ces directions pourront être réalisées par l'intégration des concepts et des outils issus de l'intelligence artificielle et du génie logiciel. A l'heure actuelle, cette conjonction a déjà été manifestement entamée dans le milieu de la recherche informatique [BRODIE84] [MIRANDA86].

*
***
*****

CHAPITRE II - LANGAGE DE BASE DE DONNEES RELATIONNELLE: R-LTR

Résumé

Ce chapitre présente une extension du langage LTR-3, avec l'incorporation d'un langage de base de données relationnel. De nouveaux types de données TUPLE, RELATION et DATABASE sont proposés avec des opérations de mise à jour sur des relations (les instances du type RELATION). En particulier nous prenons en compte des contraintes d'intégrité internes à la définition d'une base de données. Pour l'interrogation sur des relations, nous utilisons des expressions de l'algèbre relationnelle. Toutes les nouvelles structures introduites dans le langage sont développées en vue d'être compatibles avec celles de LTR-3.

Sommaire

1. Introduction
2. Langage de définition de données avec contraintes d'intégrité
   2.1. Type de TUPLE
      1.2.1. Syntaxe
      1.2.2. Sémantique
      1.2.3. Exemple
      1.2.4. Fonctions prédéfinies pour les contraintes d'intégrité
   2.2 Déclaration de relations
      2.2.1. Syntaxe
      2.2.2. Sémantique
      2.2.3. Exemple
      2.2.4. Fonctions prédéfinies pour les contraintes d'intégrité
   2.3. Déclaration de base de données
      2.3.1. Syntaxe
      2.3.2. Sémantique
      2.3.3. Exemple
3. Langage de manipulation de données
   3.1. Opérations de mise à jour
   3.2. Interrogations de données par une expression relationnelle
      3.2.1. Syntaxe
      3.2.2. Sémantique
      3.3.3. Exemple
      3.3.4. Fonctions prédéfinies de calcul
4. Structure d'une application
5. Réalisation du préprocesseur R-LTR
6. Conclusion

CHAPITRE II

LANGAGE DE BASE DE DONNEES RELATIONNELLE: R-LTR

1. Introduction

Dans ce chapitre, nous proposons un langage de BD appelé R-LTR (LTR Relationnel), qui est une extension de LTR-3† destiné à la réalisation des applications de base de données relationnelle. Ce langage s'appuie sur:

- le modèle relationnel de données [CODD70];
- la notion de type abstrait de données [LISKOV74].

Tout d'abord, nous rappelons que LTR-3 est un langage de programmation modulaire plus particulièrement destiné à la programmation des applications temps réel. Sur le plan de la forme, une syntaxe de type PASCAL a été adoptée pour LTR-3. Sur le plan de la fonctionalité, les principaux amendements apportés au langage concernent, notamment le traitement du temps réel, la modularité, l'exception, le type paramétré, les entrées-sorties en langage de haut niveau, etc..

Depuis le début des années soixante-dix, d'abondantes études ont été menées sur les BDs relationnelles dont certaines ont abouti à la définition et à la réalisation de langages relationnels non-procéduraux. Des exemples de telles études incluent ILL [LACROIX77], QUEL [HELD75] [McDONALD81], SQUARE [BOYCE75], SQL [IBM82], TYP-R [TOUNSI83], etc..

Un peu plus tard, dans le domaine de la méthodologie de programmation, les notions de Type Abstrait de Données et de Modularité sont apparues et ont joué un grand rôle comme les principales disciplines de programmation. Nous pouvons citer certains langages de programmation basés sur ces concepts tels que MODULA [WIRTH77], CLU [LISKOV81],

(†) LTR-3 est une abréviation de "Langage de Temps Réel, Version 3". LTR est depuis 1974 le langage standard du Ministère de la Défense Français pour la programmation des systèmes opérationnels [CIMSA83].

ADA [ACM79], langage de TYP [CHABRIER82-a], LTR-3, etc.. En effet, la notion de type abstrait de données permet de caractériser un ensemble d'objets uniquement par la spécification des opérations que nous pouvons appliquer aux objets du type indépendamment de leur représentation [GUTTAG77] (ex. type opaque de LTR-3).

Récemment, un certain nombre de langages de BD inspirés par cette notion ont vu le jour, c'est le cas, par exemple, de RIGEL [PRENNER78] [ROWE79], PLAIN [WASSERMAN79,81] et PASCAL Relationnel [SCHMIDT77]. Ces langages contiennent le type prédéfini de données RELATION permettant de construire et d'utiliser les BDs relationnelles.

La plupart des langages de BD fournissent un langage de définition et un langage de manipulation. Ces langages sont incorporés dans un langage hôte (ex. SQL dans PL/1 [IBM82], QUEL dans le langage C [ALLMAN76] [WOODFILL84]), au contraire, ils sont développés comme un langage autonome (ex. PLAIN, RIGEL). En particulier PASCAL Relationnel est réalisé en modifiant le compilateur afin d'ajouter à PASCAL un nouveau type de données RELATION. Ce travail présente un avantage important puisqu'il fournit un seul langage pour résoudre des problèmes généraux et des problèmes d'applications de BD.

Dans ce qui suit, nous proposons une extension de LTR-3, nommé R-LTR. D'abord, nous introduisons deux nouveaux types de données: TUPLE et RELATION, afin de définir une BD relationnelle. Rappelons qu'une BD relationnelle peut être représentée par un schéma relationnel constitué d'un ensemble de schémas de relation et d'un ensemble de contraintes d'intégrité inter-relations; dans notre cas nous avons choisi des dépendances d'inclusion. En outre, nous proposons une clause WHERE en vue d'exprimer certaines contraintes d'intégrité internes à chaque niveau: tuple, relation et base.

Ensuite, nous présentons sur des exemples, les opérations de mise à jour et d'interrogation de la base. Nous supposons que la BD est en troisième forme normale afin d'éviter la redondance des informations stockées et pour bien appliquer les opérations de mise à jour sur des relations.

Nos objectifs sont:

- exploiter autant que possible les structures existantes du langage;
- minimiser le nombre de nouvelles structures du langage;
- simplifier le langage de BD et son implantation.

La présentation va suivre la forme du manuel de LTR-3 [CIMSA83]. La syntaxe présentée dans ce chapitre est simplifiée pour la compréhension immédiate. La syntaxe complète se trouve dans [CRU-ZLARA84].

[Remarque] La première proposition du langage R-LTR était faite dans [DERNIAME84]. En cours de réalisation, il y a eu un certain nombre d'améliorations et de modifications du langage. Nous en tenons compte ici.

[Notation] La signification des méta symboles pour la description syntaxique est résumée comme suit:

::=	est défini comme
|	ou
{}	peut être répété zéro ou plusieurs fois
{}+	peut être répété une ou plusieurs fois
{}0	est optionnel

2. Langage de définition de données avec contraintes d'intégrité

2.1. Type de TUPLE

2.1.1. Syntaxe

```
définition-type-tuple ::= TUPLETYPE nom-type-tuple :
                                TUPLE
                                        {attribut}+ ;
                                        {clause-WHERE-tuple}0
                                END TUPLE ;
attribut                ::= nom-attribut : type-domaine ;
clause-where-tuple      ::= WHERE  appel-fonction {, appel-fonction}
type-domaine            ::= types-simples-prédefinis

constante-tuple ::= < nom-attribut : constante-littérale
                    {, nom-attribut : constante-littérale } >
```

2.1.2. Sémantique

Un type de tuple permet de représenter un objet constitué d'un agrégat d'attributs qui ne sont pas nécessairement du même type (cf. le type RECORD de LTR-3). Un attribut est défini par un couple: nom de l'attribut et son domaine de valeurs. L'ordre des attributs dans la définition d'un type tuple est non significatif.

Un type de tuple hérite de l'opération "." permettant d'accéder à un attribut du tuple.

La clause WHERE permet d'exprimer les CIs internes qui font intervenir un ou plusieurs attributs d'un même tuple. Une clause WHERE contient après le mot clé WHERE une liste d'appels de fonction.

Une constante de tuple représente une valeur d'un tuple. Le nombre d'attributs et les types d'attributs doivent correspondre à la définition du type du tuple.

2.1.3. Exemple

```
CONST smic : INTEGER = 4400;
CONST min_sal : INTEGER = 5000;
CONST max_sal : INTEGER = 25000;
        :
        :
TUPLETYPE emp_tuple : TUPLE
                        num_employe     : INTEGER (1..num_maxemp);
                        nom             : STRING (25);
                        salaire         : REAL (min_sal..max_sal);
                        num_department  : INTEGER (1..num_maxdep);
                        num_directeur   : INTEGER (1..num_maxemp);
                      WHERE RESTRICT(salaire, "GE", smic)
                      END TUPLE;
        :
        :
```

Pour le type de tuple "emp_tuple" défini ci-dessus, on peut avoir une constante de tuple:

<num_employe:100, nom:"KIM M. J.", salaire:5000, num_departement:3, num_directeur:35>

[Note] Dans ce qui suit, nous utilisons le terme "tuple" à la place de "constante de tuple" s'il n'y a pas d'ambiguité.

2.1.4. Fonctions prédéfinies pour les contraintes d'intégrité

Pour imposer des CIs internes sur les attributs d'un tuple, nous avons à notre disposition le sous-type et le type-intervalle fournis par LTR-3. Néanmoins nous définissons une fonction prédéfinie RESTRICT qui restreint le domaine d'un attribut. Dans l'exemple du sous-paragraphe 2.1.3., nous avons montré qu'un utilisateur peut définir une CI, "le salaire d'un employé doit être supérieur ou égal à 'smic'" sous forme d'un appel de la fonction RESTRICT. De plus, nous proposons au niveau des tuples, une fonction prédéfinie [NASSIF83]:

- ORDER (attribut-1, opérateur, attribut-2)

qui permet d'assurer une contrainte entre les valeurs de deux

attributs, où l'opérateur est l'un des suivants:

- "EQ": égal à
- "NE": non égal à
- "GT": supérieur à
- "GE": supérieur ou égal à
- "LT": inférieur à
- "LE": inférieur ou égal à

Dans l'exemple précédent, si un employé ne peut pas être son propre directeur, on peut ajouter la CI suivante sur le type de tuple "emp_tuple" dans sa clause WHERE:

ORDER(num_employe, "NE", num_directeur)

2.2. Déclaration de relation

2.2.1. Syntaxe

```
déclaration-relation-de-base
                    ::= RELATION nom-relation-de-base :
                        [ KEY : nom-attribut {, nom-attribut} ]
                        OF nom-type-tuple
                        {clause-where-relation}0
                        {initialisation}0 ;
clause-where-relation ::= WHERE appel-fonction {, appel-fonction}
initialisation        ::= SIZE littéral-entier

déclaration-relation-temporaire
                    ::= RELATION  nom-relation-temporaire
                                  {, nom-relation-temporaire} OF * ;
```

2.2.2. Sémantique

Un type de relation représente un ensemble de tuples (plus précisément constantes de tuple). Le constructeur de type RELATION permet de déclarer une relation basée sur un type de tuple avec la désignation de la clé principale dont la notion représente un type important de CI appelé dépendance fonctionnelle (voir §3.1.1.).

La clause WHERE permet d'associer à une relation les CIs internes qui font intervenir les tuples d'une relation.

On peut donner le nombre maximum de tuples dans une relation par l'initialisation lors de sa déclaration.

Les relations ainsi définies sont des relations de base, car elles font partie du schéma de la base de données.

Contrairement les relations temporaires ne font pas partie du schéma de la base, mais elles sont utilisées pour contenir le résultat de l'évaluation d'une expression relationnelle, ou pour renommer une relation de base. La durée de vie des relations temporaires est en général celle du programme d'application qui les a créées.

2.2.3. Exemple

```
RELATION emp_rel : [KEY : num_employe] OF emp_tuple
                   WHERE NEW(salaire, "GT");
```

Une relation "emp_rel" basée sur le type de tuple "emp_tuple" est définie ainsi avec une CI interne qui exprime "le nouveau salaire est toujours supérieur à l'ancien" (voir §2.2.4.).

Si on désire déclarer deux relations temporaires: "rel_1" et "rel_2" pour les utiliser désormais, il suffit d'écrire la ligne d'instruction suivante:

```
RELATION rel_1, rel_2 OF *;
```

Le type et la valeur d'une relation temporaire ainsi declarée sont fournis par le résultat de l'évaluation d'une expression relationnelle dans un programme d'application. Ceci est analogue à un paramètre effectif de type indéfini indiqué par le symbole "*" dans le cadre de type paramétré de LTR-3.

2.2.4. Fonctions prédéfinies pour les contraintes d'intégrité

Au niveau des relations, nous proposons quatre fonctions prédéfinies qui permettent d'exprimer certaines CIs internes faisant intervenir un tuple ou l'ensemble des tuples d'une relation. Les opérateurs utilisés dans chaque fonction sont ceux de comparaison (voir §2.1.5.).

- **MEAN (nom-attribut, opérateur, valeur)** fait intervenir le fait que la moyenne des valeurs d'un attribut (le premier argument) de la relation doit avoir la signification de l'opérateur (le deuxième argument) par rapport à une certaine valeur donnée (le troisième argument).

Par exemple, si, dans une entreprise, on décide que la moyenne des salaires des employés doit être supérieure ou égale à 5000 francs, on peut poser la CI:

MEAN (salaire, "GE", 5000)

- **NOCCS (nom-attribut, opérateur, valeur)** fait intervenir le fait que le nombre d'occurrences différentes des valeurs d'un attribut doit avoir la signification de l'opérateur par rapport à la valeur donnée.

Par exemple, si l'équipe de direction est formée d'au plus dix directeurs, on peut poser la CI:

NOCCS (num_directeur, "LE", 10)

- **NOCCM (nom-attribut-1, nom-attribut-2, opérateur, valeur)** fait intervenir le fait que pour une valeur de l'attribut-1, le nombre des différentes valeurs de l'attribut-2 doit avoir la signification de l'opérateur par rapport à la valeur donnée.

Par exemple, si un directeur ne peut avoir plus de 20 employés sous sa responsabilité immédiate, on peut l'exprimer par la CI:

NOCCM (num_directeur, num_employe, "LE" 20)

- **NEW (nom-attribut, opérateur)** fait intervenir une CI dynamique dans le sens où un changement d'état de relation se fait par référence à l'état antérieur. La nouvelle valeur d'un attribut doit être la signification de l'opérateur par rapport à l'ancienne valeur de cet attribut, pour tous les tuples d'une relation sur laquelle cette contrainte est posée.

Pour l'exemple de cette dernière fonction, si le nouveau salaire d'un employé doit être supérieur à l'ancien, on peut poser la CI déjà illustrée dans l'exemple du sous-paragraphe 2.2.3..

2.3. Déclaration de base de données

2.3.1. Syntaxe

```
déclaration-bd ::= DATABASE nom-bd OF [ nom-relation-de-base
                                        {, nom-relation-de-base} ]
                  {WHERE appel-fonction-isin {, appel-fonction-isin} }0 ;
appel-fonction-isin ::= ISIN (nom-attribut-qualifié,nom-attribut-qualifié)
nom-attribut-qualifié ::= nom-relation-de-base . nom-attribut
```

2.3.2. Sémantique

Comme nous l'avons défini dans le CHAPITRE I, un schéma relationnel est composé de l'ensemble des schémas de relation et de celui des

dépendances d'inclusion (DIs) qui expriment des CIs de référence. En vertu de l'étude décrite dans le CHAPITRE IV, nous pouvons obtenir un schéma relationnel en 3FN y compris un ensemble de DIs qui garantit la cohérence de données au moment de la mise à jour des données.

Le langage permet de déclarer une BD constituée de l'ensemble de relations et celui de DIs définies entre un attribut d'une relation référente et un attribut d'une relation référée. Chaque DI peut être exprimée par un appel de la fonction prédéfinie ISIN dans la clause WHERE.

2.3.3. Exemple

```
% ----------------------------------------------------------------------
% Exemple de la définition d'une base de données, tiré de [DATE81,p.64],
% utilisant le langage R-LTR.
% ----------------------------------------------------------------------
% --- définitions des types de tuple
% ----------------------------------------------------------------------
TUPLETYPE fournisseur_tuple:
                TUPLE
                        num_f:INTEGER(1..35);
                        nom_f:STRING(20);
                        status:INTEGER(1..10);
                        ville:STRING(15);
                END TUPLE;
% -----
TUPLETYPE piece_tuple:
                TUPLE
                        num_p:INTEGER(1..200);
                        nom_p:STRING(20);
                        poids:REAL;
                        couleur:STRING;
                END TUPLE;
% -----
TUPLETYPE fp_tuple:
                TUPLE
                        num_f:INTEGER(1..35);
                        num_p:INTEGER(1..200);
                        quantite:INTEGER;
                END TUPLE;
```

```
% ------------------------------------------------------------------
% --- déclarations des relations
% ------------------------------------------------------------------
RELATION fournisseur : [KEY: num_f] OF fournisseur_tuple;
RELATION piece : [KEY : num_p] OF piece_tuple;
RELATION f_p : [KEY : num_f, num_p] OF fp_tuple;

% ------------------------------------------------------------------
% --- déclaration de base de données
% ------------------------------------------------------------------
DATABASE gerer-piece OF [fournisseur, piece, f_p]
                WHERE   ISIN(f_p.num_p, fournisseur.num_f),
                        ISIN(f_p.num_p, piece.num_p);
% ------------------------------------------------------------------
% fin de l'exemple
% ------------------------------------------------------------------
```

3. Langage de manipulation de données

3.1. Opérations de mise à jour

Trois opérations sont définies pour l'insertion, la suppression et la modification d'un tuple dans une relation:

- INSERT(nom-relation-de-base, constante-tuple);
- DELETE(nom-relation-de-base, valeur-clé);
- MODIFY(nom-relation-de-base, valeur-clé, spec-modification);

où spec-modification est un ensemble de couples: nom-attribut et sa nouvelle valeur.

On ne peut pas modifier la valeur d'une clé de relation. Cependant on peut supprimer un tuple désigné par la valeur de sa clé à moins que cette valeur ne soit référencée par une autre relation. Ce qui correspond à l'intégrité de référence (voir le CHAPITRE I, §2.1.3.) exprimée par la fonction prédéfinie ISIN.

(Exemple)

INSERT (founisseur, <num_f:30, nom_f:"Pierre Dupont", status:3, ville:"Nancy">);

DELETE (f_p, [num_f:18, num_p:108]);

MODIFY (piece, [num_p:33], <poid:203.50>);

Lors d'une opération de mise à jour sur une relation, des CIs explicitement décrites dans les clauses WHERE seront vérifiées selon les règles indiquées dans la Fig.II-1. Si une opération de mise à jour viole au moins une CI, l'opération ne s'effectue pas.

CI mode	CI sur attribut (ex. contrôle de type, RESTRICT)	CI sur tuple (ex. ORDER)	CI sur relation (ex. NOCCS, NOCCM)	CI dynamique (ex. NEW)
INSERT	O	O	O	X
DELETE	X	X	O	X
MODIFY	O	O	O	O

CI sur base (ex. ISIN)	
rel. référente	rel. référencée
O	X
X	O
O	O

O: vérification effectuée

X: pas de vérification

(Fig.II-1) Règles de vérification des CIs internes

3.2. Interrogation de données par une expression relationnelle

3.2.1. Syntaxe

```
interrogation-bd
        ::=     nom-relation-temporaire := expression-relationnelle ;

expression-relationnelle
        ::=     nom-relation
                | opération-union
                | opération-differ
                | opération-intersect
                | opération-select
                | opération-project
                | opération-join
```

```
opération-union
        ::=  UNION     ( expression-relationnelle , expression-relationnelle ,
                         correspondance )
opération-intersect
        ::= INTERSECT  ( expression-relationnelle , expression-relationnelle ,
                         correspondance )
opération-differ
        ::=  DIFFER    ( expression-relationnelle , expression-relationnelle ,
                         correspondance )
opération-select
        ::=  SELECT    ( expression-relationnelle , théta-comparaison )
opération-project
        ::=  PROJECT   ( expression-relationnelle , liste-attributs )
opération-join
        ::=  JOIN      ( expression-relationnelle , expression-relationnelle ,
                         specification-join )
        :
```

(Pour le détail de la syntaxe, voir [CRUZLARA84])

3.2.2. Sémantique

Une interrogation de la base de données sera effectuée sous la forme d'affectation d'une expression relationnelle à une relation temporaire. Une relation temporaire sera élaborée après l'évaluation d'une expression relationnelle dont les opérandes peuvent être des relations de base, des relations temporaires ou des sous-expressions relationnelles. Les relations temporaires héritent du type du résultat de l'évaluation de la partie droite (expression relationnelle) d'une affectation.

Les opérateurs relationnels sont UNION, DIFFER, INTERSECT, SELECT, PROJECT et JOIN. Leur sémantique est précisée dans le CHAPITRE I, le sous-paragraphe 2.3.2.. Il n'y a pas de priorité entre les opérateurs relationnels sauf celle imposée par le parenthésage dans la mesure où nous employons la notation préfixée.

Nous venons d'introduire un troisième type de relation: relation intermédiaire. Ce type de relation n'est que le résultat interne des opérations relationnelles qui est normalement transparent à l'utilisateur. Si on voulait garder une telle relation, il faudrait l'affecter à une relation temporaire.

3.2.3. Exemple

(1) Interrogation sur plus d'une relation:

Q: Pour chaque pièce fournie, obtenir son numéro et les noms des villes d'où vient une telle pièce.

```
R:  rel_1 := PROJECT (JOIN (f_p, fournisseur,
                            f_p.num_f=founisseur.num_f)
                     <num_p, ville>);
```

(2) Interrogation contenant l'opération JOIN sur la même relation:

Q: Obtenir toutes les paires de numéros de fournisseurs différents se trouvant dans la même ville.

```
R:  rel_1 := PROJECT (fournisseur, <num_f, ville>);
    rel_2 := PROJECT (SELECT (JOIN (fournisseur, rel_1,
                                    fournisseur.ville = rel_1.ville)
                              fournisseur.num_f < rel_1.num_f),
                      <fournisseur.num_f, rel_1.num_f>);
```

(3) Interrogation utilisant l'opération UNION:

Q: Obtenir les numéros des pièces pésant plus de 10Kg ou fournies par le fournisseur numéro 33.

```
R:  rel_1 := UNION (PROJECT (SELECT (piece, poids > 10), <num_p>),
                    PROJECT (SELECT (f_p, num_f = 33), <num_p>),
                    <(piece.num_p, f_p.num_p)>);
```

3.2.4. Fonctions prédéfinies de calcul

La plupart des langages de BD offrent des possibilités pour calculer des résultats globaux sous le nom de "fonction d'agrégation" (ex. DAPLEX [SHIPMAN81], QUEL) ou "fonction prédéfinie utilitaire" (ex. ILL, SQL).

Nous proposons 5 fonctions de calcul définies sur le domaine d'un attribut:

- COUNT (expression-relationnelle)
- SUM (expression-relationnelle, attribut)
- AVERAGE (expression-relationnelle, attribut)
- MAXI (expression-relationnelle, attribut)
- MINI (expression-relationnelle, attribut)

La fonction COUNT retourne la cardinalité du résultat de l'évaluation d'une expression relationnelle, c'est à dire, le nombre de tuples résultants. SUM retourne la somme des valeurs d'un attribut du résultat de l'évaluation d'une expression relationnelle. AVERAGE calcule la moyenne des valeurs d'un attribut du résultat de l'évaluation d'une expression relationnelle. AVERAGE est effectivement définie comme SUM/COUNT. Pour SUM et AVERAGE, le domaine d'un attribut spécifié doit être de type numérique (ex. INTEGER ou REAL). MINI et MAXI retournent respectivement une valeur minimale et celle maximale d'un attribut du résultat de l'évaluation d'une expression relationnelle.

(Exemples)

(1) Obtenir le nombre de fournisseurs de la pièce nommée "engrenage".

```
VAR nombre_fournisseur : INTEGER;
:
:
nombre_fournisseur
       := COUNT (SELECT (JOIN (piece, f_p, piece.num_p = f_p.num_p),
                         piece.nom_p = "engrenage"),
                 f_p.num_f);
```

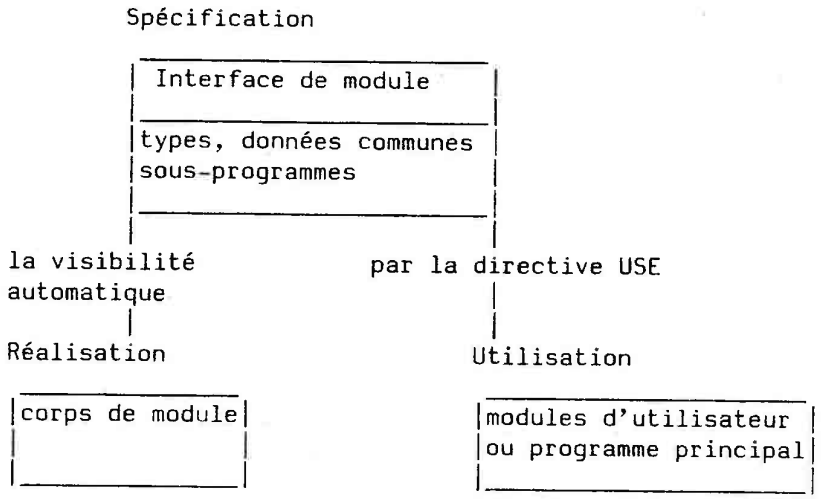
(2) Obtenir la quantité totale fournie de la pièce numérotée 23.

```
VAR quantite_fournie : REAL;
:
:
quantite_fournie := SUM (SELECT (f_p, num_p = 23), quantite);
```

4. Structure d'une application

Dans le cadre de LTR-3, une application est constituée d'un ensemble de modules et d'un programme principal, tous devant avoir un nom différent. Un module se subdivise en deux parties: une interface de module et son corps (celui-ci est optionnel). Il est plausible de considérer une interface comme une partie de spécification et son corps comme une partie de réalisation de l'interface.

Entre une interface et son corps de module, la visibilité de nouveaux types, objets et sous-programmes définis dans l'interface s'étend automatiquement au corps. D'autre part, on peut utiliser la directive USE pour étendre la visibilité à d'autres modules d'utilisateur ou au programme principal. La Fig.II-2 illustre ces notions.



(Fig.II-2) La structure d'une application en LTR-3

L'utilisateur peut définir des BDs dans une interface de module. Pour l'utilisation des bases, il peut les considérer comme des objets externes et les connecter aux programmes d'application à l'aide de la directive USE en tête de programme. Il faut signaler que si l'interface contenant une définition des BDs est recompilée, tous les programmes d'application utilisant cette définition doivent être recompilés à cause des règles de recompilation de LTR-3.

Les fonctions prédéfinies, y compris les fonctions utilitaires, sont supposées être déclarées dans le module "database_standard". Une clause "USE database_standard;" sera explicite en tête de chaque module d'application de BD.

(Exemple)

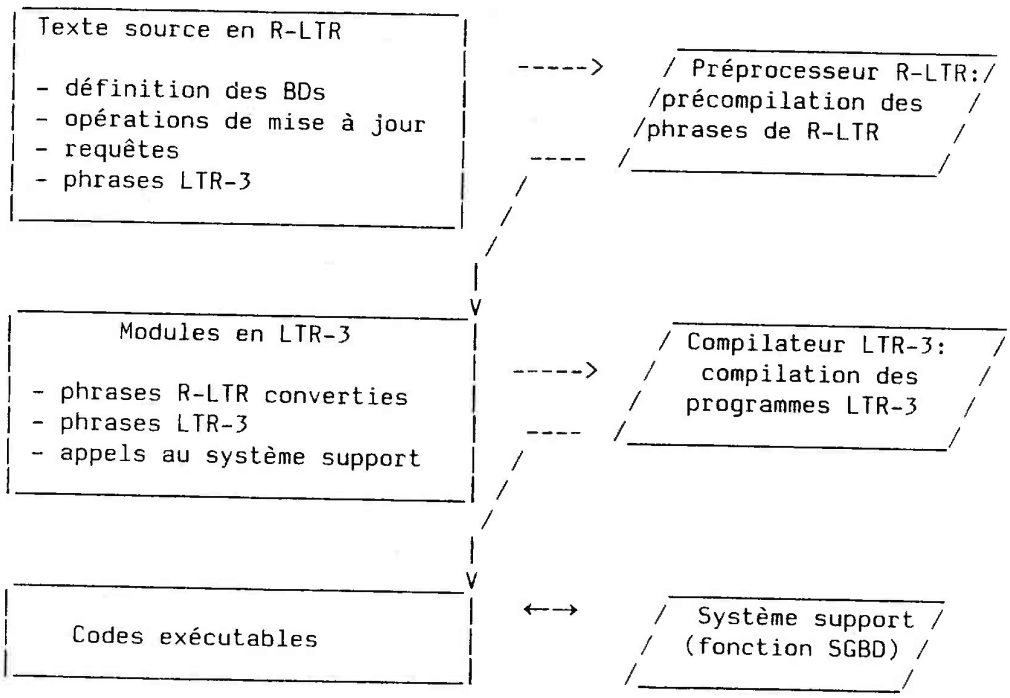
```
% ------------------------------------------------------------------
% Exemple de la définition d'une BD dans une interface de module
% ------------------------------------------------------------------
INTERFACE OF definition_gerer_piece
        USE database_standard;
        TUPLETYPE fournisseur_tuple :
                TUPLE
                        num_f : INTEGER(1..35);
                        nom_f : STRING(20);
                        status : INTEGER(1..10);
                        ville : STRING(15);
                END TUPLE;
        TUPLETYPE piece_tuple :
                :
                :
        (le contenu de l'Exemple 2.3.3.)
                :
                :
% ------------------------------------------------------------------
% --- déclarations des relations
% ------------------------------------------------------------------
RELATION fournisseur : [KEY: num_f] OF fournisseur_tuple;
RELATION piece : [KEY : num_p] OF piece_tuple;
RELATION f_p : [KEY : num_f, num_p] OF fp_tuple;
% ------------------------------------------------------------------
% --- déclaration de base de données
% ------------------------------------------------------------------
DATABASE gerer-piece OF [fournisseur, piece, f_p]
                WHERE   ISIN(f_p.num_p, fournisseur.num_f),
                        ISIN(f_p.num_p, piece.num_p);
END INTERFACE;
% ------------------------------------------------------------------
% Fin de l'exemple de la définition d'une BD
% ------------------------------------------------------------------
```

```
% ------------------------------------------------------------
% Exemple de l'utilisation d'une BD
% ------------------------------------------------------------
PROGRAM application_gerer_piece;
        USE database_standard;
        USE definition_gerer_piece;
        RELATION rel_1, rel_2 OF *;
        BEGIN
%
% --- Obtenir les noms de différents fournisseurs qui fournissent
% --- la pièce numéro 5.
%
                rel_1 := SELECT (f_p, num_p=5);
                rel_2 := PROJECT (JOIN (fournisseur, rel_1,
                                        founisseur.num_f=rel_1.num_f),
                                  <num_f>);
%
% --- Imprimer le résultat
%
                print_relation (rel_2);
        END;
END PROGRAM;
% ------------------------------------------------------------
% Fin des exemples
% ------------------------------------------------------------
```

[Note] La fonction "print_relation" est supposée être définie comme fonction utilitaire pour imprimer une relation dans le module "database_standard".

5. Réalisation du préprocesseur R-LTR

Les programmes écrits en R-LTR contiennent des instructions de LTR-3 ainsi que celles concernant les applications de BD relationnelle. Ces dernières ne pourront pas être traitées par le compilateur LTR-3, et on a donc besoin de les transformer en des instructions LTR-3 de manière à ce que le programme résultant puisse être compilé par le compilateur LTR-3. Le principe est d'utiliser un préprocesseur pour traduire une application écrite en R-LTR en un programme LTR-3 contenant des appels au système support. Voir la Fig.II-3.



(Fig.II-3) Traitement d'un programme R-LTR

Le système support est chargé de réaliser les fonctionalités d'un SGBD relationnel. Par exemple:

- structurer des données,
- stocker des données,
- exécuter les opérations de mise à jour,

- vérifier des contraintes d'intégrité,
- évaluer des expressions relationnelles.

Les études sur l'évaluation d'une expression relationnelle, notamment la décomposition et l'optimisation d'une expression relationnelle sont faites dans [GEND83]. Pour développer un préprocesseur, nous avons supposé un système support qui réalise toutes les fonctionalités mentionnées ci-dessus.

Comme nous disposons d'un ensemble de produits logiciels existants, particulièrement le système GALL (Générateur d'Analyseur LL(1)) [SYSECA83], nous avons décidé de les utiliser. Suivant la technique utilisée par le système GALL, la traduction est dirigée par la syntaxe à l'aide des actions sémantiques associées à la grammaire.

La réalisation du préprocesseur R-LTR se subdivise de la manière suivante:

- la définition d'une grammaire LL(1) intégrant le langage R-LTR;
- l'analyseur lexicographique capable de reconnaître les unités lexicales du langage R-LTR;
- l'analyseur syntaxique (ou "parser") capable de déterminer si le texte source en R-LTR est syntaxiquement correct et d'engendrer des instructions LTR-3 à partir des instructions R-LTR;
- l'ensemble de programmes et de structures de données utilisés par ces analyseurs. Par exemple, la table des symboles, le module de saisie du texte.

Le préprocesseur est expérimentalement implanté sur la machine MICROMEGA sous le système d'exploitation SOL en utilisant le langage PASCAL-SOL [SYSECA84]. Les détails de l'implantation sont présentés dans [CRUZLARA84]. Remarquons que le préprocesseur analyse non seulement le texte R-LTR mais aussi celui LTR-3; ce qui rend aux utilisateurs une transparence de l'intégration du langage de BD et une sûreté syntaxique avant de passer à la phase de compilation LTR-3. Les

fonctions prédéfinies de calcul ne sont pas mises en oeuvre par le préprocesseur actuel.

Il faut noter que le préprocesseur n'est qu'une partie de la réalisation du projet R-LTR; il reste encore à achever la partie concernant le système support, soit en développant un tel système ou soit en réalisant un interfaçage avec un SGBD existant.

6. Conclusion

Comme nous l'avions évoqué dans l'introduction, il y a trois approches différentes pour utiliser les langages de programmation comme langages de BD:

- définition de nouveaux langages autonomes destinés à la fois aux applications classiques et aux applications BD;
- incorporation d'un langage de BD existant dans des langages de programmation;
- extension d'un langage de programmation par des concepts de langage de BD.

La première approche conduit à un environnement de programmation généralement plus cohérent parce que les langages proposés sont conçus dès le départ avec un accent mis sur les traits linguistiques nécessaires au développement des applications de BD. Par contre, cette approche demande un effort important de réalisation, au niveau du compilateur et des dispositifs spéciaux relatifs à la gestion de BD pour l'exécution.

La deuxième approche est souvent adoptée, dans les systèmes commercialisés, par son intérêt de pouvoir directement se connecter à un SGBD puissant à partir un langage hôte [GARDARIN83]. Malheureusement cette approche se limite au niveau de communication entre le langage hôte et le langage de BD. Pour cette raison, il est souvent nécessaire de définir des variables globales précédées par des

symboles spéciaux pour les distinguer des autres variables du langage (ex. "##" pour EQUEL, ":" pour SQL). Ces varialbles servent à communiquer les résultats des requêtes.

L'approche que nous avons suivie s'apparente à la troisième parce que la première oblige à changer de langage pour les applications orientées BD et que la deuxième pose un problème de compatibilité entre les objets du langage de programmation et ceux du langage de BD. Par ailleurs, notre étude s'est déroulée sous les trois conditions précises suivantes:

- choix du langage de programmation LTR-3 pour son extension;
- nous ne touchons pas le compilateur LTR-3 pour la réalisation;
- nous ne fixons pas a priori un SGBD particulier pour le système support.

Cela nous a conduit à réaliser le préprocesseur de R-LTR, différemment de PASCAL Relationnel qui est réalisé par la modification du compilateur.

Notre approche a cependant quelques inconvenients pratiques: malgré le traitement entier du langage R-LTR (la partie LTR-3 et celle étendu), notre approche (plutôt notre réalisation actuelle) est encore limitée par le fait que le préprocesseur n'effectue ni vérification de types, ni protection d'objets par la même mesure prise par les traits existants. Sur le plan de la forme, on peut trouver les déclarations assez lourdes et qu'une expression relationnelle, comme requête sur la base, est moins facile à comprendre que l'utilisation d'un langage basé sur le calcul relationnel (ex. QUEL).

Il faut rappeler une autre approche possible pour l'extension d'un langage de programmation en vue des applications de BD relationnelle. Il s'agit d'un langage fournissant un mécanisme de généricité. Dans ce cas, on peut définir un type générique "relation" avec un ensemble d'opérations (ex. celles de l'algèbre relationnelle), qui permet de construire des applications. Les travaux de notre équipe, TYP-R [TOUNSI82] et ADONIS [NASSIF83], se situent dans ce cadre.

CHAPITRE III - LES ETUDES SUR LE MODELE ENTITE-ASSOCIATION

Résumé

Le modèle Entité-Association (E-A) est un des modèles sémantiques largement utilisé pour la conception des bases de données. Dans ce chapitre, nous nous proposons de redéfinir la notion de cardinalité dans le but d'éliminer certaines ambiguités d'interprétation d'un diagramme E-A, qui concerne les dépendances fonctionnelles entre entités de différents types d'entité dans le contexte d'un type d'association. Ainsi un fait réel bien précis peut être correctement représenté par un diagramme E-A, d'où un sensible rapprochement entre concepteur et utilisateur. Une démarche pour la normalisation d'un diagramme E-A basée sur cette nouvelle interprétation est examinée. Nous avons aussi l'intention d'intégrer les notions d'abstraction de données telles que classification, généralisation, agrégation et partition dans le modèle E-A en proposant des types génériques d'association.

Sommaire

1. Introduction
2. Concepts et notions
   2.1. Modèle Entité-Association
   2.2. Diagramme Entité-Association
3. Elimination de certaines ambiguités sur les propriétés de cardinalité
   3.1. Cardinalité d'un type d'entité dans un type d'association
      3.1.1. Fonctionalité
      3.1.2. Cardinalité
   3.2. Dépendance entre entités
      3.2.1. Dépendance d'existence et dépendance d'identification
      3.3.2. Dépendance fonctionnelle entre entités
   3.3. Interprétation d'un type d'association n-aire par dépendance fonctionnelle entre entités
   3.4. Comparaison
4. Un guide d'application du modèle Entité-Association
   4.1. Le premier pas vers la normalisation
      4.1.1. Type d'entité bien défini
      4.1.2. Type d'association bien défini
      4.1.3. Diagramme Entité-Association bien défini
   4.2. Règles de mise à jour des instances
5. Quelques types génériques d'association basés sur les notions d'abstraction de données.
   5.1. Classification - Instanciation
   5.2. Généralisation - Spécialisation
   5.3. Agrégation - Décomposition
   5.4. Regroupement - Partition
   5.5. Remarques concluantes
6. Conclusion

CHAPITRE III
LES ETUDES SUR LE MODELE ENTITE-ASSOCIATION

1. Introduction

D'après ANSI/SPARC [ANSI75], le terme "schéma conceptuel" désigne une description abstraite du monde réel (l'univers de discours) selon le modèle de données d'une Base de Données (BD). Ce schéma décrit une manière de structurer, en vue de les manipuler, des informations dans une BD. Un système de gestion de bases de données fournit un langage de BD pour spécifier le schéma conceptuel et aussi manipuler des informations décrites par le schéma.

D'ailleurs, il est généralement admis, qu'une étape de conception précéde celle de programmation dans le cycle de vie d'un logiciel (analyse des besoins, conception, programmation, test et intégration, maintenance) [BAUER80] [BOEHM79]. Dans l'approche BD, il est aussi généralement accepté qu'une étape conceptuelle plus abstraite précéde celle de construction de la structure de données (schéma conceptuel dans le sens de ANSI/SPARC) représentées dans une BD.

Pour cette raison, de nombreux modèles conceptuels (au sens général) de données sont proposés pour décrire le monde réel indépendamment de l'implantation dans un SGBD. Rappelons qu'un schéma de données est le résultat de la description d'un certain univers à l'aide d'un modèle de données. Pour introduire cette étape de conception, les termes "schéma d'entreprise" [CHEN77] [TSICHRITZIS82], "schéma de conception" [KING84] [ROLLAND76], "schéma descriptif" [FOUCAUT82], "modélisation de vue" [YAO82] ou encore "schéma conceptuel" [MIRANDA84] sont utilisés†. Nous choisissons le terme "schéma (modèle) conceptuel" dans le sens "schéma (modèle) de conception".

(†) A noter que pour ne pas confondre ce terme avec la notion de ANSI/SPARC, [BRODIE84-a] [MIRANDA84] utilisent "schéma logique global" à la place de schéma conceptuel au sens de ANSI/SPARC.

Les modèles conceptuels nous permettent de décrire le monde réel d'une manière non seulement plus aisée mais aussi plus fidèle sémantiquement que les modèles classiques (modèles hiérarchique, réseau, relationnel) parce que l'idée de départ est non pas de mieux organiser une BD, mais plutôt de mieux décrire des problèmes qui sont parfois subtils. D'ailleurs, avec ces modèles conceptuels, il est relativement aisé d'appréhender plus d'aspects sémantiques qu'il n'est possible avec les modèles classiques. C'est pour cela qu'on regroupe ces modèles conceptuels sous le nom de "modèle sémantique" [BRODIE84-a].

Le modèle Entité-Association (noté E-A) [CHEN76], proche d'une technique de représentation de connaissances: réseau sémantique dans le domaine de l'intelligence artificielle, est un des modèles largement utilisé comme modèle conceptuel pour la conception des BDs. La différence entre le modèle E-A et le réseau sémantique est étudiée dans [SOWA84,pp.302-310]. D'une manière générale, la différence peut être mesurée par le taux de descripteur de données (type de données) par rapport aux données individuelles (instances). Dans l'approche BD, un type de données peut avoir de nombreuse répétitions du fait de ses instances. Même si une BD devient plus complexe, le nombre des types reste beaucoup plus petit que le nombre des instances de chaque type. Au contraire, dans l'approche intelligence artificielle, il y a de si petite répétition que chaque donnée peut avoir son propre type (en fait, la plupart des systèmes d'intelligence artificielle ne distinguent pas les données des types de données).

Comme la plupart des modèles sémantiques de données, le modèle E-A n'est pas basé sur un formalisme rigoureux (à la différence du modèle relationnel). L'absence du formalisme rigoureux donne une grande souplesse d'expression d'une part, mais d'autre part, le modèle peut comporter quelques ambiguités inhérentes, gênantes au moment de son application.

Par ailleurs, un des avantages du modèle E-A est sa représentation d'un schéma par un diagramme. L'utilisation d'un graphique donne un grand avantage à la communication entre les hommes,

qui peuvent être des concepteurs de systèmes aussi bien que des utilisateurs finals, puisqu'il est relativement aisé à comprendre par rapport à une longue liste d'expressions mathématiques [OLLE85].

La propriété de cardinalité du modèle E-A est très importante, car elle permet de décrire des dépendances fonctionnelles entre entités de différents types dans le contexte d'un type d'association, pour refléter des contraintes structurelles sur les données représentées par un schéma.

Dans ce chapitre, nous avons l'intention d'approfondir la notion de cardinalité afin d'éliminer certaines ambiguités, en interprétant des schémas en terme de dépendances fonctionnelles entre entités. Ainsi un fait réel peut être représenté par un diagramme E-A sans equivoque. Avec la nouvelle interprétation, nous avons aussi la possibilité d'exprimer diverses sortes de dépendances fonctionnelles pouvant exister entre n types d'entités.

Dans la section 2, nous rappelons le modèle Entité-Association avec des exemples simples qui seront utilisés dans la suite. La section 3 est consacrée au développement des notions de cardinalité et de dépendances fonctionnelles entre types d'entités, qui sont à la base de la nouvelle interprétation des diagrammes E-A. Dans la section 4, nous proposons le premier pas vers une normalisation de ces derniers.

L'abstraction de données est particulièrement utile dans la modélisation. Elle peut prendre des formes diverses. Parmi elles, on peut citer la classification - instanciation, la généralisation - spécialisation, l'agrégation - décomposition et le regroupement - la partition. Dans la section 5, nous étudions ces quatre mécanismes dans le cadre du modèle E-A et nous proposons des types génériques d'association.

2. Concepts et notions

Dans cette section, nous allons présenter le modèle Entité-Association et ses caractéristiques ainsi que sa représentation par un diagramme.

2.1. Modèle Entité-Association

Le **modèle Entité-Association** [CHEN76,77,82] est un modèle de données utilisé lors de l'étape conceptuelle, et qui s'appuie sur trois concepts de base pour décrire le monde réel: entités, associations et attributs.

Entités et Type d'entité

Une **entité** est un individu ou toute chose (concret ou abstrait) qui existe dans un certain univers et qui est perceptible par rapport à une sémantique. Cela suppose qu'une entité peut être identifiée et décrite par rapport à cette sémantique. Dans le modèle, cette description consiste en un ensemble de valeurs représentant des propriétés caractéristiques.

(Exemple)

Chez un concessionaire de voiture, le client appelé "Dupont" et la voiture immatriculée "4523ST54" sont des entités. Le client "Dupont" peut être caractérisé par un ensemble de valeurs des propriétés: Numéro de Sécurité Sociale, Nom, Date de naissance, Adresse, etc.. Pour la voiture, "Renault", "grise", "4523ST54" et "1100" sont des valeurs qui décrivent un véhicule selon les propriétés: Marque, Couleur, Matricule et Cylindrée.

Un **Type d'Entité** (noté TE) est un ensemble d'entités caractérisées par les mêmes propriétés. Par exemple, les TEs CLIENT et VOITURE.

Comme on peut le constater, la notion d'entité n'est pas rigoureusement définie. Pourtant il nous fournira la souplesse de

pensée pour appréhender le sens du monde réel, qui est généralement compliqué et difficile à saisir.

Associations et Type d'association

Une **association** est définie entre deux ou plusieurs entités qui ont un certain lien entre elles. Ce lien représente quelque fait ou prédicat significatif sur ces entités. Un **Type d'Association** (noté TA) n-aire est un ensemble d'associations définies entre n types d'entité $E_1,E_2,\ldots,E_n$, qui vérifient le même fait.

(Exemple)

Un couple (client "c", voiture "v") est une association traduisant le fait que "p" est le propiétaire de "v". On peut définir un TA PROPRIETAIRE entre les TEs CLIENT et VOITURE.

Plus formellement, un TA est défini comme une relation R, c'est à dire un sous-ensemble du produit cartésien $E_1xE_2x\ldots xE_n$ ($n\geq 2$) entre des TEs $E_1,E_2,\ldots,E_n$, non nécessairement distincts. Un n-uplet $(e_1,e_2,\ldots,e_n)\in R$, où $e_i\in E_i$, si et seulement si R est vérifiée entre $e_1,e_2,\ldots,e_n$.

On appelle **rôle** d'un TE dans un TA, la fonction que remplit ce TE dans le TA. L'ordre des TEs composants dans un TA peut être ignoré si le rôle de chacun d'eux est explicité.

Le **degré** d'un TA est défini comme le nombre des TEs participants à ce TA. Si un TA est défini entre deux, trois ou n TEs, il est dit un TA binaire, ternaire ou n-naire.

Attribut et Clé d'un type d'entité

Une propiété d'un TE ou TA est représentée par un couple: **attribut** et **domaine de valeurs**. "grise", "Dupont", "1100",... sont des valeurs de propriétés. On appelle attribut, le nom d'une propriété d'un TE ou TA. Par exemple, COULEUR est un attribut du TE VOITURE et DATE_ACHAT est un attribut du TA PROPRIETAIRE. Formellement, un

attribut est défini comme une fonction monovaluée d'un TE ou TA vers un domaine.

Une **clé d'un type d'entité**† est un attribut ou un groupe d'attributs dont les valeurs peuvent identifier, de façon unique, une entité dans un TE. Il peut y avoir plus d'une clé dans un TE. Dans ce cas, on laissera à l'utilisateur le choix d'une clé sémantiquement plus significative; ça sera la clé principale. En général, une clé d'un type d'association est définie comme l'union des clés principales des TEs participants. [CHEN76,77]

Un TE est noté par l'expression $E(A_1/D_1,A_2/D_2,\ldots,A_m/D_m)$ où E est son nom et A_i/D_i, i=1,2,...,m sont des couples: attribut et domaine de valeurs. Une entité e du type E est représentée par un tuple de valeurs $<a_1,a_2,\ldots,a_m>$ où $a_i \in D_i$, i=1,2,...,m. Pour l'abréviation de l'expression d'un TE, nous utiliserons $E(\underline{A_1},\underline{A_2},\ldots,\underline{A_k},A_{k+1},\ldots,A_m)$ pour désigner un TE E dont la clé est soulignée. Egalement $R(R_1:E_1,R_2:E_2,\ldots,R_n:E_n,A_1/D_1,A_2,/D_2\ldots,A_p/D_p)$ désigne un TA défini entre les TEs $E_1,E_2,\ldots,E_n$ avec les attributs $A_1,A_2,\ldots,A_p$ ($p \geq 0$). R_i précise le rôle de chaque E_i au sein du TA. Si dans un TA tous les TEs ont des noms distincts, nous confondrons ce nom avec le rôle. Nous ferons cette hypothèse dans tout ce qui suit. Ainsi, pour l'expression d'un TA, on utilisera $R(E_1,E_2,\ldots,E_n,A_1,A_2,\ldots,A_p)$. Une association r de type R est représentée par un tuple $<\underline{e_1},\underline{e_2},\ldots,\underline{e_n},a_1,\ldots,a_p>$ où e_i, i=1,2,...,n sont les valeurs de la clé de E_i et $a_j \in D_j$, j=1,2,...,p sont les valeurs des attributs du TA R.

Autres concepts

Un type d'entité est dit **obligatoire** au sein d'un TA si toute entité du type participe à l'association, sinon il est dit **optionnel**.‡ La signification de la notion d'obligation/option est intéressante dans le sens où on peut adjoindre une entité dans un TE optionnel sans

(†) cf. "identifiant" d'un TE [BODART83].
(‡) cf. obligatoire/non-obligatoire [HOWE82], total/partiel [EL-MASRI80] [SCHEUERMANN80], impératif/optionnel [PALMER82].

se préoccuper de son association avec les autres entités.

Une caractéristique majeure d'un TA est déterminée par la **cardinalité** de chaque TE participant indiquée par deux chiffres représentatifs "M" (de l'anglais "Many") et "1". La propriété de cardinalité d'un TA pose une restriction sur le nombre d'entités d'un TE participant qui peuvent être liées à une entité de l'autre TE (le cas binaire) ou à un (n-1)-uplet d'entités des autres TEs (le cas n-aire, n>2). Cette caractéristique joue un rôle important, de façon précise et simple, dans l'expression de certaines règles structurelles relatives au monde modélisé que nous verrons en détail à la section 3.

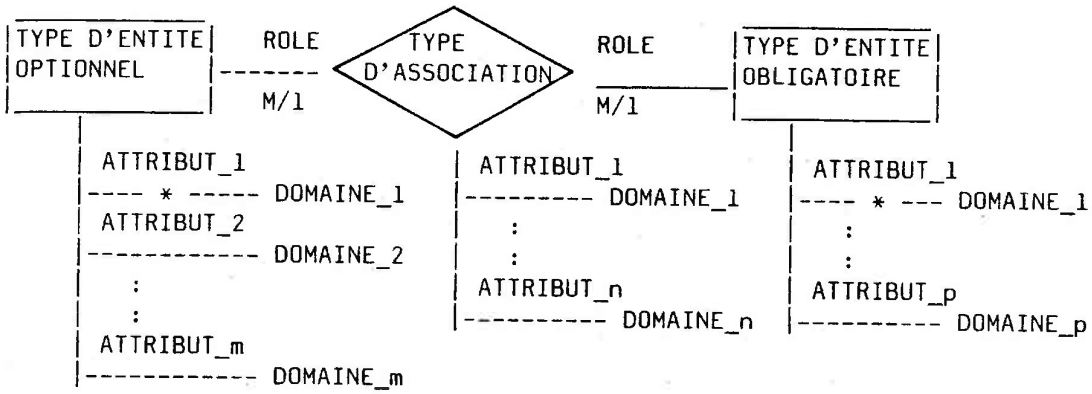
2.2. Diagramme Entité-Association

Un schéma Entité-Association est le résultat d'une application particulière du modèle E-A à un certain univers. Un **diagramme Entité-Association** est une représentation graphique du schéma E-A.

Dans un diagramme E-A, chaque TE et chaque TA sont respectivement représentés par un rectangle et un losange. Le rôle d'un TE participant au TA est explicité sur un arc valué par sa cardinalité (voir §3.1.2.), liant un rectangle et un losange. Rappelons que la précision du rôle d'un TE dans un TA a pour but de se débarrasser de l'ordre des TEs. Ici nous suggérons un ordre ad hoc de gauche à droite et de haut en bas dans un diagramme E-A pour l'interprétation. S'il y a une ambiguité avec cet ordre, nous précisons des rôles pour lever l'ambiguité.

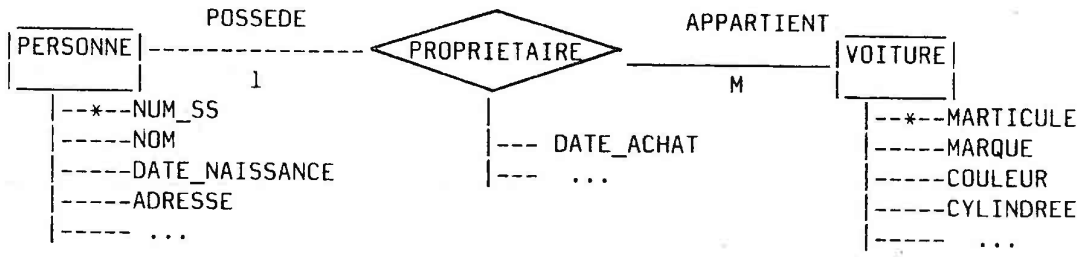
Une liste de couples d'attribut/domaine est énumérée le long d'un TE ou d'un TA. Un attribut ou un groupe d'attributs qui constitue la clé principale d'un TE est marqué par "*" dans le diagramme.

Un TE obligatoire est distingué par un arc en ligne continue liant le TE et un TA, différent d'un arc en tirets indiquant un TE optionnel. Une présentation d'un diagramme E-A se trouve dans la Fig.III-1.



(Fig.III-1) Représentation graphique du modèle E-A par un diagramme

Par convention et par simplicité, quand un attribut et son domaine de valeurs portent le même nom, le domaine ne sera plus représenté dans un diagramme E-A (voir la Fig.III-2). Dans ce qui suit, on utilisera des lettres majuscules pour représenter des types (TE, TA ou attribut), et des lettres minuscules pour représenter les instances d'un type (entités, associations ou valeurs d'un attribut).



(Fig.III-2) Exemple d'un diagramme E-A

3. Elimination de certaines ambiguités sur les propriétés de cardinalité

Dans cette section nous allons étudier plus précisément les caractéristiques du modèle E-A dans le but d'éliminer certaines ambiguités en évitant de tomber dans un piége de complication.

3.1. Cardinalité d'un type d'entité dans un type d'association

Une partie centrale de la conception d'un schéma E-A est l'idée de dépendance entre données, qui constitue une contrainte sur les instances possibles d'un schéma. Avant de développer certaines dépendances en détail dans la section suivante, nous définissons la cardinalité d'un TE dans un TA à l'aide de la notion de fonctionnalité.

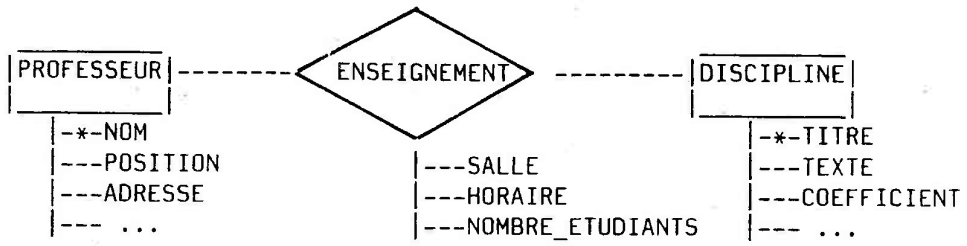
3.1.1. Fonctionnalité

La notion de **fonctionnalité** est un moyen de classifier des TAs selon le nombre d'entités (une ou plusieurs) d'un TE E_1 qui peuvent être associées à une entité d'un TE E_2, dans un TA défini entre E_1 et E_2, à un instant donné [ULLMAN82,p.15].

Un TA (binaire) est du type **Many-to-One, One-to-One ou Many-to-Many** (notés respectivement M-1, 1-1 ou M-M). Soit A un TA entre E_1 et E_2, A est dit Many-to-One entre E_1 et E_2, dans cet ordre, si et seulement si au plus une entité de E_2 peut être associée à une entité de E_1 (autrement dit A a la caractéristique d'une fonction de E_1 vers E_2). Il est dit One-to-One si et seulement si pour une entité dans un des TEs, il y a au plus une entité associée par A de l'autre TE. Enfin A est dit Many-to-Many si et seulement si aucune des règles précédentes n'est vérifiée.

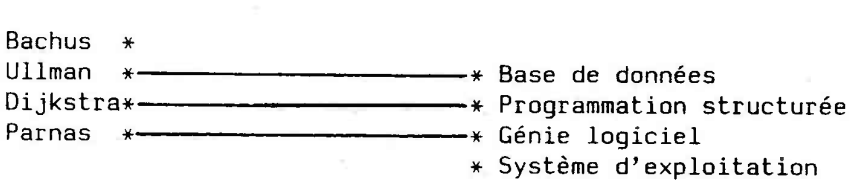
(Exemple 1)

Le diagramme E-A suivant donne un exemple pour deux TEs PROFESSEUR, DISCIPLINE et un TA ENSEIGNEMENT entre ces deux TEs.



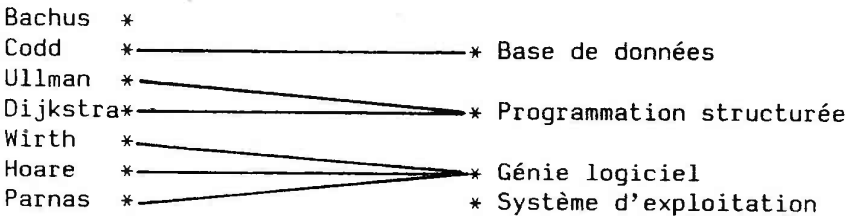
(1) ENSEIGNEMENT est du type 1-1 si on a les deux règles:

"Un professeur ne peut enseigner qu'une discipline au plus."
"Une discipline ne peut être enseignée que par un professeur au plus."



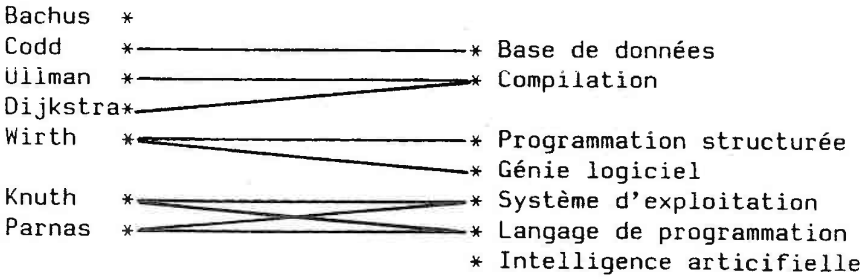
(2) ENSEIGNEMENT est du type M-1 si on a les règles:

"Un enseignant ne peut enseigner qu'une discipline au plus."
"Une discipline peut être enseignée par plusieurs professeurs."



(3) ENSEIGNEMENT est du type M-M si on a les règles:

"Un professeur peut enseigner plusieurs disciplines."
"Une discipline peut être enseignée par plusieurs professeurs."



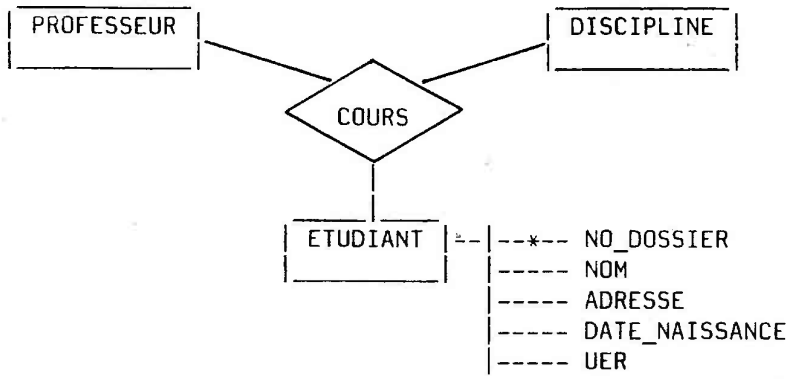
Dans l'exemple (2) ci-dessus on peut remarquer que la première règle énonce que l'association est fonctionnelle entre PROFESSEUR et DISCIPLINE dans cet ordre, et que cette fonctionnalité n'est pas réciproque, ce qui est formulé par la seconde règle. Dans l'exemple (1) les deux règles sont fonctionnelles, et dans (3) aucune ne l'est. En fait, les règles non fonctionnelles peuvent être supposées implicites.

A noter que certaines entités peuvent ne pas participer à l'association avec d'autres. Dans l'exemple (1) ci-dessus, l'entité "Bachus" du TE PROFESSEUR n'a d'association avec aucune entité du TE DISCIPLINE. La règle du genre "tout professeur doit enseigner" est traduite par la notion d'obligation/option d'un TE dans un TA. Un TE est supposé optionnel par défaut.

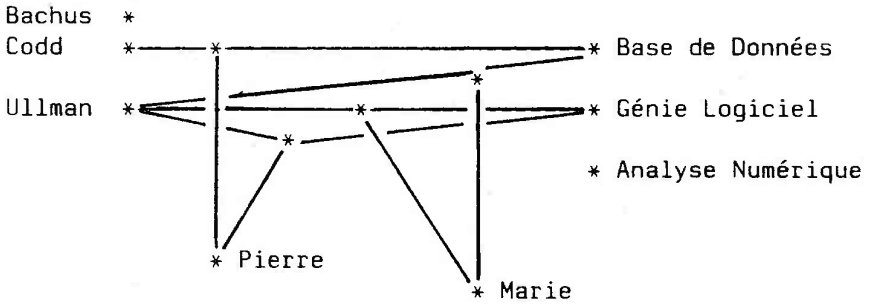
L'extension du concept de fonctionnalité aux TAs n-aires (n>2) se fait comme suit: soit A un TA défini entre $E_1,E_2,\ldots,E_n$. Isolons un E_i. A peut être considéré comme une association binaire entre $E_1,E_2,\ldots,E_{i-1},E_{i+1},\ldots,E_n$ d'une part, et E_i d'autre part. Le concept de fonctionnalité s'applique alors à A considéré comme binaire dans le sens ci-dessus. Cette fonctionnalité peut être représentée par les cardinalités associées aux TEs participants dans un TA. Nous verrons, dans le paragraphe suivant, comment symboliser dans un diagramme E-A la fonctionnalité, ce qui permettra à un concepteur d'envisager toute règle de ce type lors de la conception de son schéma E-A.

(Exemple 2)

Le diagramme suivant représente un programme de cours dans un établissement. C'est une extension de l'Exemple 1 au TE ETUDIANT.



Si nous voulions imposer le fait que dans un cours il y a un seul professeur qui donne le cours, cela se traduirait par les occurrences suivantes:



Nous verrons, dans le paragraphe suivant, comment symboliser dans un diagramme E-A, la fonctionnalité d'un TA, ce qui permettra au concepteur d'envisager toute règle de ce type lors de la conception de son schéma.

3.1.2. Cardinalité

La notion de cardinalité dérive de celle de fonctionnalité. La **cardinalité** d'un TE participant dans le contexte d'un TA est défini comme suit:

(1) pour un TA binaire: le nombre maximum de différentes entités d'un TE qui peuvent être liées à une entité de l'autre TE dans le contexte d'un TA.

(2) pour un TA n-aire: soit un TA, A défini sur un ensemble de n TEs, $E=\{E_1,E_2,\ldots,E_n\}$. La cardinalité d'un TE, E_i par rapport aux autres TEs, $E_1,E_2,\ldots,E_{i-1},E_{i+1},\ldots,E_n$ (noté $E-E_i$) dans le contexte de A est définie comme le nombre maximum de différentes entités de E_i qui peuvent être liées à une instance de (n-1)-uplet de $E-E_i$.

La cardinalité d'un TE au sein d'un TA est **1** si et seulement si le nombre défini ci-dessus est 1, sinon **M.** Pour les cardinalités nous prenons deux valeurs représentatives telles que "M" et "1". La valeur "M" désigne une valeur supérieure à 1.†

Dans le cas des professeurs et des disciplines de l'Exemple 1, la cardinalité de PROFESSEUR est M si plusieurs professeurs peuvent enseigner la même discipline. Par contre, si une discipline ne peut être enseignée par un seul professeur au plus, la cardinalité de PROFESSEUR est 1. Dans le diagramme E-A, cette cardinalité est symbolisée par "M" ou "1" sur le trait reliant un TE à un TA, comme indiqué dans l'Exemple 3.

(Exemple 3)

Les exemples donnés dans l'Exemple 1 peuvent être présentés par les cardinalités suivantes:

(1)

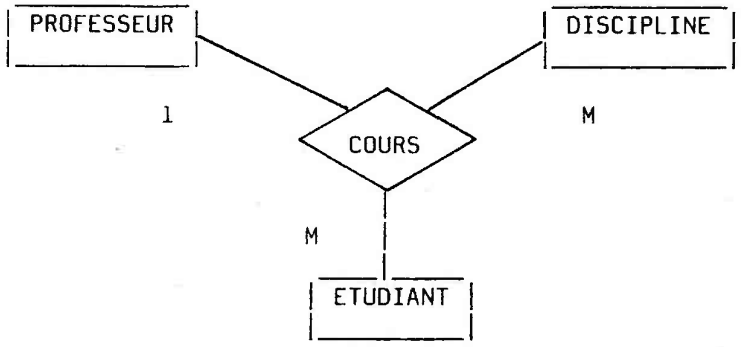
(2)

(3)

(†) Originalement [CHEM76] propose d'utiliser les lettres M, N, P, ... pour exprimer un TA de type M-M, M-M-M, ... Nous utilisons simplement une lettre représentative "M" de l'anglais "Many".

L'exemple donné dans l'Exemple 2 est représenté par les cardinalités suivantes:

(4)



A noter que le diagramme (4) représente également une information importante: "un étudiant doit suivre au moins un cours", par le fait que le TE ETUDIANT est obligatoire.

3.2. Dépendance entre entités

Comme domaine d'interprétation du modèle E-A, nous étudions certaines dépendances existant entre des types d'entité.

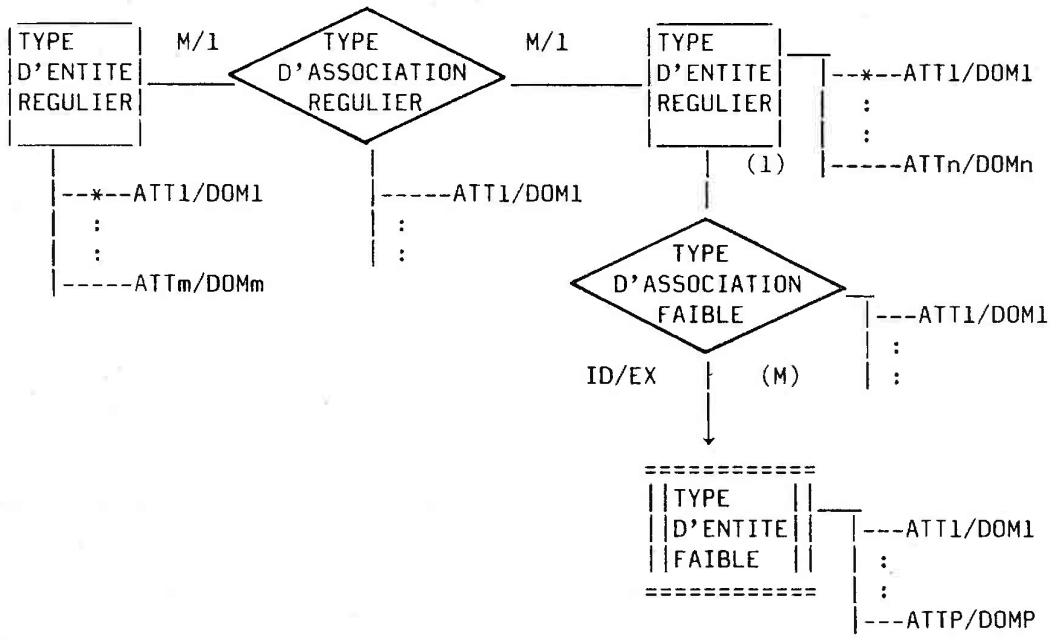
3.2.1. Dépendance d'existence et dépendance d'identification

Si l'existence d'une entité dépend d'une autre entité auquelle elle doit être associée, il est dit qu'il y a une **dépendance d'existence** entre deux types d'entité. Ceux-ci ne sont pas nécessairement distincts. Le type d'entité dépendant est appelé un type d'entité **faible** et le type d'association concerné est appelé un type d'association faible. Un TE qui n'est pas faible sera appelé un TE **régulier** et un TA qui ne concerne aucun TE faible sera appelé un TA régulier. Cette dépendance d'existence impose une contrainte de mise-à-jour telle que si une entité de type régulier est supprimée, toutes les entités de type faible liées à cette entité seront aussi automatiquement supprimées.

En principe, chaque TE a une clé, car nous avons accepté le fait qu'on pouvait identifier une entité. Pourtant il arrive parfois

qu'une entité ne peut pas être identifiée par la (les) valeur(s) de son (ses) attribut(s), mais peut être identifiée par l'association avec une autre entité, on dira qu'il y a une **dépendance d'identification** entre ces deux TEs. Le TE dépendant est aussi un type d'entité faible.

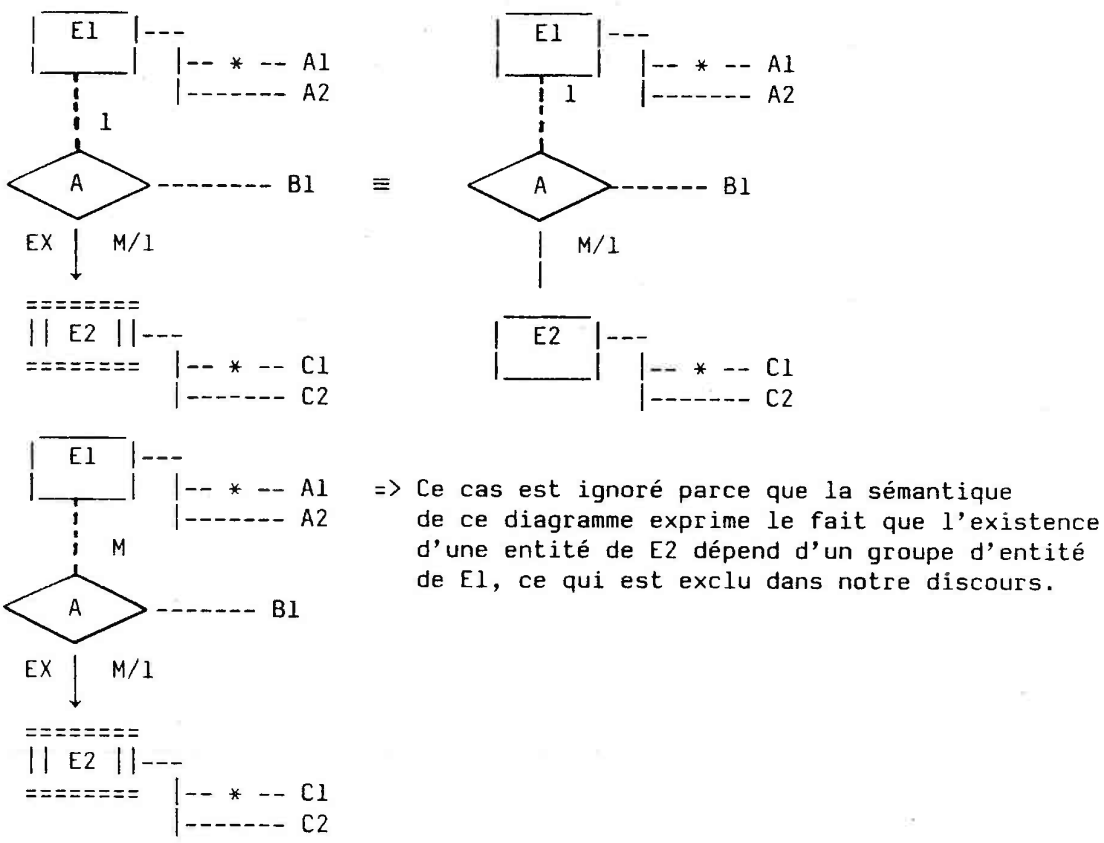
Dans un diagramme E-A originalement proposé par [CHEN76], un TE faible est représenté par un rectangle doublement encadré et est désigné par une flèche venant d'un TA faible. Pour distinguer les deux différentes sortes de TEs faibles, on indique la sorte de dépendance par "ID" pour dépendance d'identification et "EX" pour dépendance d'existance. Voir la Fig.III-3. Ici, nous ne nous intéressons qu'à la dépendance d'existence ou identification dans les TAs binaires.



(Fig.III-3) Diagramme E-A simplifié avec TE et TA faibles.

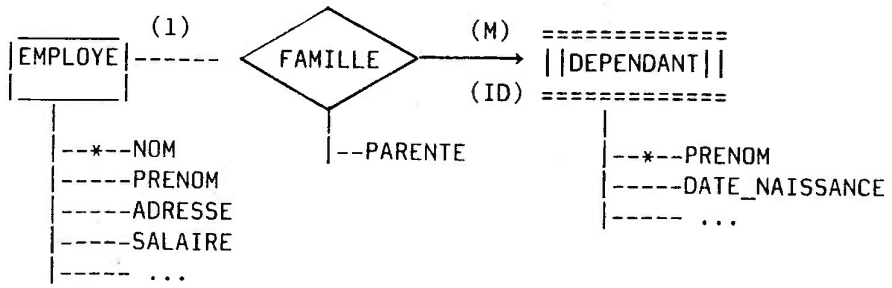
Remarquons premièrement qu'une dépendance d'identification est implicitement incluse dans une dépendance d'existence. Pour la clarté, nous allons exclure la dépendance d'identification de la

définition d'une dépendance d'existence. Deuxièmement, une dépendance d'existence peut être spécifiée par la notion d'obligation, parce que dans un TA binaire, si un TE est obligatoire, toute entité de ce type est strictement liée à une autre entité dans l'association concernée. Autrement dit, toute entité d'un TE obligatoire dépend par existence d'une autre entité dans leur association [KIM85] [LENZERINI83] [SCHEUERMANN80]. Cette représentation est illustrée dans la Fig.III-4.



(Fig.III-4)
Représentation d'un TE faible basé sur une dépendance d'existence

Par conséquent, pour le TE faible, nous ne prenons en considération que celui défini par une dépendance d'identification avec le type d'association binaire de type M-1, illustré comme suit (cf. [CHEN76]):



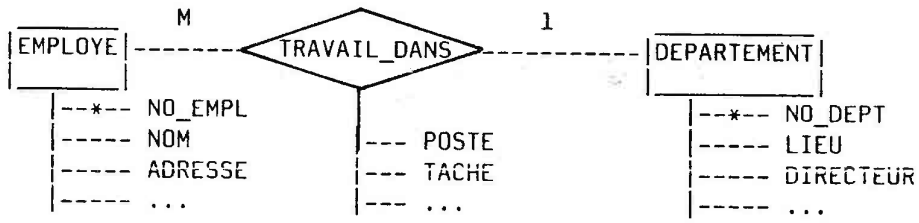
Ce schéma présente un TE faible DEPENDANT lié au TE EMPLOYE avec une dépendance d'identification. La clé de DEPENDANT est l'union de EMPLOYE.NOM et DEPENDANT.PRENOM.

Sous un autre angle, la nécessité d'un TE faible provient du fait qu'un TE peut avoir une propriété (au sens général) non-atomique, c'est à dire, celle composée, multivaluée ou encore composée et multivaluée.

3.2.2. Dépendance fonctionnelle entre entités

On dit qu'il y a une **Dépendance Fonctionnelle entre Entités** (en abrégé DFE) de deux TEs E_1 et E_2, notée $E_1 \rightarrow E_2$, si et seulement si, il existe un TA entre ces deux TEs dans lequel une entité de E_1 est associée à au plus une entité de E_2, à tout moment [KIM86-a] (cf. [WILKIE82]).

(Exemple 4)



Le département où un employé travaille est fonctionnellement déterminé par un employé. C'est-à-dire que chaque employé ne travaille que dans un département.

Cette définition signifie que si un TA (soit A) défini entre E_1 et E_2 est représenté par un diagramme, il est de type M-1 ou 1-1. Autrement dit dans A de type M-1, la connaissance d'une entité de E_1 suffit pour déterminer l'autre entité de E_2. Dans le cas 1-1 on a alors $E_1 \rightarrow E_2$ et $E_2 \rightarrow E_1$ (noté $E_1 \longleftrightarrow E_2$). En conséquence de cette définition, une DFE se juge par rapport à un TA. Reprenons l'Exemple 3, alors en terme de DFE nous avons:

pour (1), PROFESSEUR ⟷ DISCIPLINE
pour (2), PROFESSEUR → DISCIPLINE.

La définition d'une DFE peut être étendue à n TEs $E=\{E_1,E_2,\ldots,E_{i-1},E_i,E_{i+1},\ldots,E_n\}$. Il y a une DFE entre $E_1,E_2,\ldots,E_{i-1},E_{i+1},\ldots,E_n$ (en abrégé $E\text{-}E_i$) et E_i, notée $E\text{-}E_i \rightarrow E_i$, si et seulement si, il existe un TA n-aire A entre ces n TEs dans lequel, quelque soient les associations a_1 et a_2 de A, si $a_1[E\text{-}E_i]=a_2[E\text{-}E_i]$ alors $a_1[E_i]=a_2[E_i]$ où $a_1[E\text{-}E_i]$ est une projection de l'association a_1 sur le sous-ensemble de TEs participants $E\text{-}E_i$. Dans le diagramme (4) de l'Exemple 3, on a la DFE: ETUDIANT DISCIPLINE → PROFESSEUR, vérifiée par les occurrences données dans l'Exemple 2:

(a) **Pierre** suit la discipline **Base de Données** enseignée par Prof. **Codd**
(b) **Pierre** suit la discipline **Génie Logiciel** enseignée par Prof. **Ullman**
(c) **Marie** suit la discipline **Base de Données** enseignée par Prof. **Ullman**
(d) **Marie** suit la discipline **Génie Logiciel** enseignée par Prof. **Ullman**

A noter au passage qu'aucune autre DFE n'est observée:

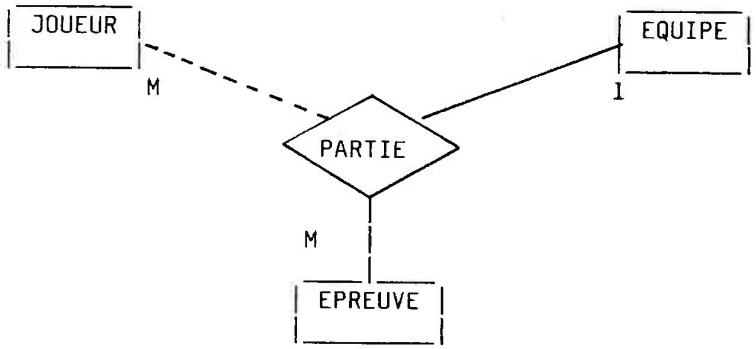
ETUDIANT -/-> DISCIPLINE, PROFESSEUR à cause de (a) et (b);
DISCIPLINE -/-> ETUDIANT, PROFESSEUR à cause de (a) et (c);
ETUDIANT PROFESSEUR -/-> DISCIPLINE à cause de (c) et (d);
DISCIPLINE PROFESSEUR -/-> ETUDIANT à cause de (b) et (d).

Comme nous pouvons le constater dans les exemples, la définition de la cardinalité ne fait que reprendre le concept des DFEs pour symboliser ces dernières par un diagramme E-A. En bref, si un TE porte

la cardinalité "1" au sein d'un TA, ses entités sont fonctionnellement déterminées par celles des TEs restants.

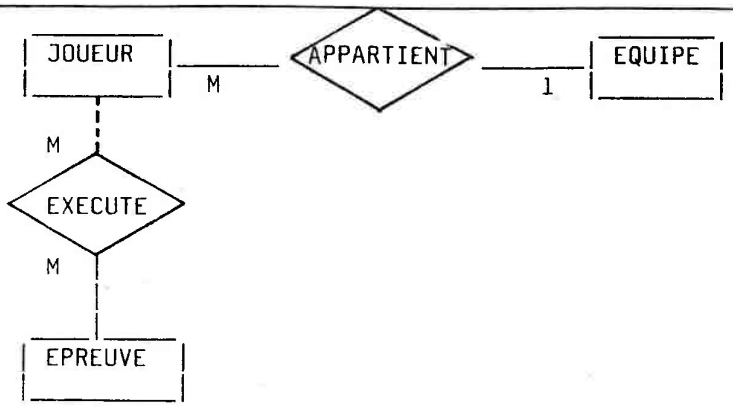
3.3. Interprétation d'un type d'association n-aire par dépendance fonctionnelle entre entités

Considérons le diagramme de la Fig.III-5, sur une partie de gymnastique et où les joueurs de chaque équipe exécutent un certain nombre d'épreuves. Ici, et en vertu de la définition, la cardinalité du TE EQUIPE est 1. Mais, dans le TA PARTIE, la connaissance d'un joueur suffirait pour connaître son équipe. L'épreuve n'apporte aucune information supplémentaire dans ce sens.



(Fig.III-5) Autre exemple de cardinalités M-M-1.

Bien qu'en terme de DFE nous puissions toujours affirmer que JOUEUR EPREUVE → EQUIPE, cette DFE n'est pas significative (elle est en fait triviale), puisqu'en réalité nous avons JOUEUR → EQUIPE. Ainsi les deux diagrammes (le diagramme de la Fig.III-5 et celui de l'Exemple 3, (4)) sont similaires, quant à la cardinalité, mais ils ont des interprétations différentes parce que le diagramme ci-dessus (qui est parfaitement plausible) traduit mal la réalité qu'il modélise. Si dans cet exemple la seule contrainte était JOUEUR → EQUIPE, alors le bon diagramme serait celui de la Fig.III-6.



(Fig.III-6) Diagramme modifié pour la Fig.III-5.

Ce qui nous ramène à préciser la cardinalité 1 comme suit: dans un TA, un TE a la cardinalité 1 s'il dépend fonctionnellement de la totalité des TEs restants.

Ainsi à un diagramme E-A donné avec cardinalité, est associé un ensemble bien déterminé de DFEs, qui est vide si aucune cardinalité n'est égale à 1. Soit un TA défini entre n TEs $E_1,E_2,\ldots,E_n$. Supposons les k ($0 \leq k \leq n$) premiers TEs ayant la cardinalité M et les autres 1. Dans ce cas nous avons uniquement les DFEs suivantes:

$$
\begin{aligned}
E_1E_2\ldots E_kE_{k+2}\ldots E_{n-1}E_n &\rightarrow E_{k+1}\\
E_1E_2\ldots E_{k+1}E_{k+3}\ldots E_{n-1}E_n &\rightarrow E_{k+2}\\
&\vdots\\
&\vdots\\
E_1E_2\ldots E_{n-2}E_{n-1} &\rightarrow E_n.
\end{aligned}
$$

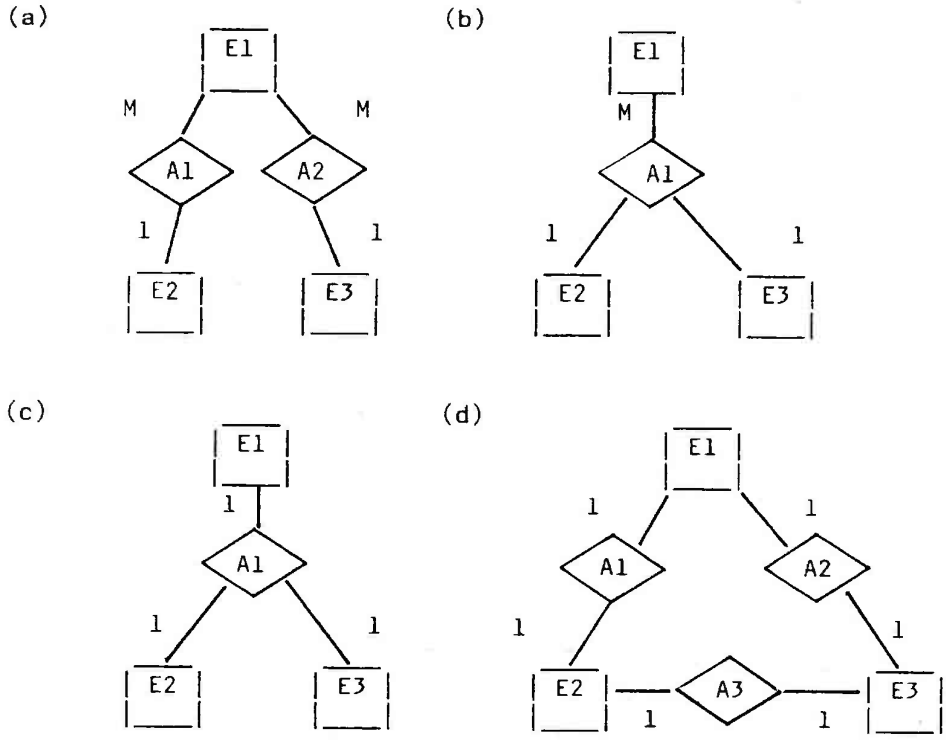
Un diagramme E-A que nous proposons est complet dans le sens où il peut décrire toutes les sortes de DFE possibles parmi un ensemble de n TEs. Par exemple, s'il existe une DFE $E_iE_{i+1},\ldots,E_{i+k} \rightarrow E_j$ pour $k \geq 0$ et $j \notin \{i,i+1,\ldots,i+k\}$, alors cette DFE peut être décrite par la cardinalité "1" de E_j, dans le contexte du TA défini entre $\{E_i,E_{i+1},\ldots,E_{i+k},E_j\}$.

3.4. Comparaison

Dans les sections précédentes nous avons vu comment la cardinalité permettait, dans un diagramme E-A, de représenter des DFEs. Nous avons proposé une définition des cardinalités pouvant s'interpréter de façon exacte par un ensemble de DFEs, et de là un diagramme unique qui peut les représenter.

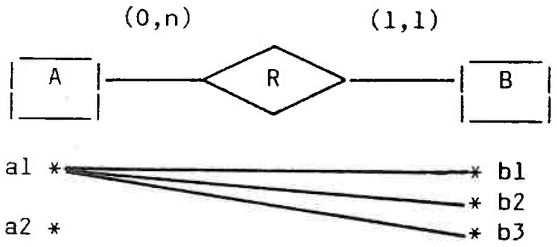
Dans divers travaux [JAJODIA83] [KIM85] [WILKIE83], une interprétation différente de la notion de cardinalité du modèle E-A est proposée que nous résumons ainsi: dans un TA, chaque TE portant la cardinalité 1 est fonctionnellement déterminé par un TE ou un ensemble de TEs qui portent la cardinalité M. D'après leur interprétation le diagramme (a) de la Fig.III-7 est interprété par un ensemble de DFE $\{E_1 \rightarrow E_2, E_1 \rightarrow E_3\}$, ceci est aussi le cas du diagramme (b). Nous considérons cela comme une ambiguité parce que les deux diagrammes E-A différents sont interprétés par le même ensemble de DFEs. Avec notre interprétation (a) est interprété par $\{E_1 \rightarrow E_2,\ E_1 \rightarrow E_3\}$ et (b) par $\{E_1E_2 \rightarrow E_3,\ E_1E_3 \rightarrow E_2\}$.

Ils ont aussi interprété le diagramme (c) par un ensemble de DFEs $\{E_1 \longleftrightarrow E_2,\ E_2 \longleftrightarrow E_3,\ E_3 \longleftrightarrow E_1\}$ et de la même façon le diagramme (d). Avec notre interprétation (c) est interprété par $\{E_1E_2 \rightarrow E_3,\ E_2E_3 \rightarrow E_1,\ E_3E_1 \rightarrow E_2\}$ et (d) par $\{E_1 \longleftrightarrow E_2,\ E_2 \longleftrightarrow E_3,\ E_3 \longleftrightarrow E_1\}$. Un exemple de ce cas (c) est celui où A_1 représente une relation entre trois nombres dont la somme est constante.



(Fig.III-7) Exemples d'ambiguité du diagramme E-A

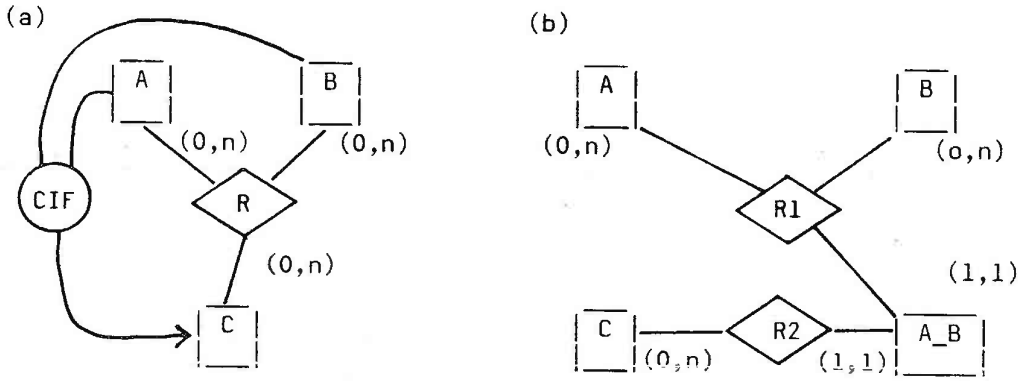
Dans [GALACSI84] et [TARDIEU83,85], la cardinalité a un autre sens issu du modèle d'Abrial [ABRIAL74] (cf. le modèle Z [DELOBEL82]). Ils utilisent la cardinalité minimale et la cardinalité maximale comme étant respectivement, le nombre minimum d'implications d'une entité au sein du TA considéré, et son nombre maximum d'implications au sein du même TA. La cardinalité minimimale représente, en fait, le caractère "optionnel" ou "obligatoire" de la participation des entités d'un TE dans le TA considéré [LENZERINI83] [SCHEUERMANN80]. Leur définition de la cardinalité maximale est aussi nommée "degré de participation maximum" [KENT83]. La notation utilisée dans un diagramme est un couple: (cardinalité minimale, cardinalité maximale). Voir la Fig.III-8.



(Fig.III-8) Représentation d'une association de type 1-M entre A et B.

Dans un cas comme celui-ci, on peut dire que B→A. Mais généralement on ne peut pas, au vu des cardinalités, dire quelles sont les dépendances fonctionnelles (appelées Contraintes d'Intégrité Fonctionnelles) existant entre des TEs.

Soit R un TA défini entre trois TEs A, B et C. Pour représenter AB→C, dans [TARDIEU85] on indique clairement sur le diagramme un arc joignant A, B et C (Fig.III-9, (a)), et dans [GALACSI84] on crée un TE intermédiaire, soit A_B, et on décompose le TA R autour de ce TE A_B (Fig.III-9, (b)).



(Fig.III-9) Représentation de AB→C

Dans [LENZERINI83], les auteurs considérent une cardinalité semblable à la nôtre, mais leur but est de proposer un moyen de la définir pour des combinaisons quelconques de TEs parmi tous ceux qui participent à un TA, offrant ainsi la possibilité de restreindre les associations entre entités dans ces diverses combinaisons. Pour éviter les incohérences pouvant survenir entre ces différentes cardinalités, ils proposent aussi une dizaine de règles à vérifier.

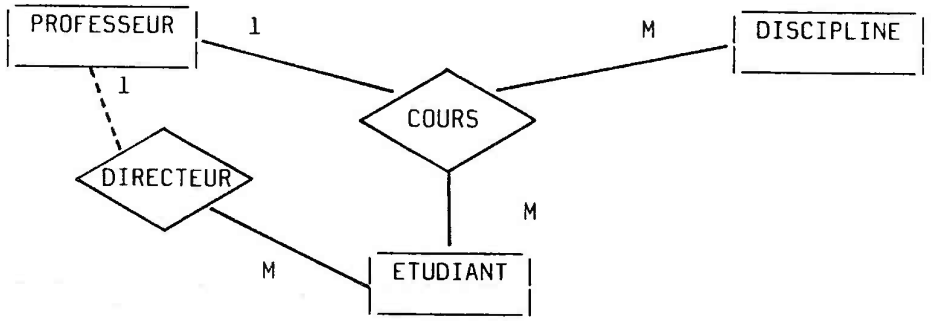
Dans cette section, nous avons mis en évidence certaines propriétés du modèle E-A. Cela ne signifie pas que notre étude garantit de construire d'un schéma E-A bien défini, mais cela permettra au moins aux utilisateurs d'éviter de se méprendre au moment de l'application et l'interprétation d'un schéma E-A. Dans la section suivante, nous nous consacrons à décrire une méthode d'application du modèle E-A pour obtenir un schéma bien défini.

4. Un guide d'application du modèle Entité-Association

L'élaboration d'un schéma conceptuel est une phase cruciale dans le processus du développement d'un système utilisant une grande masse d'informations décrivant un certain univers. Pour l'établir nous posons des disciplines sur l'application du modèle E-A.

[Hypothèse] Chaque TA représente un fait bien précis du monde réel.

Une première intention derrière cette hypothèse, est de ne pas mélanger des liens définis entre les mêmes TEs dans un même contexte; chacun de ces liens pouvant avoir ses contraintes propres. La deuxième intention est de ne pas interpréter automatiquement un TA n-aire défini entre des TEs comme une composition des TAs plus simples, ayant le degré inférieur à n. C'est aussi un raisonnement logique basé sur le fait que dans le modèle E-A, on ne peut pas définir des associations entre associations.† La Fig.III-10 illustre bien plusieurs TAs possibles entre des TEs, qui ne peuvent pas être confondus au moment de l'interprétation.



Les TAs représentés ont des significations différentes. Le TA DIRECTEUR représente une association entre des étudiants de 3ème cycle et leur directeur de recherche, ce qui n'a aucun rapport direct avec le TA COURS.

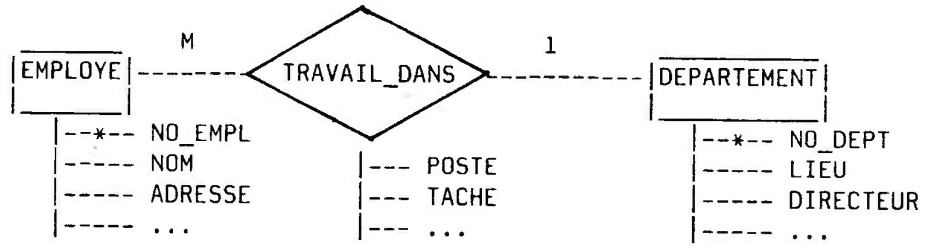
(Fig.III-10) Exemple de TAs définis entre les mêmes TEs.

(†) [SCHEUERMANN80] propose une extension du modèle E-A en proposant un mécanisme permettant de définir une association entre TAs.

4.1. Le premier pas vers la normalisation

Dans un TE ou TA, on dira qu'il y a une **Dépendance Fonctionnelle entre Attributs** (en abrégé DFA), notée A→B , si et seulement si, à tout moment, la connaissance d'une valeur d'un attribut A (ou des valeurs d'un groupe d'attributs) détermine au plus une valeur d'un autre attribut B. (cf. [BODART83,pp.46-50]). Suivant la définition de clé d'un TE, chaque valeur de clé d'un TE correspond au plus une valeur de chaque attribut non-clé. Rappelons aussi que la clé d'un TA est définie comme l'union des clés principales des TEs participants. Dans l'Exemple 5, on trouve un ensemble de DFAs déduites à partir du diagramme E-A.

(Exemple 5)



DFA: NO_EMPL → NOM, ADRESSE,...;
NO_DEPT → LIEU, DIRECTEUR,...;
EMPLOYE.NO_EMPL DEPARTEMENT.NO_DEPT → POSTE, TACHE,...;

DFE: EMPLOYE → DEPARTEMENT

Il faut noter que l'on peut interpréter une DFE par une dépendance fonctionnelle entre les clés des TEs participants dans le contexte du TA concerné. Dans l'Exemple 5, la DFE: EMPLOYE → DEPARTEMENT, déduite à partir des cardinalités, peut être interprétée par la DFA: EMPLOYE.NO_EMPL → DEPARTEMENT.NO_DEPT dans le TA TRAVAIL_DANS. Dans [JAJODIA83], pour exprimer cette idée, l'auteur utilise la notion de "dépendance entre clés des TEs" ("entity-key dependency").

Le concept de dépendance fonctionnelle entre attributs joue un rôle important dans l'élaboration d'un schéma E-A, notamment il est

utilisé pour définir un TE ou TA bien défini dans les paragraphes suivants.

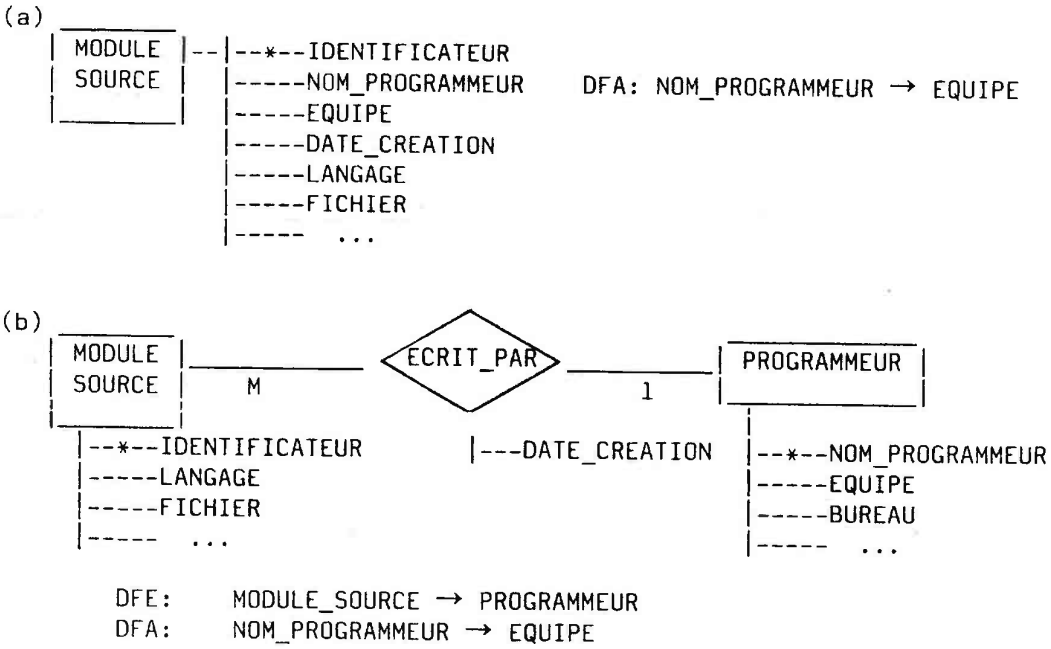
4.1.1. Type d'entité bien défini

Un **type d'entité est dit "bien défini"** s'il n'existe aucune dépendance fonctionnelle entre les attributs non-clés.

Dans le cas contraire, un TE n'est pas bien défini, car il est probable que dans le TE conçu nous ayons inclu une autre entité (nous la considérons comme une entité cachée) qui aurait dû normalement être perçue séparément.

La modification proposée est basée sur l'idée de la décomposition en forme normale d'un schéma de relation tout en gardant toutes les dépendances fonctionnelles et les informations [ULLMAN82,p.242]. En effet, une entité cachée peut être remplacée par une entité dont la cardinalité est "1" avec un nouveau TA qui lie le TE originale au nouveau TE créé. Prenons un exemple.

(Exemple 6)



Ainsi que cela est illustré par le diagramme (a) de l'Exemple 6, on peut concevoir un TE MODULE_SOURCE avec un ensemble d'attributs. Si un programmeur appartient à une seule équipe de travail, on peut le traduire par la dépendance fonctionnelle entre attributs NOM_PROGRAMMEUR→EQUIPE. Cette DFA, que l'on a perçu ultérieurement, amène à introduire le nouveau TE PROGRAMMEUR ayant la clé NOM_PROGRAMMEUR, avec un TA ECRIT_PAR qui lie PROGRAMMEUR à MODULE_SOURCE, ce qui est montré par (b). Après la modification, on constate qu'on peut aussi élaborer le schéma modifié. Par exemple, on ajoute l'attribut BUREAU au TE PROGRAMMEUR, et aussi on déplace l'attribut DATE_CREATION au TA ECRIT_PAR créé, parce que dans ce cas l'attribut DATE_CREATION s'attache mieux à ce TA qu'au TE MODULE_SOURCE.

Il est logique que les TE MODULE_SOURCE et PROGRAMMEUR soient obligatoires pour exprimer le fait que, pour chaque module source, on doit préciser son auteur (un programme est toujours écrit par quelqu'un), et que le nouveau TE PROGRAMMEUR est issu des attributs du MODULE_SOURCE. En principe, les TEs deviendront obligatoires dans le diagramme modifié comme illustré par (b) de l'Exemple 6.

En pratique, on rencontre souvent un problème qui n'est pas facile à résoudre: en effet, décider si un objet issu de l'analyse préalable doit être traité comme un attribut d'une entité ou comme une seconde entité associée à la première [KENT83]. Dans [PALMER82], on trouve certains critères pour distinguer un attribut caractérisant une entité, d'une autre entité associée. Les critères peuvent servir de guide, d'une manière générale: un attribut d'un TE est mieux traité comme un nouveau TE si (i) l'attribut a d'autres attributs dépendants; (ii) le nouveau TE résultant de (i) a une signification dans le monde réel; (iii) l'attribut est la clé du nouveau TE; (iv) le nouveau TE peut être associé à plusieurs entités du premier TE; (v) le nouveau TE peut être associé avec d'autres TEs.

Ces critères sont justifiés dans notre démarche. Cependant le deuxième nous paraît avoir un caractère subjectif dépendant du concepteur.

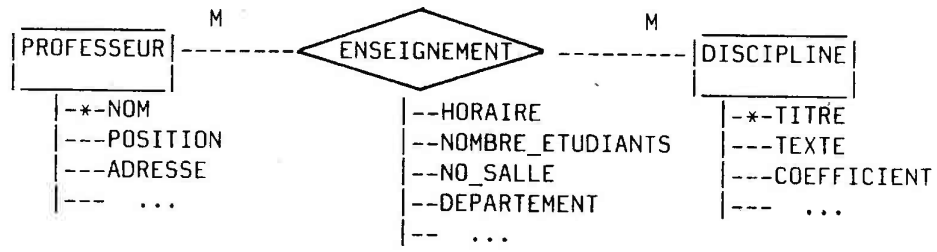
4.1.2. Type d'association bien défini

Un **type d'association est dit "bien défini"** s'il n'existe pas de dépendance fonctionnelle entre ses attributs. Si un TA n'est pas bien défini, nous proposons une modification analogue du diagramme.

(Exemple 7)

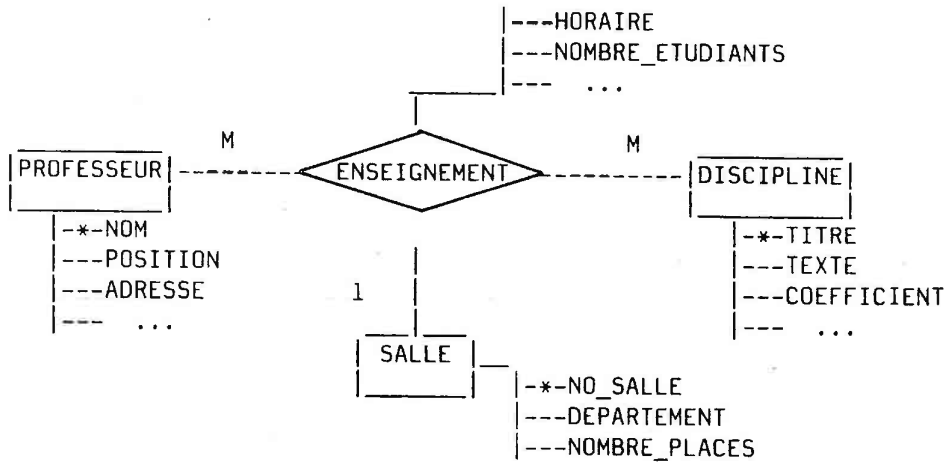
Reprenons l'exemple donné dans l'Exemple 1. Si chaque département dispose un certain nombre de salles de cours, alors on peut dire qu'il y a une DFA: NO_SALLE → DEPARTEMENT.

(a)



DFE: ø,
DFA: NO_SALLE → DEPARTEMENT; ...

(b)



DFE: PROFESSEUR DISCIPLINE → SALLE
DFA: NO_SALLE → DEPARTEMENT; ...

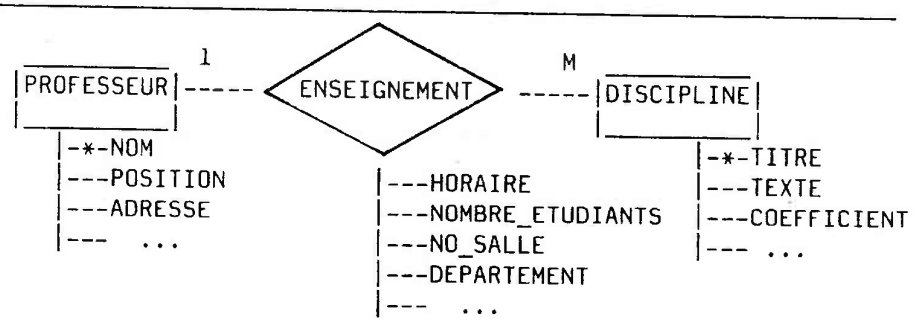
Remarquons que ces propositions de décomposition soulèvent deux problèmes:

- D'abord, le nouveau TE créé peut n'avoir pas de signification vis-à-vis du monde réel, car une dépendance fonctionnelle existante entre des attributs n'implique pas toujours un entité cachée.

- Deuxièmement, pour le cas des TAs, les cardinalités doivent être réexaminées après la modification, excepté les cas M-M, M-M-M,...

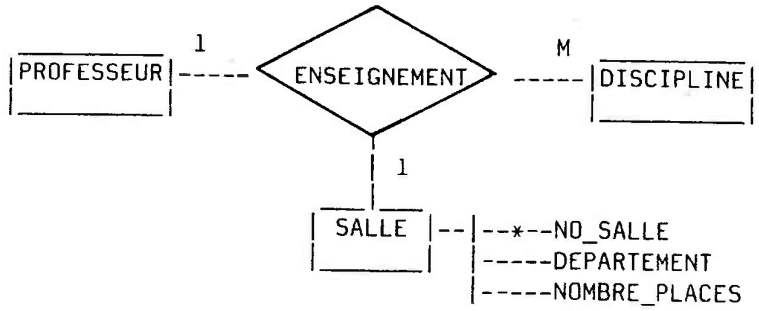
Examinons le cas où les cardinalités ne sont pas toutes M. S'il y a une condition du genre "la même discipline est enseignée par un seul professeur et un professeur peut enseigner plusieurs disciplines", ce fait se traduit par les cardinalités, 1 pour le TE PROFESSEUR et M pour DISCIPLINE, comme illustré par le diagramme (a) de la Fig.III-11. Le résultat de la modification proposée est illustré par le diagramme (b). Suivant notre interprétation, le diagramme modifié est interprété par un ensemble de DFEs: {DISCIPLINE PROFESSEUR → SALLE, DISCIPLINE SALLE → PROFESSEUR}, ce qui est différent de la réalité perçue préalablement par la DFE: DISCIPLINE →PROFESSEUR et la DFA: PROFESSEUR.NOM DISCIPLINE.TITRE → NO_SALLE (rappelons que la clé du TA ENSEIGNEMENT est l'union des clés principales des TEs PROFESSEUR et DISCIPLINE). Si on veut préserver ces dépendancess fonctionnelles, un diagramme E-A possible serait celui (c) obtenu par une transformation de (b) telle qu'il est décrit dans la section suivante.

(a)



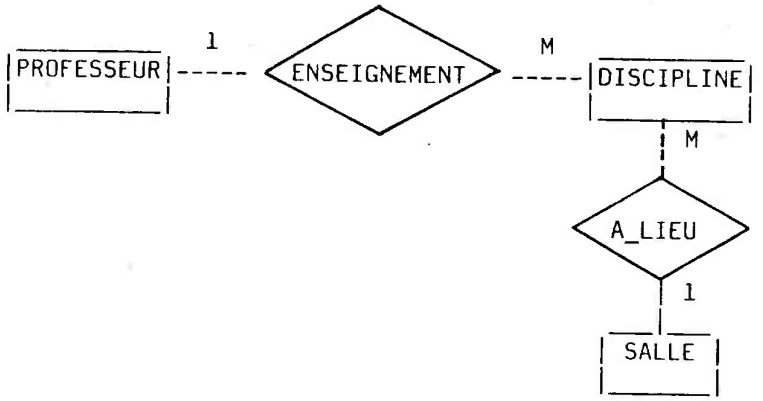
DFE: DISCIPLINE → PROFESSEUR;
DFA: NO_SALLE → DEPARTEMENT;
PROFESSEUR.NOM DISCIPLINE.TITRE → NO_SALLE; ...

(b)



DFE: PROFESSEUR DISCIPLINE → SALLE;
DISCIPLINE SALLE → PROFESSEUR;
DFA: NO_SALLE → DEPARTEMENT; ...

(c)



DFE: DISCIPLINE → PROFESSEUR; DISCIPLINE → SALLE
DFA: NO_SALLE → DEPARTEMENT; ...

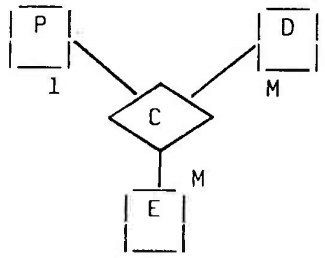
(Fig.III-11)

S'il y a une complexité de dépendances fonctionnelles entre des attributs d'un TE ou TA (ex. un attribut non-clé détermine fonctionnellement une partie de la clé dans un TE), cela rejoint exactement le problème de la décomposition d'un schéma de relation. Par conséquent, notre proposition n'est plus valable. Pourtant on peut espérer ne pas avoir une telle complexité de DFAs grâce à l'analyse préalablement faite par l'abstraction en terme d'entités et associations.

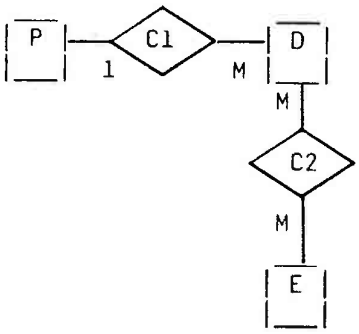
4.1.3. Diagramme E-A bien défini

Nous appelons **diagramme (schéma) E-A bien défini**, un diagramme (schéma) E-A où tous les TEs et TAs sont bien définis, et que la

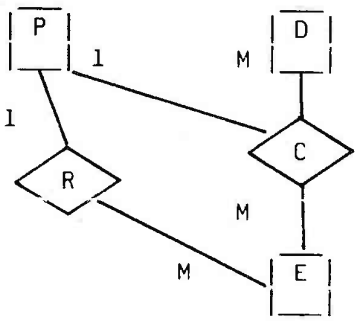
propriété de cardinalité représente exactement les DFEs du monde réel, éventuellement aucune. Nous présentons un scénario de la normalisation basée sur l'idée de décomposition en formes normale du modèle relationnel de Codd [CODD72-a]. P, D et E sont respectivement l'abréviation de PROFESSEUR, DISCIPLINE et ENSEIGNEMENT.



On établit une première approche du diagramme où tous les TEs ont la cardinalité M. On examine la cardinalité de chaque TE et on la change en 1, si nécessaire, par l'examen de la DFE appropriée.
Ici, si on a D E → P alors la cardinalité de P est 1. Diagramme ci-contre.



On peut se demander ensuite si P n'est pas déterminé par E ou D seul, au sein de C. Supposons que oui avec D → P (i.e. une discipline n'est enseignée que par un seul professeur). On décompose alors le TA concerné C en deux TAs C1 et C2 comme indiqué par le diagramme ci-contre.
Le rapprochement d'un étudiant donné avec ses professeurs se fait à travers les disciplines suivies par celui-ci. A cause D → P ce rapprochement est correct (cf. [AHO79]).



Supposons maintenant qu'au lieu de D → P nous ayons E → P, mais cette fois-ci dans le sens où un étudiant n'a qu'un seul professeur comme directeur de recherche.
Cette DFE ne contraint pas le TA C déjà spécifié, et concerne un autre TA (soit R) qu'il faut rajouter au diagramme comme indiqué ci-contre.

Ainsi, de proche en proche on peut raffiner un diagramme selon la réalité (perçue par le concepteur) qu'on veut modéliser.

Le scénario soulève les deux points importants suivants:

- Les DFEs ne sont pas automatiquement mélées dans le même domaine d'interprétation. Ceci dit, un concepteur a la possiblilité de localiser son intention par un sous-diagramme E-A, en particulier, à un TA à la fois. On peut étendre cette démarche pour fusionner progressivement deux sous-schémas E-A.

- La définition de DFE permet une analyse plus abstraite que celle de dépendance fonctionnelle définie au niveau des attributs. Ceci a pour avantage de permettre au concepteur de bien refléter la réalité du monde réel, de façon progressive, sans se préoccuper des détails dans un premier temps.

4.2. Règles de mise-à-jour des instances

Quand on considère les instances du schéma E-A, on doit prendre en compte les règles de mise-à-jour de ces instances, qui garantissent la cohèrence de l'état du schéma. Nous donnons ici deux règles:

(1) Chaque instance d'un TA doit faire référence aux instances de chaque TE participant.

Cette règle semble être triviale. Pourtant elle avertit du fait que si l'on supprime une entité d'un TE participant, la suppression se propage dans un ensemble d'associations définies avec celle-ci.

(Exemple 8)

Si un professeur quitte l'établissement, la règle implique qu'il faut faire la mise-à-jour (éventuellement suppression) de tous les enseignements donnés par ce professeur.

(2) Pour un TE obligatoire, au moment de la création d'une entité de ce type, il faut insérer également l'association définie avec cette entité.

Par exemple, si on a une contrainte "chaque professeur doit enseigner au moins une discipline", le TE PROFESSEUR sera représenté comme un TE obligatoire, illustré par le diagramme de l'Exemple 8. Quand un nouveau professeur arrive à l'établissement, non seulement on insère ce nouveau professeur dans l'ensemble des entités pour le TE PROFESSEUR, mais on doit aussi lui affecter des cours à donner. Sinon, la base contenant des instances du schéma serait momentanément incohérente. Ce risque devrait être bien maîtrisé par un système d'integrité.

5. Quelques types génériques d'association basés sur les notions d'abstraction de données

Dans le développement de notre compréhension d'un phénomène complexe, l'outil le plus puissant qui est offert à l'intellect humain est l'abstraction. L'abstraction résulte de la reconnaissance, de similarités entre certains objets, situations ou processus dans le monde réel, et de la décision de se concentrer sur ces similarités et d'ignorer, pour l'instant, les différences [HOARE72]. C'est justement à cause de la limite de notre intellect que l'abstraction devient primordiale; à un moment donné, le raisonnement précis d'un être humain est possible seulement en terme d'un petit nombre de concepts différents. Ce nombre "peut" varier autour du chiffre "7" selon l'individu à cause de:

- notre capacité de mémoire immédiate;
- notre capacité d'estimation absolue permettant de distinguer environ sept catégories;
- notre capacité d'appréhension qui embrasse environ six objets d'un seul coup d'oeil [MILLER56].

En ce qui concerne le domaine BD, quatre différentes formes d'abstraction de données sont retenues: classification, généralisation, agrégation et partition. Dans divers travaux sur l'extension du modèle E-A, un sous-ensemble de ces concepts est intégré, par exemple, le concept de généralisation dans [NAVATHE83] [SAKAI83], les concepts de généralisation et d'agrégation dans [SCHEUERMAN80].

Nous étudions ces quatre puissants outils intellectuels dans le cadre du modèle E-A. Nous montrons, d'une part, comment ces concepts sont déjà immanents dans le modèle et, d'autre part, nous proposons quelques types génériques d'association en considérant ces formes d'abstraction comme des aspects spécifiques d'association.

5.1. Classification - Instanciation

La **classification** est une forme d'abstraction dans laquelle une collection d'objets est considérée comme une classe d'objets de plus haut niveau [BRODIE84-b].

Une classe d'objets n'est en fait qu'une caractérisation précise des propriétés partagées par chaque objet dans la collection. Ici, un objet est une représentation uniforme de l'information (donnée ou traitement) dans un ordinateur.

Des objets qui partagent les propriétés définies dans la classe sont appelés des instances de la classe. L'**instanciation** est un concept opposé à la classification et consiste à créer des instances (on dit "instancier") d'une classe.

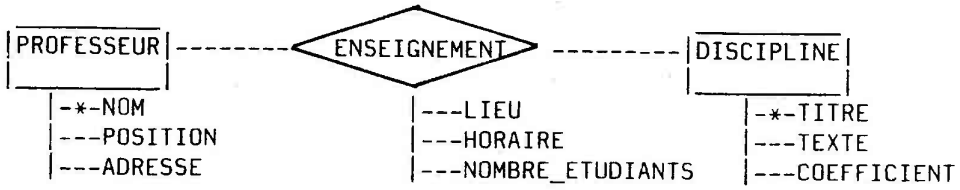
Ces concepts sont à l'origine de classe d'objets du langage SIMULA [DAHL72]. Ils sont adoptés et enrichis par l'approche des "langages orientés-objets", par exemple, SMALLTALK [GOLDBERG83].

Dans le cadre du modèle E-A, nous apercevons ce mécanisme d'abstraction au niveau d'un TE et ses entités, d'un TA et ses associations et d'un attribut et ses valeurs. Plus généralement, nous pouvons considérer qu'un schéma de données est une classification des données stockées dans la base.†

(Exemple 9)

Reprenons le cas donné dans l'Exemple 1.

(†) Dans le cadre du modèle relationnel de données, [DATE81] emploie intention/extension d'une relation pour différencier l'aspect définition d'une relation (schéma de relation) et l'aspect représentation (collection de tuples). Nous pensons que l'intention est une classification d'une collection d'extensions et qu'elle définit toutes les propriétés communes des extensions.



(1) Un TE PROFESSEUR, caractérisé par les attributs NOM, POSITION et ADRESSE, est une classification d'un ensemble d'entités:

{ <Dupont, Maître de Conférence, 15 Rue Jacquinot>,
<Durand, Assistant, 2 Place Carnot>,
<Martin, Professeur, 5 Boulevard Jean-Jaurés>, ...}

(2) Un TE DISCIPLINE, caractérisé par les attributs TITRE, TEXTE et COEFFICIENT, est une classification d'un ensemble d'entités:

{ <Base de Données, Introduction aux SGBDs relationnels, 3>,
<Programmation, Programme=Structure_de_données+Algorithme, 3>,
<Système d'exploitation, Théorie de SE, 3>, ...}

(3) Un TA ENSEIGNEMENT défini entre PROFESSEUR et DISCIPLINE et caractérisé par les attributs LIEU, HORAIRE et NOMBRE_ETUDIANTS est une classification d'un ensemble d'associations:

{ <Dupont, Base de Données, Amphi 3, Mercredi 9H00, 50>,
<Dupont, Programmation, Salle S5, Jeudi 14H00, 25>,
<Durand, Système d'exploitation, Amphi 1, Jeudi 14H00, 45>, ...}

Dans cet exemple, la notion de classe d'objets est remplacé par un type d'objets (que nous utilisons désormais à la place du terme "classe d'objets"). Des associations définies entre entités sont également considérées comme objets dans la mesure où on les considére comme informations stockées et traitées par l'ordinateur.

Chaque forme d'abstraction peut être expliquée en terme d'héritage de propriétés. En ce qui concerne la classification,

toutes les instances d'un type héritent des propriétés décrites dans le type.

Dans la plupart des modèles de données utilisés en BD, l'application de la classification demeure entre objets et leur type. Nous essayons d'appliquer cette classification au niveau des types pour faire une abstraction de plus haut niveau, dite "type générique". Nous nous contentons de proposer certains types génériques d'association afin d'étendre le modèle E-A, en nous basant sur les différentes formes d'abstraction: généralisation, agrégation et partition.

5.2. Généralisation - Spécialisation

Originalement, la **généralisation** est proposée comme une forme d'abstraction dans laquelle un ensemble d'objets similaires est considéré comme un objet générique [SMITH77-a,b].

C'est un concept proche de la classification. Pour cette raison, des auteurs distinguent ces deux concepts en proposant que:

- la généralisation est une abstraction des types (classes) d'objets pour aboutir à un type générique [BORGIDA84];
- la classification indique spécialement la généralisation de premier niveau, c'est à dire, la généralisation entre objets et type [TSICHRITZIS82].

Dans le contexte de l'extension du modèle E-A, nous choisissons la définition suivante:

> la généralisation est une forme d'abstraction dans laquelle une relation† entre types d'objets est considérée comme un type génériqué de plus haut niveau [BRODIE84-b]

(†) Ici le terme "relation" doit être compris comme un terme général au sens rapport, parenté ou "relationship" en anglais.

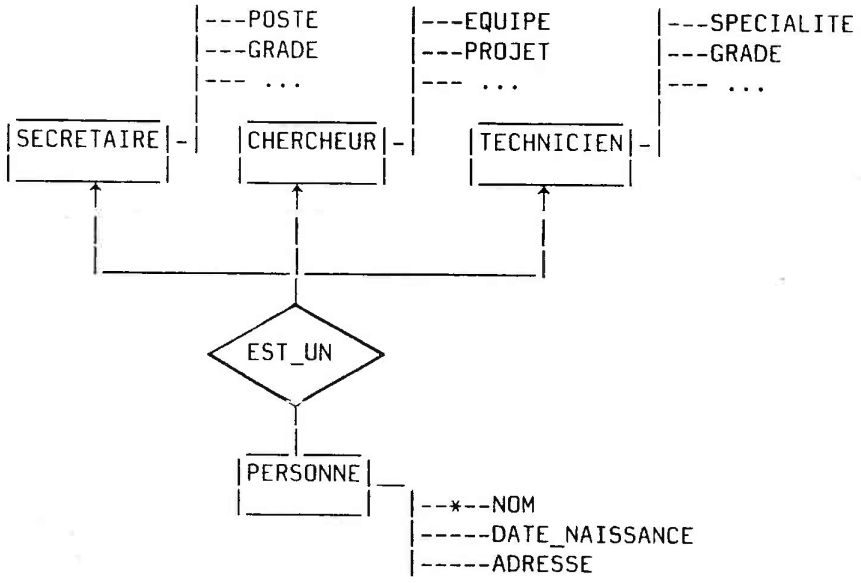
car notre intention est de proposer quelques types génériques d'association. La **spécialisation** est un concept opposé à la généralisation, qui consiste à préciser les propriétés propres au type spécialisé.

Héritage de propriétés: En général, un type généralisé doit avoir toutes les propriétés communes à ses constituants. C'est à dire que toutes les propriétés d'un type généralisé peuvent être transmises en héritage aux types spécialisés.

Nous proposons un type générique d'association, nommé EST_UN pour réaliser cette forme d'abstraction dans le modèle E-A. Voir l'Exemple 10. La flèche indique la direction de l'héritage de propriétés dans le diagramme E-A étendu (cf. Association ISA [ULLMAN82,p.13]).

(Exemple 10)

Un TE généralisé PERSONNE concernant les personnes dans un centre de recherche peut être conçu à travers une généralisation des TEs SECRETAIRE, CHERCHEUR et TECHNICIEN. Les attributs communs NOM, DATE_NAISSANCE et ADRESSE sont hérités par chacun de TEs SECRETAIRE, CHERCHEUR et TECHNICIEN.



5.3. Agrégation - Décomposition

L'**agrégation** est proposée comme une abstraction dans laquelle une relation[†] entre objets est considérée comme un objet de plus haut niveau [SMITH77-b].

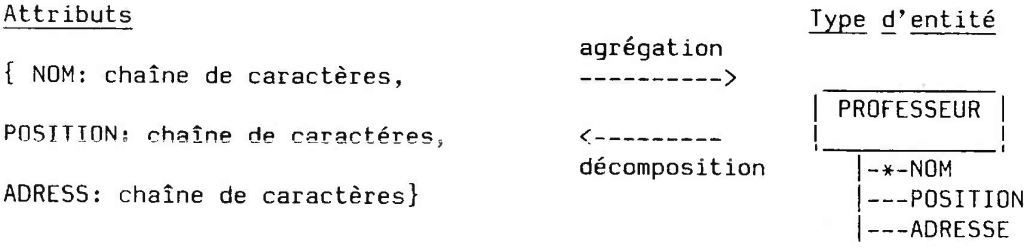
Nous étendons cette définition comme une abstraction dans laquelle une relation entre objets (ou types) composants est considérée comme un objet (ou type) agrégat de plus haut niveau. Cette définition implique que l'objet agrégat a, comme partie de sa structure, les objets composants. La **décomposition** est un concept opposé à l'agrégation (cf. "individuation" [SOWA84,pp.115-123]).

Héritage de propriétés: Les propriétés de chaque objet composant sont héritées par l'objet agrégat.

On peut considérer qu'un TE est une agrégation des attributs et qu'un TA est une agrégation des clés des TEs participants et ses propres attributs dans le modèle E-A. Nous proposons un type générique d'association, nommé FAIT_PARTIE_DE, pour intégrer l'agrégation dans le modèle E-A au niveau des types. Voir l'Exemple 11. La flèche indique la direction de l'héritage de propriétés dans le diagramme E-A étendu.

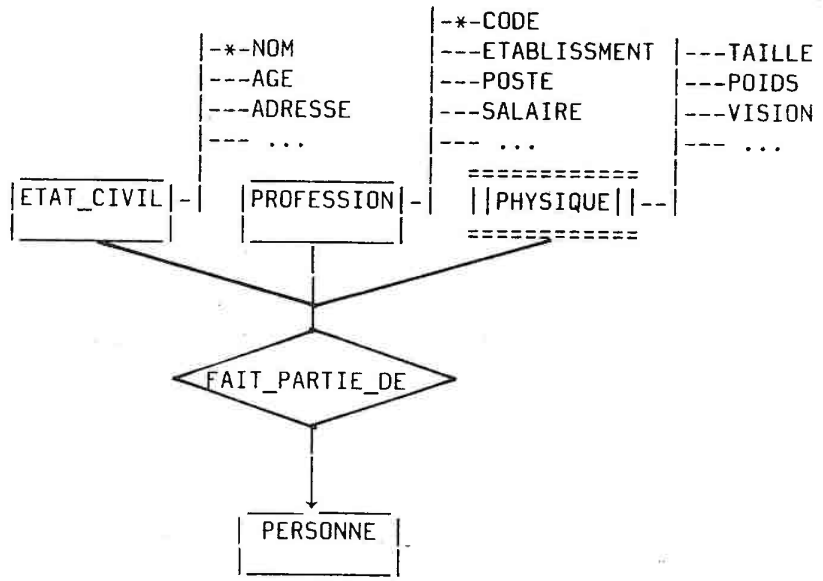
(Exemple 11)

(1) Un TE PROFESSEUR est une agrégation des attributs NOM, POSITION et ADRESSE.

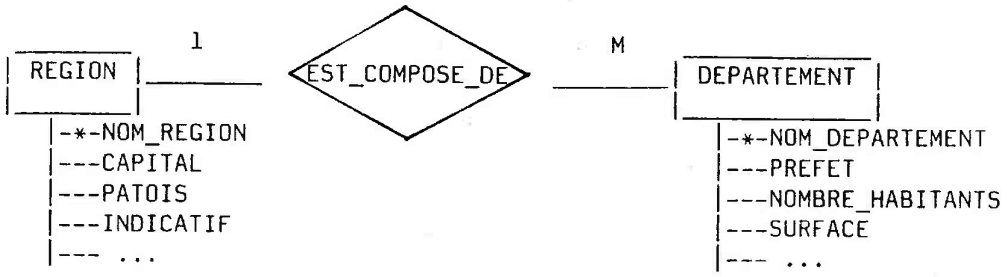


(†) Voir le bas de page du paragaphe 5.2..

(2) Un TE agrégat PERSONNE peut être considéré comme agrégation des TEs ETAT_CIVIL, PROFESSION et PHYSIQUE à travers d'un TA générique FAIT_PARTIE_DE. A noter que le TE PHYSIQUE est un TE faible.



Nous allons évoquer maintenant un cas spécial d'agrégation, celui d'agrégation des composants de type identique. Par exemple, une région est une agrégation de départements: la région Lorraine est une agrégation des départements Meurthe-et-Moselle, Moselle, Meuse et Vosges. Dans cet exemple, la région Lorraine n'est pas une simple collection des quatre département, puisqu'elle peut être, elle-même, considérée comme un nouvel objet d'un type REGION et ayant ses propres attributs. On peut tout de suite imaginer certains attributs: CAPITAL, PATOIS, INDICATIF, ... liés au type REGION. Dans [OBENSON83], l'auteur nomme cette forme d'abstraction "collection". Le type générique d'association proposé ne permet pas d'exprimer ce cas. Pourtant nous pouvons résoudre ce problème en définissant les TEs REGION et DEPARTEMENT et un TA EST_COMPOSE_DE comme suit:



5.4. Regroupement - Partition

La **partition** est une forme d'abstraction dans laquelle une relation entre objets membres est considérée comme un objet ensemble d'objets reliés de plus haut niveau [BRODIE82].† Elle permet de définir des sous-ensembles dans un type d'objets. Le concept opposé à la partition est le **regroupement.**

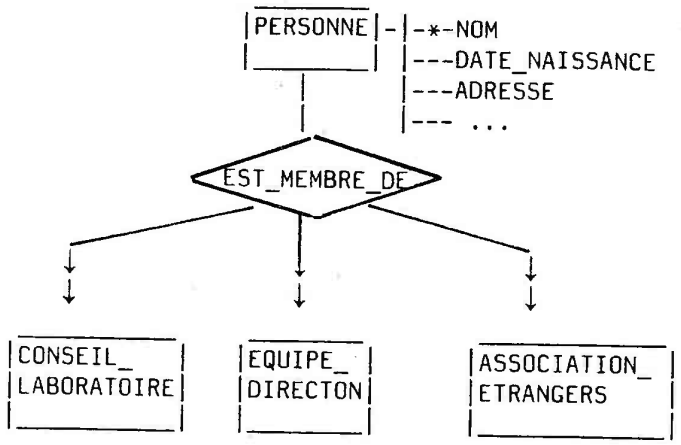
Héritage de propriétés: En général, toutes les propriétés des membres sont héritées par l'objet ensemble.

Nous proposons un type générique d'association, nommé EST_MEMBRE_DE, pour intégrer la partition dans le modèle E-A. Voir l'Exemple 12. La flèche montre la direction de l'héritage de propriétés dans le diagramme E-A étendu.

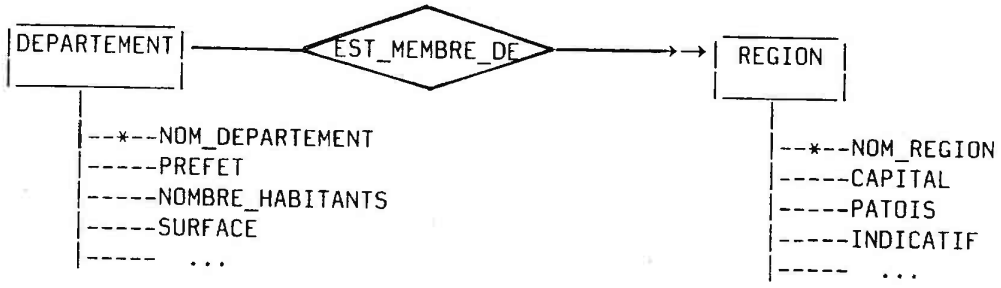
(Exemple 12)

(1) A l'aide du type générique d'association EST_MEMBRE_DE, nous pouvons définir des sous-ensembles de personnes selon leur participation à diverses activités.

(†) Originalement Brodie utilise le terme "association" (en anglais) mais puisque dans notre propos, ce mot a une signification bien particulière, nous préférons utiliser "partition".



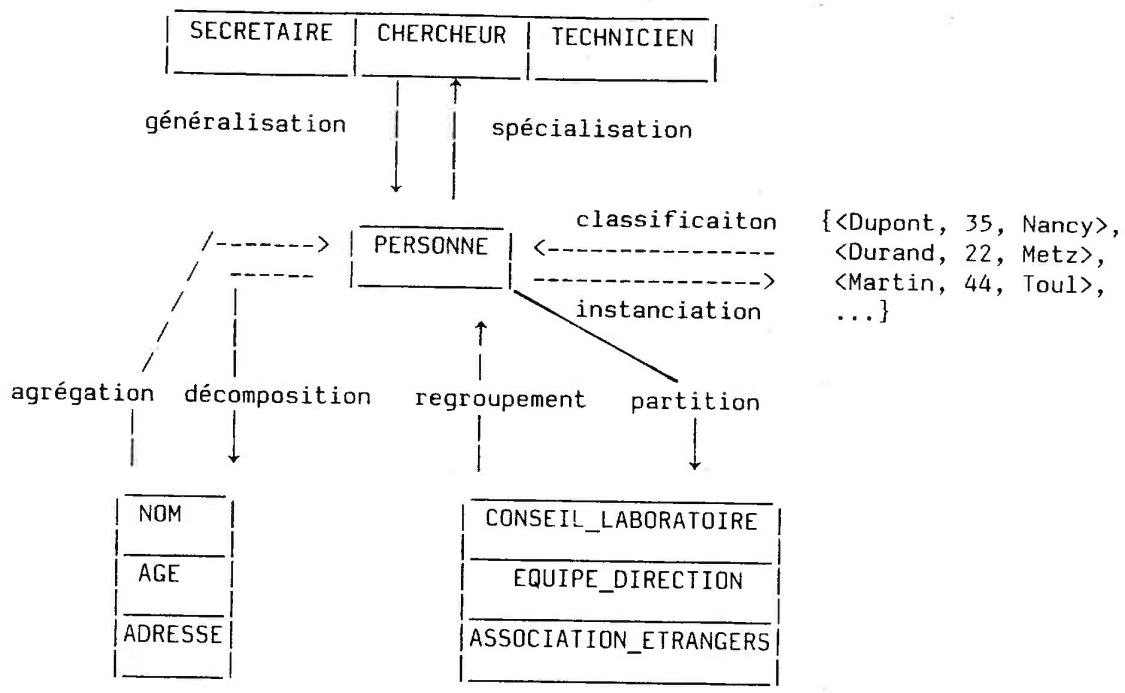
(2) Un TE REGION peut être conçu comme un objet ensemble du TE DEPARTEMENT. Un sous-ensemble de départements du TE DEPARTEMENT forme une région du TE REGION.



5.5. Remarques concluantes

Dans cette section, nous avons montré l'intégration de différents concepts d'abstraction de données dans le modèle E-A. La démarche que nous avons prise est d'appliquer ces concepts au niveau de TA.

Il faut noter que les quatre formes d'abstraction que nous avons introduites sont orthogonales dans le sens où on peut les appliquer indépendamment l'une des autres. Un exemple pour un TE PERSONNE est illustré dans la Fig.III-12.



(Fig.III-12) Orthogonalité de différentes formes d'abstraction

Remarquons d'abord que nous devons encore appliquer le concept de la classification pour fournir un mécanisme permettant d'utiliser des TAs prédéfinis, parce que l'on trouve souvent les mêmes TAs définis dans des contextes différents. Par exemple, "un module source EST_ECRIT_PAR un programmeur", "un document de spécification EST_ECRIT_PAR un analyste", et "un cahier de charges EST_ECRIT_PAR un demandeur".

Deuxièmement, puisque, tout au long de la discussion, nous avons insisté sur l'aspect abstraction (généralisation, agrégation), nous devons aussi prendre en compte l'aspect analytique de chaque concept (spécialisation, décomposition). Par exemple, après avoir conçu un TE, on peut l'analyser en détail, non seulement par les attributs possibles sur le TE, mais aussi par les concepts: spécialisation, décomposition ou encore partition.

Finalement, nous pensons que la généralisation et l'agrégation permettent de définir un nouveau TE de plus haut niveau et que la spécialisation, la décomposition et la partition peuvent servir à définir les différentes vues d'un TE.

6. Conclusion

Une hypothèse sous-jacente du modèle E-A est: "quelque chose dans le monde réel peut être perçu comme une entité et le reste ne l'est pas" [KENT84]. En plus, nous ne pouvons pas nier notre intention finale, qui est de représenter finalement cette entité soit sous forme de relation, soit par quelque chose proche de "Record". Kent classe ce genre de méthodologie comme l'approche basée sur l'entité, en dégageant deux compléments nécessaires à cette approche:

- un critère pour décider qu'est-ce qui est une entité dans le monde réel et qu'est-ce qui ne l'est pas;
- la distinction entre des attributs d'une entité et d'autre entités associées à la première.

A notre avis, ces deux points resteront assez longtemps une question ouverte.

Bien que le problème fondamental ci-dessus ne soit pas résolu, nous avons proposé, dans ce chapitre, une nouvelle interprétation du modèle E-A en terme de Dépendance Fonctionnelle entre Entités, en étudiant la propriété de cardinalité d'un TE au sein d'un TA, et ensuite un guide de l'application du modèle en étudiant la normalisation de schéma E-A basée sur la première étude. Nous avons également étendu le modèle E-A en intégrant les quatre concepts d'abstraction de données: classification, généralisation, agrégation et partition. Nous pensons que nos études pourront contribuer à ce que le modèle E-A serve effectivement de méthode de conception de la BD et de moyen de communication entre le concepteur de BD et l'utilisateur final, grâce à sa facilité de compréhension et d'utilisation et grâce à sa représentation graphique du schéma.

*
***
*****

CHAPITRE IV - CONSTRUCTION DE SCHEMA RELATIONNEL EN TROISIEME FORME NORMALE PAR TRANSFORMATION DE SCHEMA ENTITE-ASSOCIATION

Résumé

Une méthode de transformation du schéma Entité-Association (E-A) en schéma relationnel en troisième forme normale est présentée. La transformation est faite en prenant en considération des dépendances fonctionnelles entre attributs, des dépendances fonctionnelles entre entités et des dépendances d'inclusion. Cette méthode sert de guide pour une définition systématique d'une base de données relationnelle, en utilisant le modèle E-A qui permet de mieux saisir la sémantique du monde réel. Un système interactif de construction des schémas E-A et un système de transformation sont réalisés en Le_Lisp/CEYX, sous UNIX.

Sommaire

1. Introduction
2. Transformation d'un schéma Entité-Association en schéma relationnel
   - 2.1. Etape-1 : Construction des schémas de relation pour chaque type d'entité et chaque type d'association sans se préoccuper des informations de l'obligation de type d'entité (Transformation)
   - 2.2. Etape-2 : Modification du schéma relationnel obtenu à l'Etape-1 à partir de l'information indiquant des types d'entité obligatoires (Optimisation)
3. Analyse des clés dans l'algorithme de transformation
4. Réalisation
   - 4.1. Système interactif de construction des schémas Entité-Association
   - 4.2. Système de transformation d'un schéma Entité-Association en schéma relationnel
5. Conclusion

CHAPITRE IV
CONSTRUCTION DE SCHEMA RELATIONNEL EN TROISIEME FORME NORMALE PAR TRANSFORMATION DE SCHEMA ENTITE-ASSOCIATION

1. Introduction

Malgré sa puissance théorique, le modèle relationnel est sémantiquement pauvre. Il est difficile dans le modèle relationnel de saisir des aspects sémantiques du monde réel à moins d'introduire des contraintes d'intégrité, d'une façon ou d'une autre. Par ailleurs, le modèle Entité-Association (E-A) nous permet de décrire le monde réel d'une manière beaucoup plus naturelle que le modèle relationnel et aussi d'appréhender plus d'aspects sémantiques qu'il n'est possible avec le modèle relationnel.

L'objectif de ce chapitre est de donner une méthode de conception d'une base de données relationnelle, s'appuyant sur la transformation d'un schéma E-A en un schéma relationnel de Troisième Forme Normale (3FN), y compris des contraintes d'intégrité de type Dépendance d'Inclusion (DI). C'est une contrainte d'intégrité intervenant entre deux relations, non nécessairement distinctes, et spécifiant que toute valeur d'un attribut appartenant à une relation doit figurer comme valeur d'un attribut dans l'autre relation (voir le CHAPITRE I, le sous-paragraphe 3.1.3.).

Pour construire un schéma relationnel en 3FN, un algorithme de synthèse a été proposé dans [BERNSTEIN76]. Cet algorithme a pour but d'obtenir un schéma relationnel contenant le nombre minimum de schémas de relation en préservant des Dépendances Fonctionnelles (DFs). L'algorithme a été amélioré pour permettre à des schémas différents de relation ayant des clés équivalentes d'être fusionnés en un seul schéma de relation [DELOBEL82,pp.412-418]. Différemment de l'algorithme de synthèse, on peut construire un schéma relationnel en 3FN de Boyce-Codd (3FNBC) par l'algorithme de décomposition en préservant les informations de la BD [AHO79]. Avec l'algorithme de [RISSANEN77], il est possible d'obtenir un schéma relationnel en 3FN

qui préserve à la fois des DFs et des informations. Néanmoins, ces algorithmes proposent des manipulations purement syntaxiques (ex. DFs pour l'algorithme de synthèse et opérations relationnelles: jointure et projection pour l'algorithme de décomposition) et ne tiennent pas compte de la signification relative de DFs ou d'opérations relationnelles. Ceci est normal parce que leur but est de trouver un schéma conduisant à un minimum de stockage de données dans la base.

A partir d'un schéma relationnel, éventuellement en 3FN, avec un ensemble de DIs, [CASANOVA84-a] a développé un algorithme qui transforme d'abord le schéma donné en un schéma intermédiaire (l'auteur l'appelle schéma relationnel en forme normale E-A) et puis engendre un diagramme E-A. En effet, le schéma relationnel a une interprétation en terme du modèle E-A, qui aide à mieux comprendre la sémantique d'une base de données relationnelle. L'auteur s'est donc contenté de décomposer un schéma relationnel en des TEs (réguliers ou faibles) et en des TAs sans préciser les caractéristiques de l'association (par exemple, la cardinalité qu'on peut déduire dans certains cas) qui pourtant permettent d'exprimer des informations utiles.

Mis à part l'algorithme de synthèse et celui de décomposition, pour la construction d'un schéma relationnel, beaucoup d'auteurs cherchent les méthodes permettant de transformer le schéma E-A en un schéma relationnel comme une solution alternative. Dans son ouvrage [HOWE83], l'auteur introduit la notion de l'entité obligatoire à la place de l'entité faible et il étudie, cas par cas, la transformation d'un sous-diagramme E-A élémentaire en un schéma relationnel. Dans [WONG80] les auteurs étudient la transformation en s'appuyant sur l'idée que l'on peut engendrer un seul schéma de relation pour le TE et le TA binaire dont les cardinalités sont M-1 ou 1-1. Ils proposent à cet égard que les attributs d'un tel TA puissent être considérés comme ceux du TE fonctionnellement déterminant. Dans [WILKIE83] les auteurs proposent des règles de transformation et développent un système basé sur ces règles et sur l'algorithme de synthèse. L'avantage de leur système est de pouvoir introduire des Dépendances

Fonctionnelles entre Entités (DFE), et d'aider l'utilisateur à décomposer le diagramme E-A pour obtenir un diagramme E-A normalisé que l'on peut traduire directement en schéma relationnel en 3FN. En revanche, pour traiter des DFEs dans le schéma relationnel, ils ajoutent la clé du TE fonctionnellement déterminé dans le schéma de relation représentant le TE déterminant. Cette solution présente à notre avis les inconvénients suivants:

- la redondance d'information parce que la clé du TE déterminé apparaît une autre fois dans le schéma relationnel;

- la difficulté de mise à jour parce que pour mettre à jour une relation représentant un TE et contenant la clé d'un autre TE, la connaissance de la valeur de la clé ajoutée est nécessaire si on ne tolère pas les valeurs "nulles". Ceci reste également vrai pour le travail de [WONG80].

Dans notre approche, nous traduisons une DFE comme une DF dans un schéma de relation représentant un TA. Cela nous amène à modifier les clés d'un schéma de relationnel au lieu de le faire comme [WILKIE83]. Et nous introduisons aussi des DIs dans le résultat de la transformation pour prévenir des anomalies de mise à jour.

Dans la section 2, nous expliquons avec des exemples figuratifs l'algorithme de transformation en suivant les cinq étapes progressives. La première version de l'algorithme était proposée dans [KIM85]. Les études sur les certaines caractéristiques du modèle E-A [KIM86-a] (le CHAPITRE III) nous ont permis d'améliorer cet algorithme [KIM86-b] et c'est ce dernier que nous présentons ici. Dans la section 3, nous analysons les clés dans l'algorithme proposé, en vue de montrer que le résultat de l'algorithme est en 3FN. Dans la section 4, nous présentons les deux systèmes implantés sur la machine VAX750 sous le système d'exploitation UNIX: Système de construction des schémas E-A et Système de transformation d'un schéma E-A en schéma relationnel.

2. Transformation d'un schéma E-A en schéma relationnel

Dans cette section, nous proposons un algorithme qui transforme un schéma E-A, représenté par un diagramme, en un schéma relationnel qui est composé d'un ensemble de schémas de relation et d'un ensemble de dépendances d'inclusion inter-relations.

Pour que l'algorithme de transformation puisse donner un schéma relationnel en 3FN, nous supposons que chaque TE contienne une clé principale et que les attributs non clés d'un TE ou TA soient mutuellement indépendants (c'est à dire que des TEs et des TAs soient bien définis). Cette hypothèse est liée à la normalisation du diagramme E-A, déjà discutée précédemment (voir le CHAPITRE III, la section 4).

2.1. Etape-1 : Construction de schémas de relation pour chaque type d'entité et chaque type d'association sans se préoccuper des informations de l'obligation de type d'entité (Transformation)

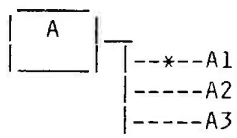
(1) Construire un schéma de relation pour chaque type d'entié régulier et ensuite pour chaque type d'entité faible

Pour chaque TE régulier, on construit un schéma de relation constitué par les attributs du TE. La clé du schéma de relation est la même que celle du TE. Voir la Fig.IV-1, (a).

Pour chaque TE faible, on construit un schéma de relation constitué par (i) les attributs du TE faible, (ii) les attributs du TA faible concerné et (iii) l'attribut clé ou le groupe d'attributs clés du TE parent associé. La clé du schéma de relation est l'union de la clé du TE faible et celle du TE parent.

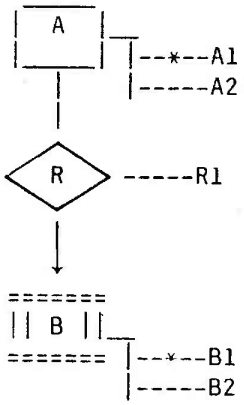
Ensuite, on déduit la contrainte de DI suivante: les valeurs des attributs (iii) ci-dessus pour cette relation figurent comme valeurs des mêmes attributs dans la relation parente associée, parce qu'une entité faible n'existe que par rapport à l'association avec une entité parente. Voir la Fig.IV-1, (b).

(a)



=> (A[A1,A2,A3], {A1})†

(b)



=> (A[A1,A2], {A1})
(B[A1,R1,B1,B2], {A1B1})

et

B[A1]⊂A[A1]

(Fig.IV-1) Etape-1, (1)

(2) Construire un schéma de relation pour chaque type d'association régulier et déduire des dépendances d'inclusion

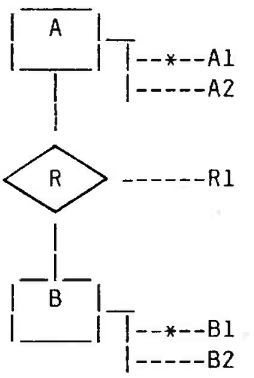
Le schéma de relation pour un TA est constitué des attributs clés des TEs participants, dont la juxtaposition forme la clé de la relation, et des attribut du TA (s'il y en a). Pour chaque attribut clé de la relation ainsi construite, on déduit une DI telle que toute valeur de l'attribut clé doit figurer comme valeur d'un attribut dans la relation obtenue à partir du TE dont cet attribut est clé. Voir la Fig.IV-2.

Si un TA est défini sur le même TE, on peut aussi appliquer l'algorithme de l'étape (2) pourvu de fournir un mécanisme pour donner

(†) [Notation] Un schéma de relation est noté (R[X,Y,Z],{K1,K2}) où R est le nom de schéma, X, Y et Z sont des attributs constituant le schéma et {K1,K2} est un ensemble de clés parmi lesquelles la clé principale est soulignée. Voir le CHAPITRE I, les sous-paragraphes 3.1.2. et 3.1.3..

un nouveau nom à l'attribut clé du TE qui a deux différents rôles dans le contexte du TA. Voir la Fig.IV-2, (c).

(a)



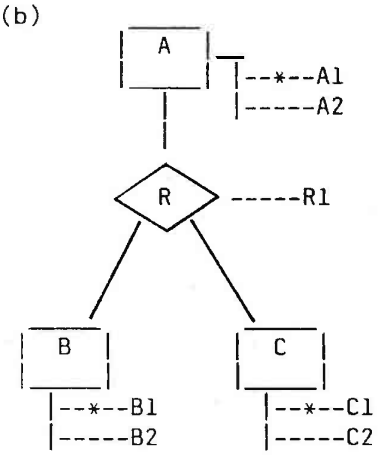
=>

(A[A1,A2], {A1})
(B[B1,B2], {B1})
(R[A1,B1,R1], {A1B1})

et

R[A1]⊂A[A1]
R[B1]⊂B[B1]

(b)



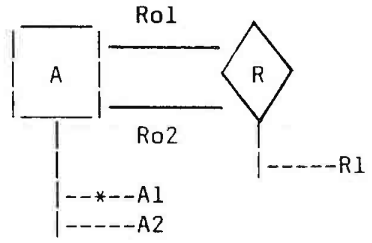
=>

(A[A1,A2], {A1})
(B[B1,B2], {B1})
(C[C1,C2], {C1})
(R[A1,B1,C1,R1], {A1B1C1})

et

R[A1]⊂A[A1]
R[B1]⊂B[B1]
R[C1]⊂C[C1]

(c)



=>

(A[A1,A2], {A1})

(R[$A1^{Rol}$,$A1^{Ro2}$,R1], {$A1^{Rol}$ $A1^{Ro2}$})

et

$R[A1^{Rol}]$⊂A[A1]
$R[A1^{Ro2}]$⊂A[A1]

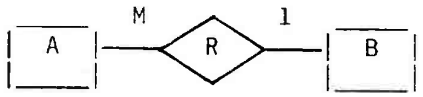
(Fig.IV-2) Etape-1, (2)

(3) Modifier la clé du schéma de relation représentant un type d'association à partir des dépendances fonctionnelles entre entités, exprimées par des cardinalités.

(3-1) S'il y a une DFE déduite dans un TA (si le TA est de type M-1, M-M-1, ...), c'est à dire qu'il y a un seul TE dont la cardinalité est 1, l'attribut clé de ce TE ne fait pas partie de la clé de relation construite à l'étape (2). Voir la Fig.IV-3.

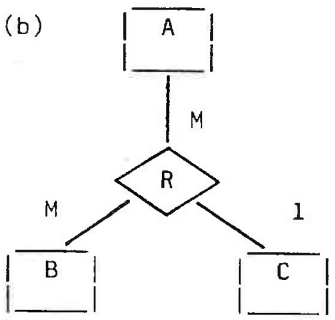
(3-2) S'il y a plus d'une DFE dans un TA (si le TA est de type 1-1, M-1-1, 1-1-1, ...,), c'est à dire qu'il y a plus d'un TE dont la cardinalité est 1, on applique autant de fois le processus de l'étape (3-1), pour qu'à chaque fois, on construise une clé candidate. Ensuite, on choisit une clé principale parmi ces clés candidates. Voir la Fig.IV-4.

(a)



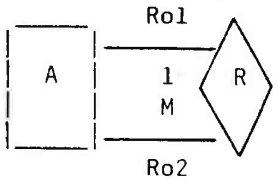
modifier la relation R;
(R[A1,B1,R1], {A1})

(b)



(R[A1,B1,C1,R1], {A1B1})

(c)

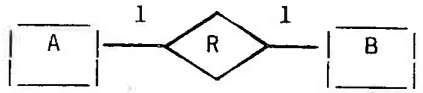


(R[A1^{Rol},A1^{Ro2}, R1], {A1^{Ro2}})

[Note] Les diagrammes se rapportent aux figures précédentes pour les attributs.

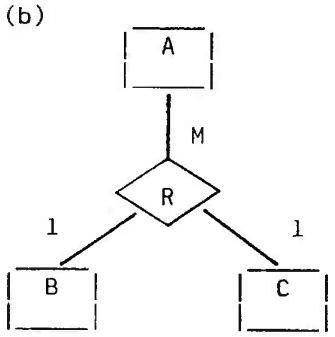
(Fig.IV-3) Etape-1, (3-1)

(a)



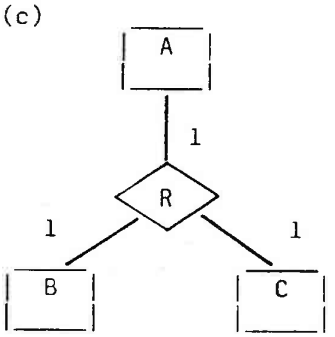
(R[A1,B1,R1], {A1,B1})

(b)



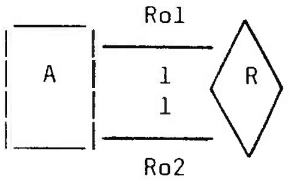
(R[A1,B1,C1,R1], {A1B1, A1C1})

(c)



(R[A1,B1,C1,R1], {A1B1, A1C1, B1C1})

(d)



(R[$A1^{Ro1}$,$A1^{Ro2}$, R1], {$A1^{Ro1}$, $A1^{Ro2}$})

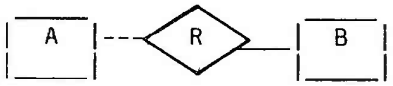
(Fig.IV-4) Etape-1, (3-2)

2.2. Etape-2 : Modification du schéma relationnel obtenu à l'Etape-1 à partir des informations indiquant des types d'entité obligatoires (Optimisation)

(4) Déduire la nouvelle dépendance d'inclusion à partir de l'information indiquant le type d'entité obligatoire et l'ajouter au schéma relationnel

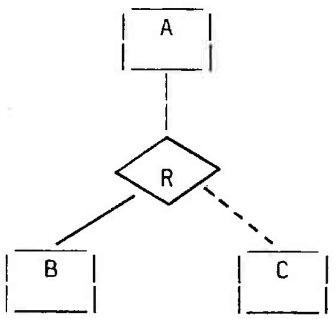
Une DI S[X]⊂R[X] est déduite comme suit: toute valeur de la clé X du schéma de relation représentant un TE obligatoire S doit figurer comme valeur de R[X] où R est un schéma de relation représentant un TA auquel S participe avec obligation. Voir la Fig.IV-5.

(a)



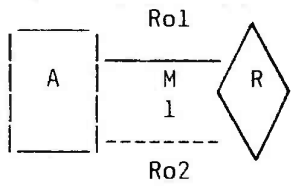
ajouter la DI: B[B1]⊂R[B1]

(b)



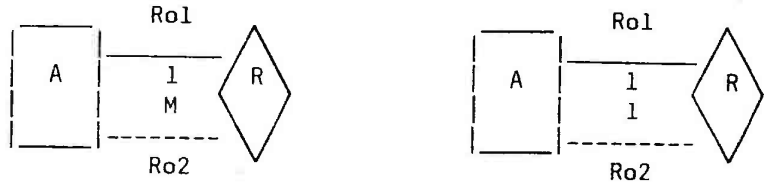
ajouter les DIs:
A[A1]⊂R[A1]
B[B1]⊂R[B1]

(c)



ajouter la DI: $A[A1] \subset R[A1^{Rol}]$

[Remarque] Les deux cas suivants n'ont pas de sens:



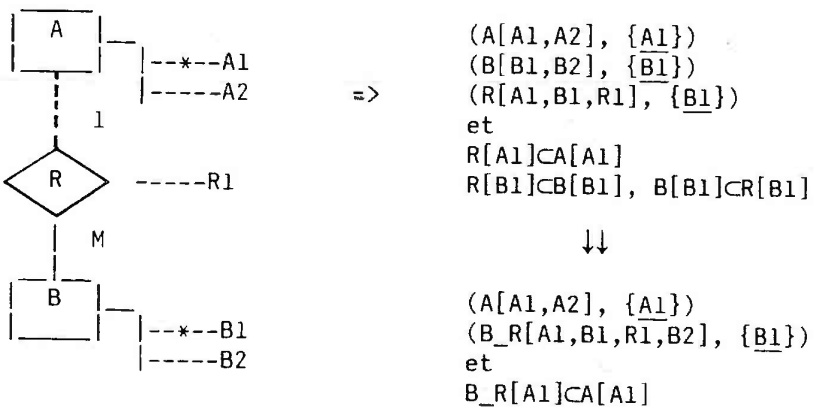
(Fig.IV-5) Etape-2, (4)

---

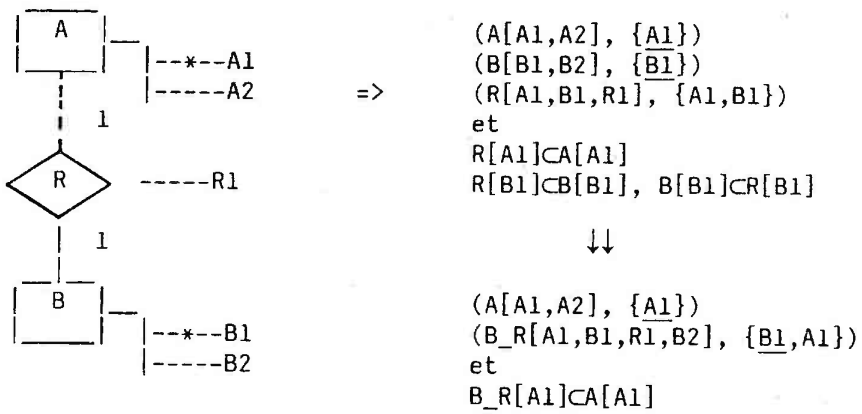
(5) Modifier le schéma relationnel en fusionnant les deux schémas de relation qui portent la même clé et aussi les dépendances d'inclusion réciproques par rapport aux clés

A l'étape (5), la clé commune peut être une des clés candidates. Par exemple, dans un schéma relationnel, s'il existe deux schémas de relation (R[X,Y,Z],{X,Y}) et (S[U,X,V],{U,X}) avec deux DIs R[X]⊂S[X] et S[X]⊂R[X], on applique l'étape (5) et on obtient comme résultat (R_S[Y,U,X,Z,V],{X,U,Y}). Pour renommer le nouveau schéma de relation, nous utilisons la concaténation des noms comme illustré par R_S ci-dessus. Dans ce cas là, les clés qui ne sont pas communes sont devenues les clés candidates dans le nouveau schéma de relation fusionné. Voir la Fig.IV-6.

(a)



(b)



(Fig.IV-6) Etape-2, (5)

Cette dernière étape est proposée comme une étape d'optimisation pour diminuer le nombre de schémas de relation engendrés. Cela concerne un des objectifs du modèle relationnel, non-redondance (représentation d'une information une seule fois dans la base). Pourtant on risque de perdre la sémantique originale, saisie par un diagramme E-A au niveau de conception. C'est la raison pour laquelle nous proposons une intervention du concepteur qui décide de l'appliquer ou non et nous préférons sacrifier la non-redondance en faveur d'une représentation plus naturelle.

3. Analyse des clés pour l'algorithme de transformation

[Notation] Quand on note XY où X et Y sont les deux ensembles d'attributs, XY signifie XUY. Quand on note AB où A et B désignent chacun un attribut, AB signifie l'ensemble {A,B}. De même pour XYZ, ABC, ...

A partir d'un diagramme E-A qui contient m TEs (réguliers ou faibles), et après avoir effectué l'étape (1), on obtient m schémas de relation:

$$(E_i[EC_i,ENC_i],\{EC_i\}) \text{ pour } i=1,2,\ldots,m;\ m\geq 2$$

où E_i est le nom d'un TE, EC_i un ensemble d'attributs clés et ENC_i un ensemble d'attributs non clés éventuellement vide. Ce schéma contient implicitement les DFs:

$$EC_i \rightarrow ENC_i \text{ pour } i=1,2,\ldots,m;\ m\geq 2 \quad ---(i).$$

Après l'étape (2), on obtient n schémas de relation de plus pour n TAs réguliers définis parmi $E_1,E_2,\ldots E_m$ avec des DIs:

$$(R_j[EC_{k1}EC_{k2}\ldots EC_{kp_j},RNC_j],\{EC_{k1}EC_{k2}\ldots EC_{kp_j}\})$$

$$\text{pour } j=1,2\ldots,n;\ p_j\geq 2;\ k1,k2,\ldots,kp_j\in\{1,2,\ldots,m\}$$

$$\text{avec } R_j[EC_k]\subset E_k[EC_k] \text{ pour } k=k1,k2,\ldots,kp_j$$

où R_j est le nom d'un TA défini entre $E_{k1},E_{k2},\ldots,E_{kp_j}$, RNC_j un ensemble d'attributs sur R_j, éventuellement vide, et $EC_{k1}EC_{k2}\ldots EC_{kp_j}$ l'union des clés des TEs participants. Dans ce schéma on trouve les DFs telles que

$$EC_{k1}EC_{k2}\ldots EC_{kp_j} \rightarrow RNC_j$$

$$\text{pour } j=1,2,\ldots,n;\ p_j\geq 2;\ k1,k2,\ldots,kp_j\in\{1,2,\ldots,m\} \quad ---(ii).$$

S'il existe une DFE déduite à partir des cardinalités des TEs participant à un TA R_j, on la traduit comme une DF dans le contexte du schéma de relation R_j; étant donné $U_j=EC_{k1}EC_{k2}...EC_{kp_j}$, et si un seul TE $E_k \in \{E_{k1},E_{k2},...,E_{kp_j}\}$ porte la cardinalité 1, alors

$$U_j\text{-}EC_k \rightarrow EC_k \quad \text{---(iii).}$$

Puisque $U_j\text{-}EC_k \rightarrow U_j$ (par règle d'augmentation) et $U_j \rightarrow RNC_j$ (la DF de (ii) pour la relation R_j), on peut déduire la DF:

$$U_j\text{-}EC_k \rightarrow RNC_j \quad \text{---(iv).}$$

Remarquons que RNC_j n'est pas transitivement dépendant de $U_j\text{-}EC_k$ à cause de la DF: $U_j \rightarrow U_j\text{-}E_k$ (DF triviale), qui ne satisfait pas la troisième condition de la définition de dépendance transitive (voir le CHAPITRE I, le sous-paragraphe 3.1.1.). A partir de (iii) et (iv), le schéma de relation pour R_j est modifié, en gardant la 3FN comme $(R_j[U_j,RNC_j],\{U_j\text{-}EC_k\})$. Cette analyse est effectuée à l'étape (3-1), au cours de la transformation.

Si dans un TA R_j, plus d'un TE participant porte la cardinalité 1, on peut traduire ce fait par des DFs telles que: étant donné $U_j=EC_{k1}EC_{k2}...EC_{kp_j}$ et $X_j \subset U_j$, où, pour tout $EC_k \in X_j$, le TE E_k porte la cardinalité 1, alors

$$U_j\text{-}EC_k \rightarrow EC_k \text{ pour tout } EC_k \in X_j \quad \text{---(v).}$$

A partir de $U_j \rightarrow RNC_j$ et (v), on peut déduire les DFs:

$$U_j\text{-}EC_k \rightarrow RNC_j,EC_k \text{ pour tout } EC_k \in X_j \text{ --- (vi).}$$

Les DFs de (vi) donnent la raison de modifier le schéma de relation R_j à l'étape (3-2) comme $(R_j[U_j,RNC_j],\{U_j\text{-}E_k | E_k \in X_j\})$. Chaque $U_j\text{-}E_k$ représente une clé candidate de la relation R_j.

A l'étape (4), pour un schéma de relation représentant un TA R_j: $(R_j[EC_{k1}EC_{k2}...EC_{kp_j},RNC_j],\{RC_1,RC_2,...,RC_r\})$, $r\geq 1$, où $\{RC_1,RC_2,...,RC_r\}$ est un ensemble de clés construites aux étapes précédentes, quand un TE E_i, $i\in\{k1,k2,...,kp_j\}$ est obligatoire, cela se traduit comme une DI:

$$E_i[EC_i]\subset R_j[EC_i].$$

Finalement dans le schéma relationnel obtenu aux étapes précédentes, s'il y a deux schémas de relation

$$(R[U_r,RNC],\{\underline{RC}_1,RC_2,...,RC_r\}),\ r\geq 1$$
$$(S[U_s,SNC],\{\underline{SC}_1,SC_2,...,SC_s\}),\ s\geq 1$$

où U_r (respectivement U_s) est un ensemble d'attributs constituant l'ensemble de clés candidates $\{RC_1,RC_2,...,RC_r\}$ (respectivement $\{SC_1,SC_2,...,SC_s\}$) de R (respectivement S) et que RNC (respectivement SNC) est un ensemble d'attributs non-clés éventuellement vide de R (respectivement S), et si R et S satisfont les deux conditions:

(a) il existe i, j tels que $RC_i=SC_j$, pour $i\in\{1,2,...,r\}$, $j\in\{1,2,...,s\}$

(b) $R[RC_i]\subset S[SC_j]$ et $S[SC_j]\subset R[RC_i]$

alors à l'étape (5) on fusionne R et S, et on obtient le schéma de relation:

$$(R_S[U_{r_s},RNC,SNC],\{\underline{RC}_i,RC_1,..,RC_{i-1},RC_{i+1},..,RC_r,SC_1,..,SC_{j-1},SC_{j+1},..,SC_s\})$$

où U_{r_s} est $U_r \cup U_s$.

A partir de la condition (a), on peut déduire trivialement la DF:

$$RC_i \rightarrow U_s,SNC \quad ---(vii).$$

Et aussi, puisque $RC_p \rightarrow RC_i$ pour une clé RC_p, $p \in \{1,2,\ldots,i,i+1,\ldots,r\}$, alors

$$RC_p \rightarrow U_s, SNC \quad ---(viii).$$

Pourtant, (viii) n'est pas une dépendance transitive à cause de la DF: $RC_i \rightarrow RC_p$. De même pour toute clé SC_q, $q \in \{1,2,\ldots j\ldots,s\}$, on déduit la DF:

$$SC_q \rightarrow U_r, RNC \quad ---(ix).$$

De plus la condition (b) garantit la fusion des deux schémas de relation, R et S, parce qu'elle représente le fait que l'ensemble de valeurs de clé de R et celui de S sont identiques. C'est à dire que pour tout tuple u de R, il existe toujours un tuple v de S tel que $u[RC_i]=v[SC_j]$ et réciproquement.

Le schéma de relation R_S obtenu est en 3FN parce que tous les attributs non clés RNC SNC sont directement dépendants des clés (vérifié par (vii), (viii) et (ix)). Il est aussi en 3FNBC, s'il n'y a pas de cas où un attribut non-clé ou un groupe d'attributs non-clés détermine fonctionnellement une partie de la clé dans un TE, parce que chaque fois que $X \rightarrow A_i$ pour $A_i \notin X$ est verifié, alors X contient une clé dans certain $(R[A_1,A_2,\ldots,A_n],\{K_1,K_2,\ldots,K_r\})$. Ce résultat est dû à la notion de clé candidate et à la condition dans laquelle des TEs et des TAs soient bien définis.

4. Réalisation

Nous avons réalisé les deux prototypes sur la machine VAX750 en CEYX [HULLOT84] (une extension du langage LE_LISP [CHAILLOUX84]): un système interactif de construction des schémas E-A (nommé "ea") et un système de transformation d'un schéma E-A en schéma relationnel (nommé "trn"). Ils sont également opérationnels sur la machine SM90 et nous sommes en train de développer une interface graphique sur la même machine [TOUZANI85]. La présentation de ces systèmes va suivre la forme de la documentation d'UNIX.

4.1. Système interactif de construction des schémas E-A

NOM

ea - système de construction des schémas E-A

SYNOPSIS

(ea) sous CEYX

DESCRIPTION

Ce système aide à construire et à modifier des schémas E-A de manière interactive. Au niveau de l'interface utilisateur, il y a trois catégories de fonctions:

(1) fonctions utilitaires;
(2) fonctions de création et de manipulation des schémas globaux E-A;
(3) fonctions de manipulations des schémas détaillés E-A.

Nous allons les présenter en suivant l'ordre.

Pour appeler le système, il faut d'abord être sous CEYX (il suffit de taper "ceyx" sous UNIX). Ensuite, on doit invoquer un module qui charge tous les modules nécessaires et qui exécute

l'initialisation du système par la commande LE_LISP:

(load /usr/users/sgbd/kim/ea/ea.ll)

Le schéma E-A construit à l'aide du système peut être transformé automatiquement en schéma relationnel par le système de transformation "trn". Le système de transformation peut être directement chargé sous mode "ea" par l'appel de la fonction "(appel-trn)".

FICHIER

/usr/users/sgbd/kim/ea

VOIR AUSSI

trn(1), ceyx, le_lisp

REQUETES

(1) FONCTIONS UTILITAIRES

(**debut-ea**)	-> débuter la session
(**fin-ea**)	-> finir la session
bib-nomsch	-> liste de noms de schéma E-A déjà existants
(**menu-ea**)	-> voir le menu de fonctions
(**doc-ea**)	-> voir le document du système "ea"
(**appel-sch** sch)	-> charger le schéma E-A
sch-charge	-> liste de noms de schéma E-A chargé
(**chois-sch** sch)	-> choisir un schéma E-A pour définir le contexte de travail. C'est une fonction primordiale pour les fonctions de manipulation de schéma détaillé.
sch-courant	-> le nom du schéma E-A courant
*	-> le nom du schéma E-A courant

[REMARQUE]

On peut utiliser "*" pour désigner le schéma E-A courant. Ce symbole évitera de taper le nom complet du schéma courant, qui est choisi par la fonction "chois-sch".

(**sauve-sch** sch) -> sauvegarder le schéma E-A dans un fichier postfixé par ".ea.ll"

(**copie-sch** sch n-sch) -> copier un schéma E-A dans un nouveau schéma

(**suppr-sch** sch) -> supprimer définitivement un schéma E-A
ATTENTION à l'utilisation

(**affic-sch** sch) -> afficher le schéma E-A sur l'écran

(**affic-te** sch te) -> afficher le schéma détaillé d'un TE du schéma E-A sur l'écran

(**affic-ta** sch ta) -> afficher le schéma détaillé d'un TA du schéma E-A sur l'écran

(**impri-sch** sch) -> imprimer le schéma E-A dans un fichier postfixé par ".lpr"

(**renom-sch** sch nouveau-nom)
-> renommer un schéma E-A par un nouveau nom donné

(**renom-te** sch te nouveau-nom)
-> renommer un TE du schéma E-A par un nouveau nom

(**renom-ta** sch ta nouveau-nom)
-> renommer un TA du schéma E-A par un nouveau nom

(**appel-trn**) -> appeler le système de transformation "trn" d'un schéma E-A en schéma relationnel. Cette fonction va charger automatiquement le système de transformation.

(2) FONCTIONS DU SCHEMA GLOBAL

(**creer-sch** nom-sch) -> sch ; créer un nouveau schéma

(**est-vide** sch) -> Bool ; tester si un schéma est vide

(**ajout-te** sch te) -> sch ; ajouter un TE dans un schéma E-A

(**ajout-ta** sch ta te1 te2 ... teN) -> sch

```
                                        ; ajouter un TA défini entre des TEs
                                          dans un schéma E-A
(avoir-te sch te)              -> Bool ; tester s'il existe un TE dans un
                                          schéma E-A
(avoir-ta sch ta)              -> Bool ; tester s'il existe un TA dans un
                                          schéma E-A
(suppr-te sch te)              -> sch  ; supprimer un TE d'un schéma E-A
(suppr-ta sch ta)              -> sch  ; supprimer un TA d'un schéma E-A
(divis-te sch te te1 te2)      -> sch  ; diviser un TE en deux TEs donnés
(divis-ta sch ta ta1 ta2)      -> sch  ; diviser un TA en deux TAs donnés
(fusio-te sch te1 te2 te)      -> sch  ; fusionner deux TEs en un TE
(fusio-ta sch ta1 ta2 ta)      -> sch  ; fusionner deux TAs en un TA
```

(3) FONCTIONS DU SCHEMA DETAILLé

[REMARQUE]

Les fonctions suivantes doivent être précédées par l'appel de fonction "chois-sch" pour choisir un schéma courant. "typ" désigne un TE ou un TA.

```
(ajout-att typ att dom)          -> typ ; ajouter un attribut
(suppr-att typ att)              -> typ ; supprimer un attribut
(rempl-dom typ att nouveau-dom)  -> typ ; remplacer un domaine
(renom-att typ att nouveau-nom)  -> typ ; renommer un attribut
(divis-att typ att att1 att2)    -> typ ; diviser un attribut
(fusio-att typ att1 att2 att)    -> typ ; fusionner deux attributs
(defin-cle te lst-att)           -> te  ; définir la clef
(defin-cle-p te lst-att)         -> te  ; définir la clef principale
(suppr-cle te lst-att)           -> te  ; supprimer la clef
(card ta te M/1)        -> ta  ; affecter la cardinalité du TE dans un TA.
                                 M par défaut.
(oblig ta te F/O)       -> ta  ; affecter l'obligation du TE dans un TA.
                                 F(acultatif)/O(bligatoire), F par défaut.
```

[NOTE] Les abréviations des fonctions sont définies pour éviter de taper le nom complet d'une fonction. Elles seront visualisées par l'appel de fonction "(menu-ea)".

4.2. Système de transformation d'un schéma Entité-Association en schéma relationnel

NOM

trn - système de transformation d'un schéma E-A en schéma relationnel

SYSNOPSIS

(trn) sous CEYX

DESCRIPTION

Ce système transforme un schéma E-A construit par le système "ea" en schéma relationnel. Le schéma relationnel engendré sera en troisième forme normale. A partir du schéma obtenu, on peut engendrer un programme de la déclaration d'une base de données soit sous forme de R-LTR, soit sous forme de l'EQUEL.

Pour appeler le système, il faut d'abord être sous CEYX. Ensuite, on peut le charger directement par la fonction LE_LISP:

(load /usr/users/sgbd/kim/trn/trn.ll)

Et aussi, on peut l'appeler sous "ea" (système de construction des schémas E-A) par la fonction fournie:

(appel-trn) ou bien (aptrn)

La dernière est plus conseillée parce que, effectivement, on a besoin des fonctions de "ea" pour consulter et manipuler des schémas E-A pour la transformation.

FICHIER

/usr/users/sgbd/kim/trn

VOIR AUSSI

ea(1), ceyx, le_lisp, ingres

REQUETES

(debut-trn) -> débuter la session de la transformation
(fin-trn) -> finir la session

(menu-trn) -> voir le menu de fonctions
(doc-trn) -> voir le document du système "trn"

(appel-sch-rel nom-sch-rel)
-> charger le schéma relationnel
(sauve-sch-rel nom-sch-rel)
-> sauvegarder le schéma relationnel dans un fichier
postfixé par ".rel.ll"
(copie-sch-rel sch-rel nouveau-sch)
-> copier le schéma relationnel dans un nouveau schéma
(suppr-sch-rel sch-rel)
-> supprimer le schéma relationnel

(trans nom-sch-ea nom-sch-rel) -> sch-rel
; transformer un schéma E-A en un schéma relationnel

(affic-sch-rel nom-sch-rel)
-> afficher le schéma relationnel sur l'écran
(impri-sch-rel nom-sch-rel)
-> imprimer le schéma relationnel dans un fichier
postfixé par ".lpr"
(impri-sch-ltr nom-sch-rel)
-> imprimer le schéma relationnel sous forme R-LTR
dans un fichier postfixé par ".LTR"

(**impri-sch-igr** nom-sch-rel)

-> imprimer le schéma relationnel sous forme EQUEL dans un fichier postfixé par ".q"

BUGS

Pour avoir un résultat souhaitable d'un programme engendré EQUEL, il faut respecter les règles suivantes issues du système INGRES.

(1) Le type de domaine doit être un des suivants:
iX (type entier de X octets, X=1,2,4)
fX (type réel de X octets, X=4,8)
cX (type chaîne de caractères, X=1,...,255)

(2) Chaque identificateur est une séquence de caractères alphanumériques dont la première est une lettre alphabétique. La taille maximum est 12. Le symbole "_" (souligne) considéré dans l'alphabet. Des lettres majuscules seront automatiquement transformées en miniscules.

(3) Dans une relation, on ne peut pas avoir plus de 49 attributs.

[NOTE] Les abréviations des fonctions sont définies de façon à éviter de taper le nom complet d'une fonction. Elles seront visualisées par l'appel de fonction "(menu-trn)".

5. Conclusion

Le modèle E-A est un modèle très adéquat, quoique informel (ce qui est parfois un avantage)†, pour décrire la structure d'un certain univers où on est ramené à utiliser une BD; les concepts utilisés étant simples à appliquer. Le modèle relationnel étant le plus apte à refléter des structures Entité-Association (cf. [CODD79]), nous avons présenté dans ce chapitre une technique pour définir un schéma relationnel, s'appuyant sur une analyse préalable en termes d'entités et d'associations.

Le concepteur se place au niveau des objets réels, les entités, à représenter dans la base, et il les décrit par des données plus élémentaires, attributs, et par les liens qui les relient à d'autres objets. L'analyse des dépendances entres les données se trouve ainsi localisée à un TE ou a un TA, ce qui d'une part, en réduit la complexité et, d'autre part, permet de distinguer les dépendances, entre données identiques, qui ont des sémantiques différentes. Le schéma relationnel ainsi obtenu est plus proche de la réalité, dans le sens où les entités et les relations perçues sont plus apparentes

Notre approche se différencie ainsi de l'approche classique [BERNSTEIN76] [DELOBEL82] [AHO79] [RISSANEN77] où un schéma relationnel est obtenu par un processus de normalisation. L'objectif poursuivi par l'approche classique est la réduction des redondances des informations, afin d'éviter les anomalies de mise à jour. Ici, les éléments de base sont les données élémentaires, les attributs, et l'idée fondamentale est de les collectionner et d'observer les dépendances pouvant exister entre eux. Il suffit ensuite d'appliquer les algorithmes appropriés pour obtenir un schéma relationnel normalisé.

L'avantage de cette approche, c'est ses bases théoriques bien établies: dépendance fonctionnelle, dépendance multivaluée, dépendance

(†) Un des buts des modèles sémantiques est de permettre une flexibilité dans l'interprétation du monde réel [DATE83].

de jointure etc.. Néanmoins, comme l'on s'intéresse aux attributs, les dépendances entre ces derniers peuvent se révéler très complexes à saisir, et il y a risque parfois de confondre plusieurs rôles spécifiques d'un même attribut [DATE81]. Dans [FOUCAUT84], l'auteur propose d'utiliser un modèle E-A pour obtenir les attributs (qu'il suffit d'inventorier) et les dépendances fonctionnelles, pour ensuite appliquer une méthode de synthèse d'un schéma relationnel.

Un algorithme de transformation d'un diagramme E-A en un schéma relationnel contenant des contraintes d'intégrité de DI est proposé. L'intérêt des DIs dans un schéma relationnel, c'est qu'elles garantissent la cohérence de données contre des anomalies en cours de mise à jour. Nous avons analysé les clés du résultat de la transformation pour montrer que le résultat est en troisième forme normale. Cette approche fournit un guide methodologique pour concevoir et réaliser une application BD selon le modèle relationnel.

Après avoir mis au point la représentation et la transformation des types génériques d'association (proposés dans la section 5 du CHAPITRE III) en un schéma relationnel, la méthode et les systèmes implantés seront complétés.

Nous envisageons une application directe de la méthode proposée pour construire une base de données relationnelle qui peut être gérée par un système de gestion de base de données existant. Parallèlement un diagramme E-A peut servir de document complet pour la base de données relationnnelle si elle est construite par la méthode. Une autre application constituant notre préoccupation est d'utiliser le modèle E-A pour représenter les objets dans un Système de Gestion d'Objets [EMERAUDE84] [DERNIAME85] [GODART86], réalisant les fonctions communes au différents ateliers de développement de logiciels. Le système de gestion d'objets est chargé de maintenir, de façon cohérente, des informations utiles au développement de logiciel. La description des objets par le schéma E-A dans un système de gestion d'objets peut être systématiquement transformée, à l'aide de notre travail présenté dans ce chapitre, en un schéma relationnel qui est un schéma global logique d'un SGBD relationnel.

CHAPITRE V - PROPOSITION D'UN SYSTEME DE GESTION D'OBJETS

Résumé

Un logiciel est un vaste ensemble d'objets de types différents qui peuvent être issus de différentes étapes de son cycle de vie. Nous proposons de structurer et stocker cet ensemble d'objets en une base de données que nous appelons Base d'Objets. Les objets de cette base sont gérés par une collection d'opérations qui constitue le Système de Gestion d'Objets. Nos objectifs sont l'étude et le développement d'une structure d'accueil basée sur un système de gestion d'objets en vue de la construction des Ateliers Logiciels. Nous utilisons le modèle Entité-Association comme schéma conceptuel utilisé par le système de gestion d'objets.

Sommaire

1. Introduction
2. Une structure d'accueil: Système de gestion d'objets
   2.1. Choix du modèle conceptuel Entité-Association
      2.1.1. Critères de choix
      2.1.2. Concepts du modèle dans le cadre du système de gestion d'objets
   2.2. Fonctions du système de gestion d'objets
      2.2.1. Fonction de description
      2.2.2. Fonction de manipulation
      2.2.3. Fonction de maintien de l'intégrité
      2.3.4. Fonction de stockage
      2.4.5. Fonction de définition de vue
      2.4.6. Fonction d'interfaçage
   2.3. Caractéristiques du système de gestion d'objets
3. Présentation des travaux en cours
   3.1. Description des connaissances d'un atelier logiciel
   3.2. Etude sur le maintien de l'intégrité
   3.3. Interfaçage utilisateur
4. Conclusion

CHAPITRE V
PROPOSITION D'UN SYSTEME DE GESTION D'OBJETS

1. Introduction

Autrefois, le développement d'un logiciel pouvait se faire de manière artisanale, basée sur l'habileté personnelle. Les outils informatiques utilisés sont simples et destinés à un but précis; par exemple, éditeur de texte, compilateur, etc.. Une phrase qui reflète cette situation se trouve dans le préface de [KNUTH73].

> Le processus de la préparation des programmes pour l'ordinateur est spécialement attirant, non seulement parce qu'il peut être profitable économiquement et scientifiquement, mais aussi qu'il peut être une expérience esthétique autant que la composition d'un poème ou d'une musique.

Cette ère a été de courte durée, car les choses ont évolué très vite. Deux facteurs essentiels sont à l'origine de cette évolution dans le domaine de programmation: la grande taille des logiciels et leur complexité devenant incontrôlable par l'esprit artisanal.

On a vu ainsi naître l'idée d'un Atelier Logiciel[†] qui se préoccupe, à notre avis, des quatre points importants:

- les disciplines de développement d'un logiciel, par exemple, cycle de vie [BOEHM84], prototypage [BLUM82] [McCRACKEN82], etc.;
- les outils informatiques sophistiqués et intéractifs, par exemple, interprète, débogueur ("debugger"), éditeur syntaxique, poste autonome de travail, etc.;

(†) Ici le terme "atelier logiciel" doit être compris comme un terme générique, non pas comme un système (ou environnement) de développement (ou production) de logiciel, ce dernier est le cas pour [BOURGEOIS85] et [KRAKOWIAK82].

- l'organisation de l'équipe de travail, par exemple, programmeur en chef ("chief programer"), gestionnaire de haut niveau ("high level manager"), groupe aux égaux ("group of peers") [ARON85];

- la gestion du projet, par exemple, communication entre des intervenants, conduite et suivi de projets, documentation, etc..

Dans la littérature, on trouve les synthèses de divers atliers logiciels. Krakowiak a clarifié, dans son article [KRAKOWIAK82], les concepts liés aux systèmes d'aide au développement de logiciel selon deux critères: modularité et gestion des versions, qui lui paraissaîent importants pour juger de la qualité d'un tel système. Dans [DE DROUAS82], les auteurs ont essayé de classifier des ateliers logiciels français selon les étapes chronologiques d'évolution. Dans l'ouvrage [BARSTOW85], on trouve un ensemble d'ateliers logiciels opérationnels, chacun de ces atelierss est relativement bien présenté par le(s) participant(s) au projet. La première revue "Génie Logiciel, No.1" de l'ADI (Agence De l'Informatique), nous offre une présentation assez complète d'un ensemble d'ateliers logiciels réalisés et en projet. On trouve, notamment dans [BOURGEOIS85], une classification des ateliers logiciels selon les concepts: cycle de vie, méthode, langage, boîte à outils et prototypage.

Nous dégageons simplement un point de vue à l'égard des ateliers logiciels en mettant l'accent sur leur objectif principal:

- Boîte à outils visant à l'intégration cohérente des outils, ce qui favorise la production d'un logiciel, par exemple, PWB/UNIX [DOLOTTA78], SOL [SYSECA84], VULCAIN [FRAVAL85], etc.;

- Système d'aide à la conception, généralement lié à une méthode, par exemple, SPRAC (spécification formelle) [FOISSEAU85], SIGL (méthode SADT) [COHEN83], SSP (machine abstraite) [CHERBONNEAU85], etc.;

- Système d'aide au développement, généralement lié à un langage de programmation, par exemple, APSE (ADA) [WOLF81], INTERLISP (LISP) [TEITELMAN81], Atelier LTR (LTR-3) [CELAR83], etc.;

- Système d'aide à la gestion de projet, par exemple, Base de connaissances logiciels ("Software Knowledge Base") [MEYER85-a].

D'abord, nous attirons l'attention sur l'utilisation d'une base de données pour stocker et manipuler des informations au sein d'un atelier logiciel (ex. SPRAC <Base de données Projet du type relationnel>, VULCAIN <SGBD Ingres>, SSP <BD relationnelle>, etc.). Ensuite, nous pensons qu'il n'y a pas de consensus autour d'une méthode ou autour d'un langage de programmation. Enfin, nous remarquons qu'il y a peu (pratiquement pas) de atelier logiciel qui couvre les différentes phases du cycle de vie d'un logiciel: Analyse de besoins, Conception, Programmation, Test-Intégration et Maintenance.

Une solution souhaitable est de réaliser un atelier permettant de couvrir toutes les étapes de cycle de vie sans imposer une méthode spécifique et un langage unique. Des efforts ont été déployés dans ce sens par le PNGL (Projet National de Génie Logiciel) avec le projet EMERAUDE [EMERAUDE84] qui propose une Structure d'Accueil pouvant servir à toute personne désirant réaliser un atelier logiciel.

Notre proposition suit le même axe que celui de EMERAUDE. Les originalités de notre travail résident dans (i) l'effort porté sur le maintien de la cohérence des objets gérés, parce que nous pensons que la cohérence des objets peut garantir une meilleure qualité du logiciel produit, (ii) l'utilisation de l'approche base de données, et plus précisément l'élaboration du schéma conceptuel Entité-Association et l'exploitation d'un SGBD relationnel.

2. Une structure d'accueil: Système de gestion d'objets

Commençons par donner quelques définitions.

Atelier logiciel

Nous définissons un **atelier logiciel** comme un système intégré d'aide à la conception, au développement de logiciels ainsi qu'à la gestion de projets. C'est un environnement composé d'un ensemble d'outils coexistants dans le but d'une production plus rapide et cohérente de logiciels. Ces outils peuvent intervenir tout au long du cycle de vie d'un logiciel. Un atelier est plus ou moins adapté à une méthode d'analyse et de programmation et à un ou plusieurs langages.

Structure d'accueil

Une **structure d'accueil** est un noyau permettant la définition d'un nouvel atelier ou l'intégration de nouveaux outils à un atelier existant. La structure d'accueil est développée indépendamment de toute méthode de conception, de spécification, de langage, etc.. Elle doit permettre d'assurer la gestion des objets spécifiques à toute méthode ou tout langage. Ces objets sont gérés par un système de gestion d'objets. Elle diffère en celà des boites à outils.

Système de gestion d'objets

Le **Système de Gestion d'Objets** (noté SGO) est le coeur de la structure d'accueil. Il permet:

- la description des objets et des associations entre ces objets. Les objets sont issus de différentes étapes du cycle de vie et sont liés soit au sein d'une même étape, soit entre deux étapes;
- la manipulation (création, accès, modification, stockage, etc.) des objets. Ces objets sont structurés et stockés dans une **base d'objets** (une collection structurée d'objets);
- le maintien de l'intégrité de la base d'objets en définissant des associations entre objets pour exprimer la sémantique de la base d'objets.

2.1. Choix du modèle conceptuel Entité-Association

Pour briser la barrière de complexité, la première ligne d'attaque est une représentation uniforme des connaissances et une modularité forte. La complexité peut être contrôlée en trouvant un moyen simple et uniforme de la représentation des connaissances [WINOGRAD73]. Notre proposition n'est pas exceptionnelle dans la mesure où on doit commençer par étudier un modèle conceptuel qui sert à décrire (représenter) des problèmes.

2.1.1. Critères de choix

Le modèle conceptuel choisi pour décrire et manipuler des objets doit être le plus proche possible de la vision, que peut avoir un informaticien (gestionnaire ou programmeur, éventuellement utilisateur final), de l'ensemble des objets qu'il manipule habituellement. Le modèle conceptuel devra en outre avoir les caractéristiques d'un bon modèle conceptuel qui sont les suivants:

- Puissance de représentation: représentation à la fois des aspects structurels et sémantiques du monde réel;

- Evolutivité: L'ajout d'une nouvelle régle sémantique ne doit pas entraîner une modification sensible du schéma. L'introduction d'un nouvel objet ou d'une nouvelle association ne doit pas remettre en cause tout le schéma précédemment construit;

- Simplicité: Le modèle doit être à la fois facile à comprendre et facile à utiliser. Pour cela, le nombre d'éléments constituant le modèle doit être réduit tout en dissociant clairement des concepts différents;

- Base formelle: Le modèle doit avoir une base rigoureuse et formelle pour éviter toute ambiguité d'application et d'interprétation et pour permettre éventuellement une vérification formelle.

Nous choisissons le modèle Entité-Association ayant un sous-ensemble des caractéristiques mentionnées ci-dessus: simplicité et évolutivité, puis éventuellement puissance de représentation. Moyennant les extensions proposées dans le CHAPITRE III, un tel modèle pourra prendre en compte tous nos besoins pour la construction d'un SGO.

2.1.2. Concepts du modèle dans le cadre du système de gestion d'objets

Pour décrire et représenter des connaissances issus d'un atelier logiciel, nous introduisons quatre concepts de base: objet, association, attribut et action [GODART86].

Objet et type d'objet

Les objets qu'on doit manipuler par un SGO sont des composants d'un logiciel et des intervenants dans le développement d'un logiciel. Un type d'objet regroupe un ensemble d'objets ayant les mêmes caractéristiques. Les types d'objet sont très variés dont voici quelques exemples:

- PROGRAMME_SOURCE,
- CODE_OBJET,
- DOCUMENTS_DE_SPECIFICATION,
- PROGRAMMEUR,
- ETAT_CIVIL_D_UN_UTILISATEUR,
- etc.

Nous considérons que le terme "objet" correspond au terme "entité" du modèle Entité-Association dans le cadre du système de gestion d'objets. Il est donc possible d'interchanger les deux dans le propos qui suit.

Association et type d'association

Saisir et manipuler des associations entre objets sont des opérations très importantes parce qu'elles expriment des faits aussi significatifs que ceux exprimés par des objets. L'association est d'abord un élément de base d'un chemin d'accès d'un objet à l'autre, et ensuite une contrainte que des objets doivent vérifier. Un type d'association représente un ensemble d'associations définies entre les mêmes types d'objet et ayant les mêmes propriétés. Voici quelques exemples de type d'association binaire défini sur les types d'objet donnés ci-dessus:

- COMPILE_DE : un code objet est compilé du programme source correspondant;
- DOCUMENTE_PAR : un programme est documenté par un document de spécification;
- ECRIT_PAR : un programme source est écrit par un programmeur;
- etc.

Un type d'association peut être spécifié par son nom, le degré (le nombre des types d'objet participants), la cardinalité de chaque type d'objet participant et l'obligation de chaque type d'objet participant.

Attribut

Un attribut permet de définir une propriété d'un objet ou d'une association par sa valeur. La valeur d'un attribut doit appartenir à un domaine simple (voir le CHAPITRE I, le sous-paragrahpe 2.1.3.). Par exemple, les attributs du type d'objet PROGRAMME_SOURCE peuvent être les suivants:

- IDENTIFICATEUR: le nom unique du programme source qui permet d'indentifier un objet (une occurrence) du type (clé d'un type d'objet);
- LANGAGE: langage utilisé (ex. PL/1, C, PASCAL, LTR3, ...);
- MODE_PROTECTION: mode de protection (ex. lecture autorisée aux

autres, lecture-écriture autorisée, rien, ...);

- CHEMIN_D_ACCES: le chemin d'accès vers le fichier qui contient effectivement le programme écrit; elle lie le SGO au système d'exploitation;
- DATE_DE_DERNIERE_MJA: la date de dernière mise à jour; cet attribut joue un rôle primordial dans le maintien de la cohérence;
- etc.

Nous soulignons ici trois attributs prédéfinis qu'hériteront des objets du type composant de logiciel: IDENTIFICATEUR, CHEMIN_D_ACCES et DATE_DE_DERNIERE_MAJ.

Il est important de noter que l'objet qu'on manipule par le SGO n'est pas un objet physique (ex. un fichier contenant un programme source), mais c'est une description ou une connaissance d'un objet avec les attributs et les associations liées aux autres objets. Par exemple, un objet du type PROGRAMME_SOURCE est décrit par les attributs: IDENTIFICATEUR, LANGAGE, MODE_PROTECTION, CHEMIN_D_ACCES, DATE_DE_DERNIERE_MAJ, ..., et par les associations COMPILE_DE et ECRIT_PAR. Un objet physique représenté par un type d'objet ne peut être qu'un attribut de ce type d'objet (ex. CHEMIN_D_ACCES au fichier contenant le texte du programme pour le type d'objet PROGRAMME_SOURCE).

Action associée

Généralement, définir une association entre deux ou plusieurs objets signifie établir une dépendance formelle entre eux. En attachant à chaque association une ou plusieurs actions, nous faisons plus qu'établir une simple dépendance. Une action décrit alors l'opération à entreprendre en cas de violation de la contrainte posée par la dépendance, après la mise à jour de l'objet auquel l'association est définie. Par exemple, l'association du type COMPILE_DE est plus que définir le fait "un code objet est le compilé d'un programme source", c'est aussi que de toute modification de l'objet du type PROGRAMME_SOURCE doit résulter une compilation de cet objet de façon à

rendre valide l'objet du type CODE_OBJET. "COMPILER" est alors une action attachée au type d'association COMPILE_DE.

2.2. Fonctions du système de gestion d'objets

Pour atteindre nos objectifs, le SGO doit remplir un certain nombre de fonctions. La première description d'un SGO nommé SGO-CRIN est proposée dans [DERNIAME85], nous la reprenons ici.

2.2.1. Fonction de description

Un SGO doit fournir un langage (Langage de Description d'Objets) capable de caractériser des objets et d'exprimer les associations entre ces objets. Autrement dit, la fonction de description doit permettre de définir des types d'objets et des types d'associations. Les associations significatives entre objets doivent pouvoir être typées afin d'être manipulées. Exemples de type d'association:

- UTILISE: un module LTR-3, M1 peut utiliser un module M2. Une définition intéressante du type d'association UTILISE est donnée par [PARNAS79], comme "nécessite la présence d'une version correcte de".
- REALISE: un corps LTR-3, C1 peut être la réalisation d'une interface I1.
- ECRIT_PAR: le programme source C1 est écrit par le programmeur Dupont.

Un langage interactif est réalisé pour construire des schémas E-A (le CHAPITRE IV, le paragraphe 5.1.). Actuellement un langage graphique est en cours de réalisation après l'étude préalabl faite dans [TOUZANI85].

2.2.2. Fonction de manipulation

Le SGO doit permettre de définir, instancier, modifier et consulter les objets ou les associations de la base d'objets. La fonction de manipulation doit aussi assurer la sécurité (droit d'accès,

confidentialité, protection, ...) surtout le maintien de l'intégrité de la base. Ce sont les types d'association qui expriment la plupart des contraintes d'intégrité. La sémantique du schéma de la base d'objets est en effet représentée par les associations. Par exemple:

- Le type d'association UTILISE indique que "M1 utilise M2" mais exprime aussi la contrainte "si le module M2 est modifié, alors M1 doit être recomplié";

- De même pour le type d'association REALISE, l'association "C1 réalise I1" doit en outre signaler lors d'une modification de I1 que le corps C1 ne correspond peut-être plus à la spécification I1.

2.2.3. Fonction de maintien de l'intégrité

Le SGO doit être capable de déclencher et d'enchaîner les actions permettant le maintien de l'intégrité lorsqu'une contrainte d'intégrité risque d'être violée. Une grande partie des contraintes d'intégrité est exprimée par les associations entre objets. Assurer le maintien de l'intégrité revient à associer au schéma d'objets l'ensemble des dépendances entre objets et à développer un mécanisme de déclenchement et d'enchaînement des actions associées.

La première maquette est expérimentalement réalisée sous UNIX sur VAX750 [BENALI85]. La maquette réalisée en langage C permet de définir des associations en les nommant et en leur attachant des actions sous forme de commandes du Shell UNIX. Le maintien d'intégrité dans le SGO-CRIN est étudié dans [GODART86].

2.2.4. Fonction de stockage

Il s'agit de stocker des objets et des associations entre objets. Certains objets sont partagés entre le SGO et le système d'exploitation; c'est en particulier le cas pour un fichier physique. Ces objets devront donc être stockés en partie dans une structure UNIX. Les autres pourront l'être dans une base de données

certainement de type relationnel. Ces choix sont guidés par notre expérience du modèle E-A et du passage de ce modèle au modèle relationnel ([KIM85,86-b], le CHAPITRE III et le CHAPITRE IV). Par le système de transformation "trn" (voir le CHAPITRE IV, le paragraphe 5.1.), on peut obtenir systématiquement un programme EQUEL, qui crée une base de données sous le SGBD INGRES à partir d'un schéma E-A.

2.2.5. Fonction de définition de vue

Cette fonction rejoint celle de manipulation. Un utilisateur du SGO ne veut pas ou ne doit pas connaître toute la base d'objets. Le SGO doit fournir un mécanisme permettant de restreindre la base d'objets à un sous-ensemble. Une solution est de considérer la base d'objets comme une collection hiérarchisée de bases d'objets. Une base d'objets serait alors une multibase d'objets par analogie avec la multibase de données [GODART81,84] [NASSIF83] [TOUNSI83]. Notons que c'est également la philosophie du système de gestion de fichiers d'UNIX.

2.2.6. Fonction d'interfaçage

La description et la manipulation d'objets sont complexes et nécessitent une aide à l'utilisateur. L'interface utilisateur-SGO sera donc un moyen de communication où l'utilisateur disposera d'un environnement pour apprendre à dialoguer avec le SGO, de façon à pouvoir l'interroger, créer ou modifier le schéma conceptuel de la base d'objets. L'aspect graphique de l'interface est souligné dans le cadre de notre projet.

Le SGO présenté ci-dessus est une couche au dessus du système d'exploitation. L'utilisateur exploitera les objets de la base à l'aide des commandes du SGO. Cependant, le SGO devra être le plus transparent possible à l'utilisateur, il doit donc émuler les fonctions du système d'exploitation.

2.3. Caractéristiques du système de gestion d'objets

Le SGO doit permettre aussi bien le développement d'outils d'aide à la spécification, à la conception que le développement d'outils d'aide au codage s'appuyant sur n'importe quelle méthode ou n'importe quel langage. C'est la condition fondamentale. Nous évoquons quelques caractéristiques de notre SGO.

Pouvoir de description des connaissances de logiciel

Les connaissances s'expriment par le schéma conceptuel E-A. Les objets et les associations entre eux constituent un véritable modèle de logiciel. La sémantique du logiciel est décrite par les association entre objets. A chaque association sera attachée une ou plusieurs actions qui décriront les procédures de maintien de la cohérence. Le maintien de la cohérence nous semble essentiel par rapport à d'autres systèmes proches, tels que EMERAUDE.

Simplicité

Le SGO doit être simple à apprendre et à utiliser. Il doit comporter un module d'apprentissage. Le schéma d'objet doit être simple à manipuler en modifiant au minimum les habitudes de l'utilisateur. Le modèle E-A a un grand avantage issu de sa simplicité conceptuelle.

Couverture de tout le cycle de vie

Le SGO doit permettre le développement d'outils d'aide aux différentes étapes du cycle de vie mais doit en plus assurer la connexion de ces différentes étapes. Il doit permettre d'exprimer des liens entre objets d'étapes différentes. Cela nous avantage par rapport à d'autres systèmes d'aide à la conception ou aux systèmes d'aide au développement.

Intégration du SGO dans l'environnement de base

Nous pensons que toute opération sur des objets de la base d'objets doit être immédiatement répercutée sur cette base. Cela signifie, en particulier, que toute commande doit contenir des primitives adéquates à la modification de la base d'objets. Le SGO doit émuler les commandes du système d'exploitation. Cela nous différencie de la base de connaissances logiciels [MEYER85-a] qui propose, pour des raisons de simplicité, que les transformation d'objets par le système soient répercutées à la main dans la base par les membres du projet.

3. Présentation des travaux en cours

3.1. Description des connaissances d'un atelier logiciel

Il est important au sein d'un atelier, de fournir à l'utilisateur une assistance, en automatisant dans le processus de production les tâches, parfois de routines, qui ne concernent en rien la créativité. Nous avons abordé cette étude dans une approche BD en envisageant une base qui mémorise et met à disposition toute information relative à un projet de développement cohérent et contrôlé d'un logiciel, ainsi que toute autre information relative à sa maintenance et à son suivi. Rappelons qu'un logiciel est produit sous forme de configurations, ensembles de composants logiciels, variant dans le nombre et la nature des composants, leur spécification, ou leurs versions. Sa fabrication fait intervenir deux éléments, la réalisation des composants et leurs connections.

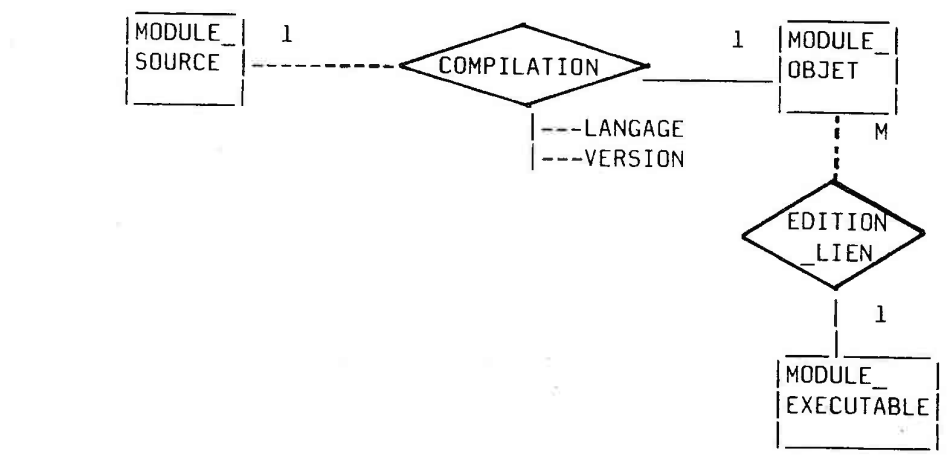
Cela conduit donc à définir pour un atelier logiciel et dans le cadre d'une structure d'accueil, un schéma conceptuel de base de données qui exprime:

- les types d'objets à gérer dans l'atelier, c'est à dire, les composants de programmes dans leurs diverses natures (interface, spécification, module source/objet, document, etc.) et dans leurs différentes versions;
- les types d'association pouvant lier ces objets et qui doivent être établis et maintenus;
- les conditions d'intégrité de ces informations;
- un ensemble d'opérations telles que création/modification d'objets, établissement/maintien de liens entre ces objets;
- un ensemble de requêtes d'interrogations.

Ces deux derniers points étant très dépendants des autres, constituent des points cruciaux, et à ce propos un modèle Entité-Association s'avère d'être une grande portée. Dans cette optique, un SGO est un système qui gravite autour de la BD et qui fournit les moyens de décrire une telle structure (schéma conceptuel) et de manipuler les objets et leurs associations en assurant le maintien de l'intégrité.

Pour le type d'application envisagé, on peut représenter certaines connaissances d'un atelier logiciel, celles en vigueur en génie logiciel, par les types prédéfinis d'objet et d'association dans le SGO. Par exemple, si l'on considère un composant comme l'unité manipulable par un outil de l'atelier, ces unités seront des objets de base dont le type varie selon l'outil considéré. Pour un compilateur, ce sera par exemple des objets de type MODULE_SOURCE ou MODULE_OBJET, et pour un éditeur de liens, des objets de type MODULE_OBJET ou MODULE_EXECUTABLE.

Des types d'association de base peuvent aussi être déterminés par les outils. Il y a une association de type COMPILATION entre un module source et un module objet, (et qui peut être caractérisée par la VERSION et le LANGAGE du compilateur) comme s'il y a une association de type EDITION_LIEN entre des modules objets et un module exécutable. Ces types d'objet et d'association peuvent être schématisés par le diagramme E-A suivant:



Avec ces types prédéfinis d'objet et d'association, un ensemble d'opérations peut être prédéfini dans le SGO tel que:

- DETERMINER les composants incorrects, suite à un changement de version d'un compilateur, les RECOMPILER éventuellement;
- CHERCHER les modules exécutables défectueux car une erreur a

été détectée dans un module source;

- CONSTITUER une configuration avec la nouvelle version de certains composants;

- etc.

En général, le typage des objets et des associations prédéfinis se fera par référence au cycle de vie du logiciel. Une réflexion générale sur les types d'association se trouve dans [BRACHMAN83], tandis qu'une classification des types d'association en vigueur en génie logiciel est proposée par [MEYER85-a]. Le SGO étant développé indépendamment de toute méthode et de tout langage, les associations et les objets de types prédéfinis ne doivent pas être les seuls à la disposition de l'utilisateur du SGO. Il doit pouvoir définir d'autres types d'association et d'autres types d'objet.

D'autres caractéristiques de l'atelier logiciel pouvant être envisagées, sont basées sur les besoins exprimés par les différents utilisateurs de l'atelier, et ont trait par exemple à la gestion du projet, au contrôle de son déroulement, aux interactions entre les membres du projet etc..

3.2. Etude sur le maintien de l'intégrité

Le maintien de l'intégrité est un point essentiel dans notre approche. Le maintien de l'intégrité se fera grâce à la nature des associations entre objets (type d'association). En effet, tout type d'association représente une sémantique de la base d'objets et le SGO doit être conçu de manière à ce que cette sémantique puisse être verifiée à tout moment. Le schéma de la base d'objets sera donc compilé en une représentation interne, un fichier de dépendance exploitable pour la procédure de maintien de l'intégrité.

Dans un premier temps, nous avons realisé cette procédure de maintien de l'intégrité autour du Make d'UNIX ([FELDMAN78]) [BENALI85]. Ultérieurement, nous comptons étendre cette procédure en y intégrant des conditions de déclenchement d'actions autre que le temps de dernière modification. Un mécanisme du maintien de

l'intégrité est proposé par [GODART86] et nous trouvons une solution relationnelle pour le processeur de la dynamique qui assure automatiquement la gestion de l'évolution dans un système d'information [THIERY85]. Nous n'excluons pas d'étudier le mécanisme de "trigger" [ASTRAHAN76] en conjonction avec une technique de déduction abordée par les bases de données déductives.

3.3. Interfaçage utilisateur

La conception de l'interface utilisateur pour le systéme de gestion d'objets est guidée par la réalisation d'un prototype d'interface pour le modèle Entité-Association. L'objectif fixé, à court terme, est de faire une interface du système de transformation d'un schéma E-A en schéma relationnel. Cette interface se base sur les points suivants:

- description des éléments intervenant dans le modèle E-A (ex. TE, TA, attributs, cardinalité, etc.);
- définition des fonctions de manipulation de ces éléments;
- gestion globale des schémas E-A.

L'approche adoptée pour la réalisation de l'interface utilise les techniques de multi-fenêtrage sur un écran de type "Bitmap" pour l'affichage des objets graphiques et des dispositifs de sélection tels que la souris. (le développement est effectué sur une machine SM90 munie du Bitmap Numelec).

Dans un second temps, nous exploiterons cette interface pour développer une interface utilisateur du SGO. En effet, les concepts fondamentaux utilisés dans le SGO reposent, au niveau de l'utilisation, sur les concepts d'objet, d'associations entre objets et d'actions associées. Ceci nous conduira à définir un langage d'interaction qui s'appuie sur la représentation graphique de schéma d'objets et leur manipulation. D'autre part, on cherchera à associer à chaque objet la notion de "document" vis-à-vis de l'utilisateur. Un document est une entité syntaxique manipulée par l'interface et représente la structure physique d'un objet (le contenu d'un objet).

Le but est de rendre indépendant l'interface utilisateur de l'application en cours et des outils utilisés.

Une autre approche pour l'interface utilisateur-SGO est basée sur l'approche édition syntaxique, dans le sens où le modèle E-A associé au SGO sera représenté par une grammaire, notamment une grammaire attribuée, et où l'utilisateur sera guidé par un ensemble de fonctions et de commandes tout au long du processus de communication avec le SGO [CRUZLARA86].

Un autre composant essentiel de l'interface est la structure du dialogue avec le système d'exploitation pour le mécanisme de déclenchement des actions lorsqu'une contrainte d'intégrité risque d'être violée. Ces actions peuvent être des opérations du système d'exploitation. D'où le besoin de définir un langage de commande propre à l'interface elle-même. Par exemple, lors d'une session, l'utilisateur aura besoin d'utiliser plusieurs outils pour développer une application spécifique. Dans un système d'exploitation classique, il définit lui-même la structure d'enchaînement des commandes du système. Cette structure appartient, de manière informelle, aux connaissances de l'utilisateur. Cela implique que l'utilisateur connaisse a priori la réaction du système en fonction des commandes. Pour définir une interface uniforme, il est nécessaire que cette connaissance soit formalisée et intégrée dans l'interface.

4. Conclusion

Nous envisageons deux grandes classes d'utilisation du système de gestion d'objets: développement de logiciels et gestion de projets.

En développement de logiciels, la base d'objets doit être la plus transparente possible. Celle-ci ne doit pas changer fondamentalement les habitudes des programmeurs. Pour cela, le système de gestion d'objets doit permettre une utilisation des objets, qui cache la structuration des objets, de même qu'un système de gestion de fichiers permet l'utilisation des fichiers indépendamment de leur structuration physique et des algorithmes d'accès.

L'utilisation du système de gestion d'objets en gestion de projets consiste d'abord à déterminer le modèle du projet à l'aide du schéma conceptuel, ensuite à manipuler la base d'objets réalisant le schéma conçu. Le gestionnaire du projet peut instancier certains objets, les mettre à jour, et également poser des questions sur la base d'objets pour répondre à des problèmes de gestion de projet. Un état de la base d'objet, à un moment donné, reflète un état de l'avancement du projet. Le gestionnaire perçoit un objet comme une relation entre les caractéristiques de cet objet, mais il ne voit pas son contenu de cet objet (l'objet physique). La base d'objets se comporte alors comme une base de données et le gestionnaire comme un utilisateur d'une base de données.

*
***
*****

CONCLUSION GENERALE

CONCLUSION GENERALE

La production effective et économique de logiciels est un problème d'une grande importance actuellement; la productivité (plus de programmation que de correction ou de tests) des programmeurs est faible, et la qualité (conformité à la spécification, facilité de maintenance et d'évolutivité) n'est pas toujours celle désirée. Il est donc important de fournir une assistance à l'utilisateur (concepteur, programmeur, ingénieur, gestionnaire, utilisateur final, etc.) au sein d'un atelier logiciel.

Dans un atelier logiciel, une grande quantité d'informations, notamment, les données issues du développement de logiciels et celles issues de la gestion de projets, doivent être partagées et utilisées par diverses applications. Ces données représentent les objets d'intérêt. Ces objets se manifestent avec certaines différences, parfois assez subtiles, et plus ou moins complexes. Tous ceux-là doivent être appréhendés et maintenus selon leurs sémantiques.

Nous avons abordé cette étude dans une approche Base de Données (BD) en envisageant une Base d'Ojets qui stocke et mette à disposition toute information relative au projet, et un Système de Gestion d'Ojets autour de la base d'objets. Un système de gestion d'objets est composé des outils de définition, de mémorisation et de recherche d'informations (objets et associations) et d'un ensemble de transactions sur la base, avec leurs conditions de déclenchement et leur enchaînement des actions associées. Cette approche est motivée par les préoccupations suivantes:

- viabilité et intégrité des informations mémorisées;
- accessibilité uniforme de ces informations aux différents membres d'un projet;
- variété et fiabilité des traitements pouvant être effectués sur la base.

Notre démarche suit les étapes suivantes:

(1) Utilisation du modèle Entité-Association étendu pour le schéma conceptuel dans le système de gestion d'objets.

Le modèle Entité-Association (E-A) est bien adapté pour répondre à la représentation d'un schéma conceptuel de la base d'objets, ceci, grâce à sa flexibilité et sa manière informelle d'abstraction.

Nous avons d'abord proposé une nouvelle interprétation du modèle E-A en terme de Dépendance Fonctionnelle entre Entités pour exprimer des sémantiques plus précises en éliminant certaines ambiguités. Ensuite une extension du modèle pour exprimer certaines caractéristiques hiérarchiques et pour décrire quelques types génériques d'association, s'appuie sur les notions d'abstraction de données: classification, généralisation, agrégation et partition. Nous avons étudié les règles d'application qui aident l'utilisateur à construire un schéma E-A, dit diagramme E-A bien défini.

Un prototype interactif de construction des schémas E-A est implanté. Les fonctions d'instanciation d'un schéma et d'interrogation de la base en terme de Entité-Association (un langage de BD en terme Entité-Association) seront décrites dans le cadre de travaux futurs.

(2) Transformation du schéma Entité-Association en un schéma relationnel pour exploiter un système de gestion de bases de données existant.

La transformation du schéma conceptuel E-A obtenu en un schéma global logique d'une base de données doit être systématique afin de permettre son implantation. Nous envisageons d'utiliser une BD de type relationnel supportée par un système de gestion de bases de données relationnel existant (ex. INGRES).

Nous avons proposé une méthode et un système de transformation à partir d'un diagramme E-A bien défini en un schéma relationnel de troisième forme normale. La réalisation d'un prototype d'interface graphique du système de transformation est en cours.

(3) Extension du langage de programmation de haut niveau LTR-3 pour les applications de base de données relationnelle.

Nous avons proposé un langage de BD appelé R-LTR (LTR Relationnel), qui est une extension de LTR-3 destinée aux applications de BD, en nous basant sur le modèle relationnel de données et sur la notion de type abstrait de données.

Nous avons introduit de nouveaux types de données TUPLE, RELATION et DATABASE avec un ensemble d'opérations de mise à jour et une expression relationnelle pour les manipulations des relations. En particulièr, nous avons pris en compte des contraintes d'intégrité internes à chaque niveau de définition d'une base de données. Une partie de la réalisation, analyse syntaxique par un préprocesseur, est expérimentalement faite. La partie d'interfaçage avec un système de gestion de bases de données reste encore à achever.

Ces études nous ont fait franchir un pas vers notre objectif qui est le développement d'une structure d'accueil basée sur un système de gestion d'objets, en vue de la construction d'ateliers logiciels.

*
***
*****

REFERENCES BIBLIOGRAPHIES

REFERENCES BIBLIOGRAPHIES

[ABRIAL74] ABRIAL J. R.

"Data Semantics", Data Management Systems, (Ed. J. W. Klimbie et K. L. Koffeman), North-Holland, Amsterdam, pp.1-59, 1974.

[ACM79] ASSOCIATION FOR COMPUTING MACHINERY

"Preliminary ADA Reference Manual", ACM SIGPLAN Notices, Vol.14, No.6, Juin 1979.

[AHO79] AHO A. V., C. BEERI et J. D. ULLMAN

"The Theory of Joins in Relational Databases", ACM Trans. on Database Systems, Vol.4, No.3, pp.297-314, Mars 1979.

[ALLMAN76] ALLMAN E., M. STONEBRAKER et G.HELD

"Embedding a relational data sublanguage in a general purposed programming language", Comm. ACM SIGPLAN notices, Vol.11, special issue, pp.25-35, Mars 1976.

[ANSI75] ANSI/X3/SPARC

"Interim Report from the Study Group on Database Management Systems" ACM SIGMOD Bulletin, Vol.7, No.2, 1975.

[ARON85] ARON J. D.

"Managing programming teams", Proc. Convention Informatique Latine, Barcelona Espagne, pp.3-46, Avril 1985.

[ASTRAHAN76] ASTRAHAN M. M., M. W. BLASGEN, D. D. CHAMBERLIN et al.

"System R: Relational Approach to Database Management", ACM Trans. on Database Systems, Vol.1, No.2, pp.97-137, Juin 1976.

[BARSTOW84] BARSTOW D. R., H. E. SHROBE et E. SANDEWALL (Ed.)

"Interactive Programming Environments", McGraw-Hill Book Co., 1984.

[BAUER80] BAUER F. L.

"A Trend for the next 10 years of Software Engineering", Software Engineering, (Ed. M. Freeman et P. M. Lewis II), Academic Press,

pp.1-23, 1980.

[BENALI85]* BENALI K.

"Structure d'accueil SGO-CRIN basée sur la notion d'objet informatique, pour le développement d'atelier génie logiciel assurant la cohérence de projets", Rapport de D.E.A. Informatique, Université de Nancy I, Septembre 1985.

[BERNSTEIN76] BERSTEIN P. A.

"Synthesizing Third Normal Form Relation from Functional Dependencies", ACM Trans. on Database Systems, Vol.1, No.4, pp.277-298, Décembre 1976.

[BLUM82] B. I. BLUM

"The Life Cycle - A Debate Over Alternate Models", ACM SIGSOFT Notes, Vol.7, No.4, pp.18-20, Octobre 1982.

[BODART83] BODART F. et Y. PIGNEUR

"Conception assistée des applications informatiques - 1. étude d'opportunité et analyse conceptuelle", Masson, Presses Universitaires de Namur, 1983.

[BOEHM79] BOEHM B. W.

"Software Engineering - As It Is", Proc. 4th Int. Conf. on Software Engineering, Munich West Germany, Septembre 1979.

[BOEHM84] BOEHM B. W.

"Software Life Cycle Factors", Handbook of Softeware Engineering, (Ed. C. R. Vick et C. V. Ramamoorthy), Van Nostrand Reinhold Co., pp.494-518, 1984.

[BOLL80] BOLL M. et J. REINHART

"Histoire de la Logique", (8ème Edition), Presses Universitaires de France, 1980.

(*) Bibliographie liée à l'équipe Génie Logiciel du C.R.I.N..

[BOURGEOIS85] BOURGEOIS J.

"Ateliers de Génie Logiciel: Etat de l'art et perspectives", Agence De l'Informatique, Génie Logiciel, No.1, pp.4-14, 1985.

[BOYCE75] BOYCE R. F., D. D. CHAMBERLIN et al.

"Specifying queries as relational expression: The SQUARE data sublanguage", Comm. ACM, Vol.18, No.11, pp.621-628, Nov. 1975.

[BRACCHI79] BRACCHI G., A. FURTADO et G. PELAGATTI

"Constraint specification in evolutionary data base design", IFIP TC8 WC on "Formal models and practical tools for information system design", Oxford, 1979.

[BRACHMAN83] BRACHMAN R. J.

"What IS-A is and isn't: An Analysis of Taxonomic Links in Semantic Networks", IEEE Computer, Vol.16, No.10, pp.30-36, Oct. 1983.

[BRODIE78] BRODIE M. L. et J. SCHMIDT

"What is the use of Abstract Data Types in Data Bases ?", Proc. 4th Int. Conf. Very Large Data Bases, pp.140-141, Sept. 1978.

[BRODIE84-a] BRODIE M. L.

"On the development of Data Models", On Conceptual Modelling, Springer-Verlag Inc., pp.19-47, 1984.

[BRODIE84-b] BRODIE M. et D. RIDJANOVIC

"On the Design and Specification of Database Transactions", On Conceptual Modelling, Springer-Verlag Inc., pp.277-312, 1984.

[CASANOVA84-a] CASANOVA M. A. et J. E. AMARAL DE SA

"Mapping uninterpreted Schemes into Entity-Relationship diagrams: Two Applications to Conceptual Schema Design", IBM J. Res. Develop., Vol.28, No.1, pp.82-94, Janvier 1984.

[CASANOVA84-b] CASANOVA M. A. et R. FAGIN

"Inclusion Dependencies and Their Interaction with Functional Dependencies", J. of Computer and System Sciences, Vol.28, No.1, pp.29-59, Février 1984.

[CHABRIER82-a]* CHABRIER J. J. et J. C. DERNIAME

"Programmation à l'aide de systèmes abstraits, le système TYP", AFCET TSI, Vol.1, No.4, pp.287-296, Juillet-Août 1982.

[CHABRIER82-b]* CHABRIER J. J.

"Spécification et construction de systèmes orientés bases de données: Technique et langages basés sur le concept de type abstrait", Thèse de Doctorat d'Etat, Université de Nancy I, Octobre 1982.

[CHAILLOUX84] CHAILLOUX J.

"LE_LISP de l'I.N.R.I.A.", Le Manuel de Référence - Version 15, INRIA, Décembre 1984.

[CELAR83]

"Spécifications Fonctionnelles de l'atelier LTR3", Centre d'Electronique de L'ARmement, Décembre 1983.

[CHEN76] CHEN P._S. P.

"The Entity-Relationship Model - Toward a Unified View of Data", ACM Trans. on Database Systems, Vol.1, No.1, pp.9-36, Mars 1976.

[CHEN77] CHEN P._S. P.

"The entity-relationship model - A basis for the enterprise view of data", Proc. AFIPS Nat. Comput. Conf., pp.77-84, 1977.

[CHEN82] CHEN P._S. P.

"Applications of the Entity-Relationship Model", Data Base Design Techniques I, LNCS No.132, Springer-Verlag Inc., pp.87-113, 1982.

[CHERBONNEAU85] CHERBONNEAU B., A. MATHIS et G. OLIVIER

"L'atelier SSP : Système de production de logiciel assistée par ordinateur", ADI, Génie Logiciel, No.1, pp.27-33, 1985.

[CIMSA83]

"LTR-3 Manuel Officiel de Référence", Compagnie d'Informatique Militaire Spaciale et Aéronautique, 1983.

[CODD70] CODD E. F.

"A Relational Model of Data for Large Shared Data Banks", Comm. ACM, Vol.13, No.6, pp.377-387, Juin 1970.

[CODD72-a] CODD E. F.

"Futher Normalization of the Data Base Relational Model", Data Base Systems, (Ed. R. Rustin), Prentice-Hall, Englewood Cliffs, pp.33-64, 1972.

[CODD72-b] CODD E. F.

"Relational Completeness of Data Base Sublanguages", Data Base Systems, (Ed. R. Rustin), Prentice-Hall, Englewood Cliffs, pp.65-98, 1972.

[CODD79] CODD E. F.

"Extending the Database Relational Model to Capture More Meaning", ACM Trans. on Database Systems, Vol.4, No.4, pp.397-434, Décembre 1979.

[CODD82] CODD E. F.

"Relational Database: A Practical Foundation for Productivity", Comm. ACM, Vol.25, No.2, pp.109-117, Février 1982.

[COHEN83] COHEN G. et G. GERMAIN

"Un Système Intégré de Génie Logiciel (SIGL)", BIGRE, No.35-36, pp.325-333, Octobre 1983.

[CRUZLARA84]* CRUZLARA SILVA S.

"Conception et Réalisation du Préprocesseur R-LTR", Rapport du D.E.A. Informatique, Université de Nancy I, Septembre 1984.

[CRUZLARA86]* CRUZLARA SILVA S.

"Une approche pour la génération d'environnements de programmation orientés langages", CRIN Rapport de Recherche, No.86-R-045, Proc. JISI86, les 2ème Journées Internationales des Sciences Informatiques, Tunis Tunisie, Avril 1986.

[DAHL72] DAHL O. J. et C. A. R. HOARE

"Hierarchical Programming Structures", Structured Programming,

Academic Press, pp.175-220, 1972.

[DATE81] DATE C. J.

"An Introduction to Database Systems", (3rd Edition) Eddison-Wesley Publ. Co., 1981.

[DATE82] DATE C. J.

"A formal definition of the relational model", ACM SIGMOD Record, Vol.13, No.1, pp.18-29, Septembre 1982.

[DATE83] DATE C. J.

"An Introduction to Database Systems - volume II", Eddisson-Wesley Publ. Co., 1983.

[DE DROUAS82] DE DROUAS E. et J. M. NERSON

"Les ateliers logiciels intégrés : Développements français actuels", AFCET TSI, Vol.1, No.3, pp.211-232, Mai-Juin 1982.

[DELOBEL82] DELOBEL C. et M. ADIBA

"Bases de données et Systèmes Relationnels", Dunod, Paris, 1982.

[DERNIAME84]* DERNIAME J. C. et M. J. KIM

"Data type of Relation in a High Level Programming Language: LTR-3", CRIN Rapport de Recherche, No.84-R-003, 1984.

[DERNIAME85]* J. C. DERNIAME, C. GODART, M. J. KIM, N. TOUNSI et al.

"Proposition d'un projet: SGO-CRIN", CRIN Rapport interne de l'équipe Génie Logiciel, Mai 1985.

[DIJKSTRA72] DIJKSTRA E. B.

"The humble programmer", Comm. ACM, Vol.15, No.10, pp.859-866, 1972.

[DOLOTTA78] DOLOTTA T. A., R. C. HAIGHT et J. R. MASMEY

"UNIX Time-Sharing System : The Programmer's Workbench", The Bell System Technical Journal, Vol.57, No.6, Juillet-Août 1978.

[EL_MASRI80] EL_MASRI R. et G. WIEDERHOLD

"Properties of relationships and their representation", Proc.

Nat. Comput. Conf. 1980, pp.319-326, 1980.

[EMERAUDE84] EMERAUDE: Groupe d'Intérêt Economique
"Les spécifications fonctionnelles d'EMERAUDE (Version 1.2)", Rapport intermédiaire, Avril 1984.

[FAGIN77] FAGIN R.
"Multivalued dependencies and a new normal form for relational databases", ACM Trans. on Database Systems, Vol.2, No.3, pp.262-278, Sept. 1977.

[FAGIN79] FAGIN R.
"Normal forms and relational database operators", Porc. ACM SIGMOD, (Ed. P. A. Bernstein) pp.153-160, 1979.

[FELDMAN78] FELDMAN S. I.
"Make - A program for Maintaining Computer Programs", Document du système UNIX, Août 1978.

[FLORY82] FLORY A.
"Base de Données: Conception et Réalisation", Economica, Paris, 1982.

[FOISSEAU85] FOISSEAU J., R. JACQUART, M. LEMAITRE et al.
"Le système Sparc : expression et gestion de spécification, d'algorithmes et de représentations", AFCET TSI, Vol.4, No.2, pp.237-254, Mars-Avril 1985.

[FOUCAUT82] FOUCAUT O.
"Modèle et outil pour conception des systèmes d'information dans les organisations. Projet REMORA", Thèse de Doctorat d'Etat, Université de Nancy I, Juin 1982.

[FOUCAUT84] FOUCAUT O.
"Bases de Données", Support de cours, CRIN No.84-E-024, 1984.

[FRAVAL85] FRAVAL J. C.
"VULCAIN: Un Atelier de Génie Logiciel", ADI Génie Logiciel, No.1, pp.24-27, 1985.

[GALACSI84] GALACSI

"Les systèmes d'information - Analyse et conception", Dunod, Paris, 1984.

[GARDARIN83] GARDARIN G.

"Intégration d'un langage de manipulation de données dans un langage de programmation", Bases de données, Nouvelles Perspectives, Rapport de groupe BD3, INRIA et ADI, Janvier 1983.

[GARDARIN85] GARDARIN G. et P. VALDURIEZ

"Bases de Données Relationnelles: Analyse et comparaison des systèmes", Eyrolles Paris, 1985.

[GEND83]* GEND P.

"Analyse Syntaxique et Optimisation des Expressions Relationnelles de R-LTR", Rapport de D.E.S.S. Informatique, Université de Nancy I, Septembre 1983.

[GODART81]* GODART C.

"SGMB: Système de Gestion Multibase", Thèse de Docteur 3ème Cycle, Université de Nancy I, Juillet 1981.

[GODART84]* GODART C., M. J. KIM, R. NASSIF, J.C. DERNIAME, W. LITWIN

"ADONIS: A Multi-database Design Method and a Multi-database Management System", CRIN Rapport de Recherche, No.84-R-20, 1984.

[GODART86]* GODART C., K. BENALI et J. C. DERNIAME

"Propositions pour un Système de Gestion d'Objets", Proc. 3ème Congrès Génie Logiciel, Versailles France, Mai 1986.

[GOLDBERG83] GOLDBERG A.

"The Influence of an Object-Oriented Langauge on the Programming Environment", Proc. ACM Computer Science Conference, Orlando Florida U.S.A., pp.35-54, Février 1983.

[GUTTAG77] GUTTAG J. V.

"Abstract Data Types and the Development of Data Structures", Comm. ACM, Vol.20, No.6, pp.396-404, Juin 1977.

[GUTTAG78] GUTTAG J. V., E. HORWITZ et D. R. MUSSER
"Abstract Data Types and Software Validation", Comm. ACM, Vol.21, No.12, pp.1048-1063, Décembre 1978.

[HELD75] HELD G. D., M. R. STONEBRAKER et E. WONG
"INGRES - A relational data base system", Proc. AFIPS Nat. Comput. Conf., pp.409-415, Mai 1975.

[HOARE72] HOARE C. A. R.
"Notes on Data Structuring", Structured Programming, Academic Press, London, pp.83-174, 1972.

[HOWDEN82] W. E. HOWDEN
"Contemporary Software Development Environments", Comm. ACM, Vol.25, No.5, pp.318-329, Mai 1982.

[HOWE83] HOWE D. R.
"Data Analysis for Data Base Design", Edward Arnold Publ. Ltd., London, 1983.

[HULLOT84] HULLOT J. M.
"Programmer en CEYX - Version 15", INRIA, Eté 1984.

[IBM82] IBM Program Product
"SQL/Data System - General Information, Concepts and Facilities", IBM 1982.

[JAJODIA83] JAJODIA S., P. A. NG et F. N. SPRINGSTEEL
"The problem of Equivalence for Entity-Relationship Diagrams", IEEE Trans. on Software Engineering, Vol.SE-9, No.5, pp.617-630, Septembre 1983.

[KAHN82] KAHN B. K.
"LDDM - A Structured Logical Database Design Methodology", Data Base Design Techniques I, LNCS No.132, Springer-Verlag Inc., pp.31-55, 1982.

[KENT83] KENT W.
"Fact-Based Data Analysis and Design", Entity-Relationship

Approach to Software Engineering, Proc. 3rd Int. Conf. on Entity-Relationship Approach, Anaheim California U.S.A., pp.3-54, Septembre 1983.

[KENT84] KENT W.

"A Realistic Look at Data", IEEE TC on Database Engineering, Vol.7, No.4, pp.22-27, Décembre 1984.

[KIM85]* KIM M. J. et J. C. DERNIAME

"Un guide pour la construction de schéma relationnel en troisième forme normale par transformation de modèle Entité-Association", CRIN Rapport de Recherche, No.84-R-82, Proc. Convention Informatique Latine, Barcelona Espagne, pp.416-437, Avril 1985.

[KIM86-a]* KIM M. J., N. TOUNSI et J. C. DERNIAME

"Cohérence d'un schéma de base de données de type Entité-Association", CRIN Rapport de Recherche, No.86-R-013, Proc. Congrès INFOCOM'86, Paris France, Juin 1986.

[KIM86-b]* KIM M. J., N. TOUNSI et J. C. DERNIAME

"Méthode et outils pour la conception du schéma relationnel: le passage du modèle Entité-Association au modèle relationnel", CRIN Rapport de recherche, No.86-R-038, 1986.

[KING84] KING R. et D. McLEOD

"A Unified Model and Methodology for Conceptual Database Design", On Conceptual Modelling, Springer-Verlag Inc. New York, pp.313-327, 1984.

[KNUTH73] KNUTH D. E.

"Fundamental Algorithms - The Art of Computer Programming", Vol.1, 2nd Edition, Addison Wesley Publ. Co., 1973.

[KRAKOWIAK82] KRAKOWIAK S.

"Systèmes intégrés de production de logiciels: Concepts et réalisations", AFCET TSI, Vol.1, No.3, pp.187-200, Mai-Juin 1982.

[LACROIX77] LACROIX M. et A. PIROTTE

"An English Structured Query Language for Relational Data Bases", Proc. IFIP, Nice France, pp.223-249, Janvier 1977.

[LENZERINI83] LENZERINI M. et G. SANTUCCI

"Cardinality Constraints in the Entity-Relationship Model", Entity-Relationship Approach to Software Engineering, Proc. 3rd Int. Conf. on Entity-Relationship Approach, Anaheim California U.S.A., pp.529-550, Octobre 1983.

[LISKOV74] LISKOV B. et S. ZILLES

"Progamming with Abstact Data Types", ACM SIGPLAN notices, Vol.9, No.4, pp.50-59, Avril 1974.

[LISKOV81] LISKOV B., R. ATKINSON, T. BLOOM et al.

"CLU Reference Manual", LNCS No.114, Springer-Verlag Inc., 1981.

[LITWIN81] LITWIN W.

"Présentation du projet B A BA", Publication INRIA-SIRIUS, No. GAL-I-042, 1981.

[LOCKEMANN83] LOCKEMANN P. C.

"Analysis of Version and Configuration Control in a Software Engineering Environment", Entity-Relationship Approach to Software Engineering, Proc. 3rd Int. Conf. on Entity-Relationship Approach, Anaheim California U.S.A., pp.701-713, Octobre 1983.

[LOPEZ83] LOPEZ M., J. PALAZZO OLIVEIRA et F. VELEZ

"The TIGRE Data Model", Rapport de recherche TIGRE No.2, CII Honeywell Bull et Laboratoire IMAG, Novembre 1983.

[McCRACKEN82] McCRACKEN D. D. et M. A. JACKSON

"Life Cycle Concept Considered Harmful", ACM SIGSOFT Notes, Vol.7, No.2, pp.29-32, Avril 1982.

[McDONALD81] McDONALD M. et J. McNALLY

"Feuture Analysis of INGRES", Relational Database Systems, Springer-Verlag Inc., pp.158-181, 1983.

[MEYER85-a] MEYER B.

"The Software Knowledge Base", Proc. 8th Int. Conf. on Software Engineering, London, pp.158-165, Août 1985.

[MEYER85-b] MEYER B.

"Etapes sur le chemin du génie logiciel", Thèse de Doctorat d'Etat, Université de Nancy I, Septembre 1985.

[MILLER56] MILLER G. A.

"The magical number seven plus or minus two: Some limits on our capacity for processing information", Psychological Review, Vol.63, pp.81-97, 1956, Traduit et publié dans "Textes pour une psycholinguistique", (Ed. J. Mehler et G. Noizet), Mouton Paris, pp.337-364, 1974.

[MIRANDA78] MIRANDA S.

"La sécurité des Données dans les SGBD relationnels", Colloque Modèles Relationnels, Institut Programmation, Paris, pp.34-72, Mars 1978.

[MIRANDA84] MIRANDA S. et J. M. BUSTA

"L'art des Bases de Données: Introduction aux Bases de Données", Eyrolles, Paris, 1984.

[MIRANDA86] MIRANDA S. et J. M. BUSTA

"L'art des Bases de Données: Les Bases de Données Relationnelles", Eyrolles, Paris, 1986.

[MORFUNI85] MORFUNI N. M.

"Equivalence Transformations in Conceptual Design", Proc. Convention Informatique Latine, Barcelona Espagne, pp.280-317, Avril 1985.

[NAVATHE83] NAVATHE S. B. et A. CHENG

"A Methodology for Database Schema Mapping from Extended Entity-Relationship Models into the Hierarchical Model", Entity-Relationship Approach to Software Engineering, Proc. 3rd Int. Conf. on Entity-Relationship Approach, Anaheim California U.S.A.,

pp.223-248, Octobre 1983.

[NASSIF83]* NASSIF R.

"Projet ADONIS: Conception et Réalisation d'un Système de Gestion de Multibase de données", Thèse de Docteur Ingénieur, Institut National Polytechnique de Lorraine, Septembre 1983.

[NICOLAS83] NICOLAS J. M.

"Les règles de cohérence ou contrainte d'intégrité", Bases de données - Nouvelles Perspectives, Rapport de groupe BD3, INRIA et ADI, Janvier 1983.

[OLLE85] OLLE T. W.

"Information Systems Design Methodologies - The choice available to the practitioner", Proc. Convention Informatique Latine, Barcelona Espagne, pp.79-97, Avril 1985.

[OSEBON83] OSEBON P.

"Conception de Bases de Données avec un Modèle Sémantique Hiérarchique", AFCET Ecole Internationale d'Informatique, Sfax Tunisie, Juillet 1983.

[PALAZZO83] PALAZZO OLIVEIRA J. et F. VELEZ

"La correspondance de schémas entre les modèles TIGRE et relationnel", Rapport de recherche TIGRE, No.5, CII Honeywell Bull et Laboratoire IMAG, Novembre 1983.

[PALMER82] PALMER I. R.

"Practicalities in Applying a Formal Methodology to Data Analysis", Data Base Design Techniques I, LNCS No.132, Springer-Verlag Inc., pp.147-171, 1982.

[PARNAS79] PARNAS D. L.

"Designing Software for Ease of Extension and Contraction", IEEE Trans. on Software Engineering, Vol.SE-5, No.2, pp.128-137, Mars 1979.

[PIROTTE81] PIROTTE A.

"A Precise Definiton of Basic Relational Notions and of the Relational Algebra", ACM SIGMOD, Vol.13, No.1, pp.30-45, Septembre 1981.

[RISSANEN77] RISSANEN J.

"Independent Components of Relations" ACM Trans. on Database Systems, Vol.2, No.4, pp.317-325, Décembre 1977.

[PRENNER78] PRENNER C. J. et L. A. ROWE

"Programming languages for relational data base systems", Proc. AFIPS Nat. Comput. Conf., pp.849-855, 1978.

[ROLLAND76] ROLLAND C.

"Quelques éléments du modèle conceptuel dans le projet REMORA", INFORSID Séminaire langage d'analyse, Nancy, Mai 1976.

[ROWE79] ROWE L. A. et K. A. SHOENS

"Data Abstraction, View and Update in RIGEL", ACM SIGMOD Proc. Int. Conf. on Management of Data, pp.71-81, Mai 1979.

[SAKAI83] SAKAI H.

"Entity-Relationship Approach to Logical Database Design", Entity-Relationship Approach to Software Engireering, Proc. 3rd Int. Conf. on Entity-Relationship Approach, Anaheim California U.S.A., pp.155-187, Octobre 1983.

[SCHEUERMANN80] SCHEUERMANN P., G. SCHIFFNER et H. WEBER

"Abstraction Capabilities and Invariant Properties Modelling within the Entity-Relationship Approach", The Entity-Relationship Approach to System Analysis and Design, (Ed. P. P._S. Chen), North-Holland Publ. Co., 1980.

[SCHMIDT77] SCHMIDT J. W.

"Some High Level Language Constructs for Data of Type Relation", ACM Trans. on Data base systems, Vol.2, No.3, pp.247-261, Septembre 1977.

[SINHA83] SINHA M. K.

"Constraints: Consistency and Integrity", ACM SIGMOD RECORD, Vol.13, No.2, pp.60-63, Janvier 1983.

[SMITH77-a] SMITH J. M. et D. C. P. SMITH

"Database Abstractions : Aggregation", Comm. ACM, Vol.20, No.6, pp.405-413, Juin 1977.

[SMITH77-b] SMITH J. M. et D. C. P. SMITH

"Database Abstractions: Aggregation and Generalization", ACM Trans. on Database Systems, Vol.2, No.2, pp.105-133, Juin 1977.

[SMITH82] SMITH J. M. et D. C. P. SMITH

"Principles of Database Conceptual Design", Data Base Design Techniques I, LNCS No.132, Springer-Verlag Inc., pp.114-146, 1982.

[SYSECA83] SYSECA

"Générateur d'analyseur LL(1) (GALL)", Manuel d'utilisation, Octobre 1983.

[SYSECA84] SYSECA

"PASCAL/ 68000/ SOL - UNIX" Manuel d'utilisation sur MICROMEGA, Janvier 1984.

[TARDIEU76] TARDIEU H., H. HECKENROTH, D. NANCI et A. ISSADI

"Méthode, modèles et outils pour la conception de la base de données d'un système d'information", Manuel de référence, CETE Aix-en-Provence, Décembre 1976.

[TARDIEU83] TARDIEU H., A. ROCHFELD et R. COLETTI

"La méthode MERISE - Principes et outils", Editions d'organisation, Paris, 1983.

[TARDIEU85] TARDIEU H., D. NANCI et D. PASCOT

"Conception d'un Système d'information - Construction de la base de données", Les éditions d'organisation, Paris, 1985.

[TEITELMAN81] TEITELMAN W. et L. MASINTER

"The Interlisp Programming Environment", IEEE Computer, Vol.14, No.4, pp.25-34, Avril 1981.

[THIERY85] THIERY O.

"LASSIF: LAngage de Spécification des Systèmes d'InFormation Logiciel d'Aide à la Spécifications des Systèmes d'Information", Thèse de Doctorat d'Etat, Université de Nancy I, Juin 1985.

[TOUNSI82]* TOUNSI N., J. C. DERNIAME, J. J. CHABRIER et W. LITWIN

"Un système de bases de données relationnelles réalisé à l'aide de types abstraits de données", Proc. Segundo Symposia Espanol de Informatica Repartida, Santiago de Compostella Espagne, Septembre 1982.

[TOUNSI83]* TOUNSI N.

"TYP-R : Réalisation en TYP d'un système multibase relationnel", Thèse de Docteur 3ème cycle, Université Nancy I, Avril 1983.

[TOUZANI85]* TOUZANI A.

"Spécification d'une interface utilisateur pour modèle Entité-Association", Rapport de D.E.A. Informatique, Université de Nancy I, Septembre 1985.

[TSICHRITZIS82] TSICHRITZIS D. C. & F. H. LOCHOVSKY

"Data Model", Prentice-Hall Inc., Englewood Cliffs, 1982.

[ULLMAN82] ULLMAN J. D.

"Principles of Database Systems", 2nd Edition Computer Science Press Inc., 1982.

[WASSERMAN79] WASSERMAN A. I.

"The Data Management Facilities of PALIN", ACM SIGMOD Proc. Int. Conf. on Management of Data, pp.60-70, Mai 1979.

[WASSERMAN81] WASSERMAN A. I., D. D. SHERERTZ, M. L. KERSTEN et al.

"Revised report on the programming language: PLAIN", ACM SIGPLAN notices, Vol.16, No.5, pp.59-80, Mai 1981.

[WILKIE83] WILKIE M. S. D. et A. O. MENDELZON

"Analysis and Translation of Entity-Relationship Database Schemas", Information Processing 83, IFIP83, Elsevier Science Publ. Co., pp.609-614, 1983.

[WINOGRAD73] WINOGRAD T.

"Breaking the Complexity Barrier (Again)", Proc. ACM SIGPLAN-SIGIR, pp.13-30, Novembre 1973.

[WIRTH77] WIRTH N.

"Modula : A Language for Modular Multiprogramming", Software - Practice and Experience, Vol.7, pp.3-35, 1977.

[WOLF81] WOLF M. I., W. BABICH, R. SIMPSON et al.

"The Ada Language System", IEEE Computer, Juin 1981.

[WONG80] WONG E. et R. H. KATZ

"Logical Design and Schema Conversion for Relational and DBTG Databases", The Entity-Relationship Approach to System Analysis and Design, (Ed. P._S. P. Chen), North-Holland Publ. Co., pp.311-321, 1980.

[WOODFILL84] WOODFILL J., P. SIEGAL, J. RANSTROM et al.

"INGRES - Version 7", Reference Manuel, Mars 1984.

[YAO82] YAO S. B., S. B. NAVATHE et J_L. WELDON

"An Integrated Approach to Database Design", Data Base Design Techniques I, LNCS No.132, Springer-Verlag Inc., pp.1-30, 1982.

[ZELKOVIC78] ZELKOVIC M. V.

"Perspectives on software engineering", ACM Computer Surveys, Vol.10, No.2, pp.197-216, Juin 1978.

*
***
*****

INDEX

INDEX

action associée, V-2.1.2.
agrégation, III-5.3.
association, III-2.1.
- (voir aussi Type d'Association)
atelier logiciel, V-2.
attribut
- attribut d'une relation, I-2.1.2.
- attribut d'une entité, III-2.1.
- attribut d'une association III-2.1.

Base de Données (BD), I-1.
base d'objets, V-2.

cardinalité, III-3.1.2.
- (voir aussi Type d'Association)
classe d'objets, III-5.1.
classification, III-5.1.
clé de relation, I-2.1.2., I-3.1.2.
- clé candidate, I-2.1.2.
- clé principale, I-2.1.2.
clé d'un type d'entité, III-2.1.
clé d'un type d'association, III-2.1.
Contrainte d'Intégrité (CI), I-1.
- inhérente, I-1., I-2.1.3.
- explicite, I-1.
- structurelle, I-1., I-3.1.
- interne, I-1., I-3.2.
- implicite, I-1.

décomposition, III-5.3.
dépendance d'existence, III-3.2.1.
Dépendance Fonctionnelle (DF), I-3.1.1.
- élémentaire, I-3.1.1.

- transitive, I-3.1.1.
Dépendance Fonctionnelle entre Attributs (DFA), III-4.1.
Dépendance Fonctionnelle entre Entités (DFE), III-3.2.2.
Dépendance d'Inclusion (DI), I-3.1.3.
dépendance d'identification, III-3.2.1.
diagramme Entité-Association (E-A), III-2.2.
- bien défini, III-4.1.3.
DIFFER, I-2.3.2.
domaine, I-2.1.1.

ea, IV-4.1.
entité, III-2.1.
- (voir aussi Type d'Entité)
état de base de données, I-1.
expression relationnelle, I-2.3., II-3.2.

fonctionnalité, III-3.1.1.

généralisation, III-5.2.

instance, III-5.1.
instanciation, III-5.1.
INTERSECT, I-2.3.2.

JOIN, I-2.3.2.

langage de base de données, I-1., II-1.
langage de définition des données, I-1., II-2.
langage de manipulation des données, I-1., II-3.

modèle de données, I-1.
- modèle relationnel, I-2.
- modèle Entité-Association (E-A), III-2.1.

objet, I-2.1.1.
- objet agrégat, III-5.3.

- objet composant, III-5.3.
- objet ensemble, III-5.4.
- objet membre, III-5.4.
- (voir aussi classe d'objets, type d'objets)

partition, III-5.4.
PROJECT, I-2.3.2.

regroupement, III-5.4.
relation, I-2.1.1.
- relation de base, I-2.3.1., II-2.2.
- relation temporaire, I-2.3.1., II-2.2.
- relation intermédiaire, I-2.3.1.
R-LTR (Langage de Temps Réel Relationnel), II-1.
rubrique, I-2.1.1.

schéma conceptuel de données, III-1.
schéma de données, I-1.
- schéma de relation, I-2.2.
- schéma relationnel, I-2.2.
- schéma Entié-Association (E-A), III-2.2.
SELECT, I-2.3.2.
spécialisation, III-5.2.
structure d'accueil, V-2.
SGO-CRIN, V-2.2.
Système de Gestion de Bases de Données (SGBD), I-1.
Système de Gestion d'Objets (SGO), V-2.

trn, IV-4.2.
Troisième Forme Normale (3FN), I-3.1.2.
Troisième Forme Normale de Boyce-Codd (3FNBC), I-3.1.2.
tuple, I-2.1.2., II-2.1.
type abstrait de données, II-1.
Type d'Association (TA), III-2.1.
- bien défini, III-4.1.2.
- cardinalité, III-3.1.2.

- degré, III-3.1.
Type d'Entité (TE), III-2.1.
- bien défini, III-4.1.1.
- faible, III-3.2.1.
- obligatoire, III-2.1.
- optionnel, III-2.1.
- régulier, III-3.2.1.
type d'objets, III-5.1.

UNION, I-2.3.2
union-compatible, I-2.3.2.

*
***
*****

CURRICULUM VITAE

Nom: KIM Myung-Joon Sexe: Masculin

Né: le 7 Août 1955 à: Busan, Corée du sud

Nationalité: Sud Coréenne (République de la Corée)

Etudes suivies:

(1) Baccalauréat

Février 1974.

(2) Bachelor of Science (Maîtrise)

Février 1978,

Université Nationale de Séoul (Séoul, Corée du sud),

Département Informatique et Statistique.

(3) Master of Science (D.E.A.)

Février 1980,

Institut National Supérieur de Science et Technologie (Séoul)

Département Informatique.

(4) Doctorat de l'Université de Nancy I (nouveau doctorat)

Mai 1986,

Université de Nancy I, U.E.R. Science Mathématique,

Département Mathématique Appliquée et Informatique.

Carrière:

(1) Assistant

Février 1980 - Juin 1981,

Université d'Ajou (Suwon, Corée du sud).

(2) Chercheur Etudiant

Octobre 1981 - Mai 1986,

Centre de Recherche en Informatique de Nancy,

Equipe Génie Logiciel.

NOM DE L'ETUDIANT : *KIM Myung-Joon*

NATURE DE LA THESE : *Doctorat de l'Université de NANCY I en Informatique*

BIBLIOTHEQUE-SCIENCES

VU, APPROUVE ET PERMIS D'IMPRIMER

NANCY, le 28 AVR. 1986

LE PRESIDENT DE L'UNIVERSITE DE NANCY I

UNIVERSITE DE NANCY I
Le Président
MAINARD

RESUME

Cette thèse a pour but de proposer une solution pour la représentation et la gestion des objets nécessaires au développement de logiciels ainsi qu'à la gestion de projets. La solution proposée intégre les expériences acquises ces dernières années.

Nous abordons ce problème par l'approche base de données en étudiant les techniques et les méthodes qu'elle propose. Les travaux menés se divisent en trois étapes progressives:

- Etude de la base de données relationnelle et intégration des applications de base de données relationnelle dans le langage LTR-3 à l'aide de la notion de type abstrait de données.

- Etudes approfondies sur le modèle Entité-Association et proposition d'une méthode de transformation du modèle Entité-Association au modèle relationnel.

- Première description d'un système de gestion d'objets basé sur le modèle Entité-Association avec la spécification de ses fonctions.

MOTS CLES

Modèle de Données Relationnel, Langage de Base de Données, Extension d'un Langage de Programmation, Conception de Base de Données, Modèle Conceptuel (Modèle Sémantique), Modèle Entité-Association, Génie Logiciel, Système de Gestion d'Objets.