STUDIA UNIV. BABEŞ-BOLYAI, MATHEMATICA, XL, 3, 1995

UN MODELE DE REPRESENTATION D'OBJETS COMPLEXES AVEC IDENTITE D'OBJETS

Gr. MOLDOVAN* and C. BOBOILA**

Received: 9.10.1995
AMS subject classification: 68P15, 68R10

REZUMAT. - Un model de reprezentare a obiectelor complexe cu identitate de obiecte. Durant ces dernières années, les systèmes de bases de données orientés-objet ont émergé et sont en passe de devenir les principaux systèmes commerciaux des années 1995. Ces systèmes fournissent aux utilisateurs des possibilités de modélisation plus variées que les systèmes relationnels.

A partir de constructeurs tels que les constructeurs de n-uplet, ensemble, tableau, liste, et en imbriquant arbitrairement ceux-ci, de tels systèmes supportent la notion d'objet complexes avec identité d'objet.

Cet article présente un modèle de représentation d'objets complexes avec identité d'objet au moyen des graphes.

On définit les notions de graphe des types et graphe de composition d'objets pour mettre en évidence les connexions inter-objet.

Mots clés: Objets complexes, Valeurs structurées, Types structurés, Identité d'objet, Graphe des types, Graphe des types avec héritage, Graphe de composition d'objets.

1. Introduction

Des nouveaux domaines d'application émergent depuis quelques années. Ces domaines tels que: la conception assistée par ordinateur (CAO), la production de documents incorporant textes, images et graphiques, le génie logiciel, nécessitent la gestion d'une grande variété de types de données ainsi que les liens entre ces données (souvent imbriquées). Ces

* *"Babeş-Bolyai" Université, Faculté de Mathematique et Informatique, 3400 Cluj-Napoca, Romania*

** *Université de Craiova, Département de l'Informatique, 1100 Craiova, Romania*

données sont généralement appelées **Objets complexes.**

Le modèle relationnel [CODD.70] ne permet pas de modéliser efficacement ces types variés de données ainsi que leur imbrication. Dans ce modèle, les données sont représentées sous forme de relations "plates", c'est la contrainte de première forme normale. Les attributs d'un n-uplet étant nécessairement des valeurs atomiqurs (entiers, caractères, réels, booléens), il est difficile de représenter un objet complexe dans son ensemble.

D'autre part, le problème du dysfonctionnement entre langage de manipulation des données et lagages de programmation [ATKI.87] constitue un inconvénient supplémentaire de ces systèmes.

Plusieurs tentatives d'extension du modèle relationnel ont été effectuées au cours de ces dernières années [MAKI.77], [SCHE.82], [ZANI.85], [ABIT.86], [BANC.86].

Bien que ces modèles généralisent le modèle relationnel, ils ne permettent pas de fournir le partage d'objets par référence [KOSH.87]. Nous disons qu'un objet est partagé s'il est utilisé dans la construction d'un ou plusieurs objets. Cela signifie que l'espace des objets possède une structure de graphe orienté. Un modèle autorise le partage d'objet, s'il fournit la notion d'identité d'objet [KHOS.85].

Le rest du papier est organisé de la manière suivante: la Section 2 présente un bref aperçu sur les concepts utilisés pour le modèle O2, la Section 3 décrit le modèle à objets complexes avec identité d'objet, la Section 4 présente les définitions pour le graphe des types et le graphe de composition d'objets, nous concluons ensuite Section 5.

2. Le modèle O2: un bref aperçu

Nous présentons dans cette section le modèle d'objets complexes utilisé dans O2, qui

est un système de bases de données orienté objet.

Bien que les solutions soient indépendantes de tel ou tel modèle (manipulant des objets complexes), nous utiliserons les notations du système **O2** [LECL.88], [LECL.89], [ADIB.93], [BENZ.90], [BENZ.93].

Dans le système **O2**, deux types de concepts coexistent: les valeurs et les objets. Les valeurs possèdent un type, qui spécifie leur structure, et sont manipulées par des primitives prédéfinies. Les objets ont leur propre identité et encapsulent des valeurs ainsi que des méthodes définies par l'utilisateur. Les objets sont associés à classes.

Les classes sont identifiées par un nom unique. A chaque classe est associé un type ainsi qu'un ensemble d'objets.

Les types sont récursivement définis au moyen des constructeurs ensemble, liste et n-uplet et à partir de types de base tels que **integer**, **string**, etc. Certains types peuvent avoir leur existence propre et n'être pas associés à une classe.

L'exemple suivant illustre la construction de types ainsi que le lien entre classes et types.

```
add   class  Film
             type   tuple ( nom: string
                            année: integer
                            metteur-en-scene: Personne
                            acteurs: set (Personne))

add   class  Personne
             type   tuple ( nom: string
                            pays: string
                            profession: string)
```

L'ensemble de tous les objets (instances) d'une classe est appelé **extension** de la clase. Gérer une extension, pour une classe c, revient à créer, automatiquement, une hiérarchie

d'héritage (ou relation de sous-typage) et les données et traitements (souvent appellés **méthodes**) sont encapsulés. Dans le modèle O2, la notion d'heritage multiple est définie [LECL.88], [LECL.89], [BENZ.93], [ADIB.93].

3. Modèle à objets complexes avec identité d'objet

Dans le système **O2**, chaque objet est identifié par un unique identificateur et représenté par un couple (i, v) où i est un identificateur et v une valeur. Nous rappelons, içi, qu'un objet peut être composé d'autres objets ou des valeurs. Un type est associé à chaque valeur.

Nous avons considéré les ensembles suivants:

- Un ensemble fini de domaines D_1, ... , D_n, $n \geq 1$ (par exemple l'ensemble **Z** de nombres entiers). Notons **D** l'union des domaines D_1, ... , D_n. Nous supposons que les domaines sont disjoints.

- Un ensemble dénombrable et infini **A** , qui s'appelle **univers d'attributs**. Les éléments de **A** sont des noms pour les champs de la structure.

- Un ensemble dénombrable et infini **I** d'**identificateurs**. Les éléments de **I** seront utilisés comme d'identificateurs pour d'objets.

Les valeurs sont construites, récursivement, de la manière suivante:

Définition 1: **Valeurs**

Soit **A** un univers d'attributs et **D** un domaine de valeurs atomiques. Une valeur simple est prise dans l'un de types prédéfinis que sont les entiers (**integer**), les valeurs réelles (**real**), les valeurs logiques (**boolean**), les caractères (**char**) et les chaînes de caractères (**string**).

(i) Tout élément de **D** est une valeur (dite **atomique**).

(ii) Un identificateur d'objet est une valeur.

(iii) Si v_1 , ... , v_n sont des valeurs et a_1 , ... , a_n des attributs de A alors

v = **tuple**(a_1: v_1,..., a_n : v_n) est une valeur structurée de type n-uplet.

(iv) Si v_1 , ... , v_n sont des valeurs distinctes alors

v = **set** (v_1 , ... , v_n) est une valeur structurée de type ensemble.

(v) Si v_1,...v_n sont des valeurs alors v= **list**(v_1,..., v_n) est une valeur structurée de type liste.

Notons **V** l'ensemble des toutes les valeurs.

Les objets sont récursivement construits de la manière suivante:

Définition 2: **Objets**

(i) Un objet est un couple o = (i , v) où i est un élément de I (un identifiant) et v est une valeur.

(ii) Si i , i_1 , ... , i_n sont des identificateurs d'objets et a_1 , ... , a_n sont des noms d'attributs à condition que $a_j \neq a_k$ pour tous les j, k de 1 à n , alors

o = (i , **tuple** (a_1 : i_1 , ... , a_n : i_n)) est un objet à valeur n-uplet.

(iii) Si i , i_1 ,... , i_n sont des identificateurs d'objets, alors

o = (i , **set** (i_1 , ... , i_n)) est un objet à valeur ensemble.

(iv) Si i , i_1 , ... , i_n sont des identificateurs d'objets alors

o = (i , **list** (i_1 , ... , i_n)) est un objet à valeur liste.

(v) Si nous notons **O** l'ensemble d'objets, alors O = I X V.

Il existe un atome particulier qui est noté **nil** et qui dénote un objet indéfini. Par abus de langage, si la valeur d'un objet est un atome, nous disons qu'il s'agit d'un **objet atomique**.

Une valeur composite se construit en utilisant les constructeurs n-uplet (**tuple**) ,

ensemble (**set**) et liste (**list**), appliqués récursivement. Si un objet a une valeur composite nous parlons alors d'**objet composite** ou **complexe**.

La figure 1 présente le diagrame pour la construction de valeurs composites pour des objets.

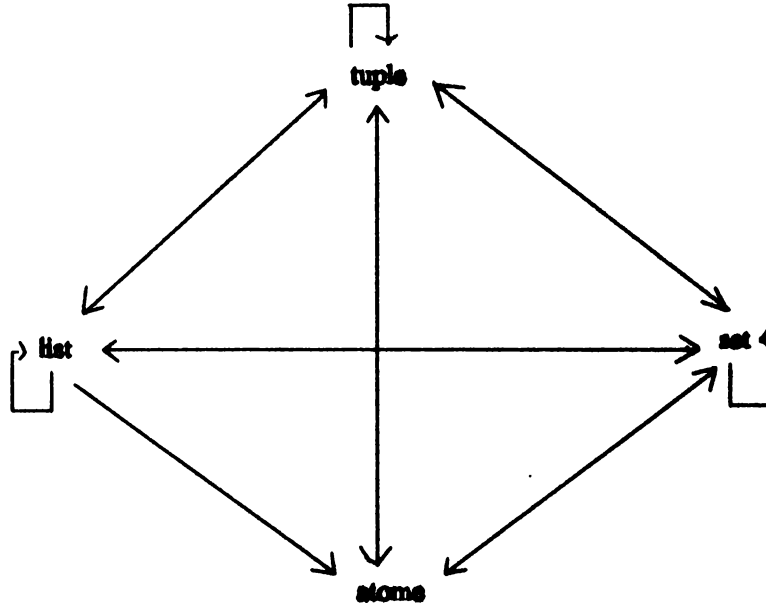

Figure 1. Construction de valeurs composites.

Les types sont récursivement construits de la manière suivante:

Définition 3: Types

Soit **A** un univers d'attributs et **T** un ensemble de types atomiques (**integer, real, boolean, string, char**, etc).

(i) Un atome est un type (dit **atomique**).

(ii) Les noms de classes sont des types.

(iii) Si t_1 , ... , t_n sont des types et a_1 , ... , a_n des attributs de **A**, alors

t = **tuple** (a_1: t_1, ..., a_n : t_n) est un type de structure n-uplet.

(iv) Si t_1 est un type alors t=**set**(t_1) est un type ensemble.

(v) Si t_1 est un type alors t=**list**(t_1) est un type liste.

(vi) Notons T l'ensemble des types.

De manière simple, un type t_1 est considéré comme un sous-type d'un type t_2, si toute instance de t_1 peut aussi être une instance de t_2.

Considérons tout d'abord un cas particulier très simple de sous-typage qui utilise les entiers. Notons n ... m le sous-type du type **integer** correspondant à l'ensemble des entiers de n à m bornes comprises.

On définit récursivement la relation de sous-typage p sur des sous-types de la façon suivante:

Définition 4: Sous-types

(i) n .. m < p .. q si et seulement si $p \leq n$ et $q \geq m$.

(ii) Si t_1,...,t_m,u_1,...,u_n sont des types et a_1, ... , a_m sont des attributs alors

tuple (a_1 : t_1 , ..., a_m : t_m) < **tuple** (a_1 : u_1 , ... , a_n : u_n) si et seulement si

$t_i < u_i$ pour tout i entre 1 et n et $n \leq m$.

(iii) Si t_1 , t_2 sont des types alors **set** (t_1) < **set** (t_2) si et seulement si $t_1 < t_2$.

(iv) Si t_1 , t_2 sont des types alors **list** (t_1) < **list** (t_2) si et seulement si $t_1 < t_2$.

Exemple

Soit le type t_1 avec la définition suivante:

t_1 = **tuple** (
nom: **string**,
adresse: **tuple** (

numéro: **integer**,
rue: **string**,
ville: **string**,
code-postal: **integer**)
téléphone: **integer**)

t_1 est un sous-type de t_2 qui est:

t_2 = **tuple** (
nom: **string**,
adresse: **tuple** (
numéro: **integer**;
rue: **string**,
ville: **string**))

set (**tuple** (nom: **string**, age: **integer**)) est un sous-type de
set (**tuple** (nom: **string**)).

list (**tuple** (nom: **string**, age: **integer**)) est un sous-type de **list** (**tuple** (nom: **string**)).

4. Graphe des types et graphe de composition d'objets

Le type t, associé à une classe c reflète partiellement la hiérarchie de composition des objets de cette classe, c'est-à-dire les liens que possèdent ceux-ci avec d'autres objets [ADIB.93], [BENZ.90], [BENZ.93].

Un type t peut être représenté par un graphe orienté étiqueté, dont la définition suit:

Définition 5: **Graphe de type GT (t)**

(i) GT (t) = (N_t , E_t) où:

(ii) N_t est l'ensemble des sommets étiquetés. Chaque sommet représente un type et est étiqueté au moyen de $\beta : N_t \rightarrow T \cup \{$ **tuple** , **set** , **list** $\}$ et $\alpha : N_t \rightarrow C$ où C est l'ensemble des clases.

(iii) Si t est associé à la classe identifié par c alors $t \in N_t$ et $\alpha(t) = c$. Sinon, α (t) n'est pas définie.

(iv) Si t est un atome alors $t \in N_t$ et β (t) = atome.

(v) Si $t_1 ... t_n \in N_t$ et t = **tuple** ($a_1 : t_1 , ... , a_n : t_n$) alors $t \in N_t$ et $\beta(t)$ = **tuple**.

(vi) Si $t_1 \in N_t$ et t=**set**(t_1) alors $t \in N_t$ et $\beta(t)$=**set**.

(vii) Si $t_1 \in N_t$ et t=**list**(t_1) alors $t \in N_t$ et $\beta(t)$=**list.**

(viii) E_t est l'ensemble des arcs orientés et étiquetés au moyen de $\gamma : E_t \rightarrow A$ où A est l'ensemble des noms d'attributs.

(xi) Si t = **tuple**($a_1 : t_1, ..., a_n : t_n$) alors $(t, t_i) \in E_t$ et $\gamma (t, t_i) = a_i$ pour tout i de 1 à n.

(x) Si t = **set** (t_1) alors $(t, t_1) \in E_t$ et $\gamma (t, t_1)$ n'est pas définie.

(xi) Si t = **list** (t_1) alors $(t, t_1) \in E_t$ et $\gamma (t, t_1)$ n'est pas définie.

La totalité des liens entre tous les types présents dans le système est donnée par le **Graphe des Types GT**.

Définition 6: **Graphe des types GT**

(i) **GT** = $\cup_{t \in T}$ **GT** (t) = ($\cup_t N_t$, $\cup_t E_t$).

La figure 2 illustre cette définition.

La relation d'héritage entre clases induit une relation de sous-typage entre types [LECL.89], [ADIB.93], [BENZ.93]. Dû à la caractérisation syntactique de la relation de sous-typage, si t_1 est un sous-type de t'_1 et s'il existe un arc (t'_1 , t'_2) étiqueté par l dans GT alors il existe un arc (t_1 , t_2) étiqueté par l dans GT avec t_2 sous-type de t'_2.

Définition 7: **Graphe de type avec héritage GTH (t)**

On ajoute à la définition 5 la ligne suivante:

(xii) Si t_1 est un sous-type de t_2 alors (t_1 , t_2) $\in E_t$ et γ (t_1 , t_2) n'est pas définie. Cet arc en pointillé représente le lien d'héritage.

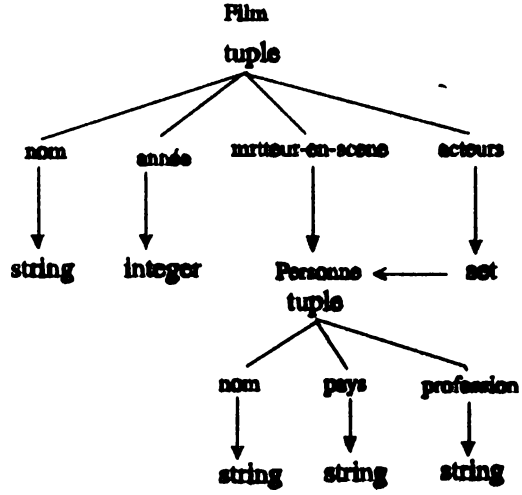

Figure 2. Graphe des types GT.

Définition 8: Graphe des types avec héritage GTH

Le graphe des types avec héritage GTH est défini comme suit:

(i) $\mathbf{GTH} = \cup_{t \in T} \mathbf{GTH}\ (t)$.

Exemple

Soient les définitions suivantes:

add class Film-de-Pub **inherit** Film
type tuple (metteur-en-scene: Publicitaire)

add class Publicitaire **inherit** Personne
type tuple (agence: **string**)

add class Marin **inherit** Personne
type tuple (bateau: **string**)

La figure 3 présente le graphe des types avec héritage.

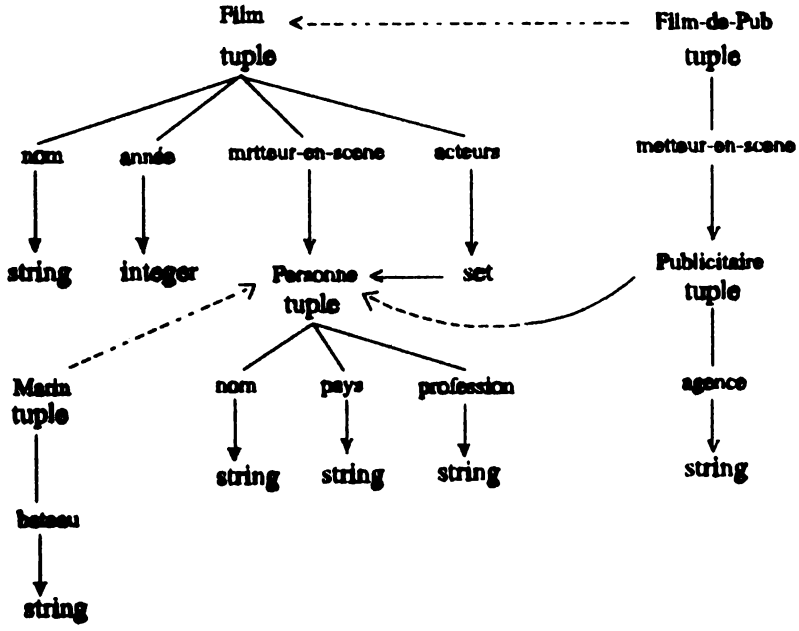

Figure 3. Graphe des types avec héritage.

Etant donné un ensemble d'objets, à nouveau, les liens entre ces objets pourront être représentés par un graphe orienté. Nous appelons ce graphe: **Graphe de Composition d'Objets GCO.**

La définition d'un tel graphe suit:

Définition 9: Graphe de composition d'objets GCO

Soit I un ensemble d'identificateurs d'objets. Le GCO est défini pour les objets/valeurs par:

(i) $G_I = (V_I , E_I)$ où:

(ii) V_I est l'ensemble des nœuds chaque nœud représente une valeur et est étiqueté

par $\alpha : V_I \rightarrow \mathbf{I}$ et $\beta : V_I \rightarrow \mathbf{V} \cup \{\textbf{tuple}, \textbf{set}, \textbf{list}\}$.

(iii) Si v est associée à l'objet identifié par i alors $v \in V_I$ et $\alpha(v) = i$.

(iv) Si v est une valeur atomique alors $v \in V_I$ et $\beta(v)$=valeur.

(v) Si $v_1 ... v_n \in V_I$ et v = **tuple** $(a_1 : v_1 , ... , a_n : v_n)$ alors $v \in V_I$ et $\beta(v)$ = **tuple**.

(vi) Si $v_1 ... v_n \in V_I$ et v = **set** $(v_1 ... v_n)$ alors $v \in V_I$ et $\beta(v)$ = **set**.

(vii) Si $v_1 ... v_n \in V_I$ et v = **list** $(v_1 ... v_n)$ alors $v \in V_I$ et $\beta(v)$ = **list.**

(viii) E_I est l'ensemble des arcs étiquetés au moyen de $\gamma : E_I \rightarrow \mathbf{A}$, où **A** est l'ensemble des noms d'attributs.

(ix) Si v = **tuple** $(a_1 : v_1 , ... , a_n : v_n)$ alors

$(v, v_k) \in E_I$ et $\gamma(v, v_k) = a_k$, pour tout k de 1 à n.

(x) Si v = **set** $(v_1 , ... , v_n)$ alors $(v, v_k) \in E_I$

et $\gamma(v, v_k)$ n'est pas définie, pour tout k de 1 à n.

(xi) Si v = **list** $(v_1 , ... , v_n)$ alors $(v, v_k) \in E_I$

et $\gamma(v, v_k)$ n'est pas définie, pour tout k de 1 à n.

Exemple

Soit les objets :

$o_1 = (i_1$, **tuple** (Nom : Ionesco, Conjoint : i_2 , Age : 35 , Enfants : **set** $(i_3)))$

$o_2 = (i_2$, **tuple** (Nom : Magda , Conjoint : i_1 , Age : 33 , Enfants : **set** $(i_3)))$

$o_3 = (i_3$, **tuple** (Nom : Chrétien , Conjoint : **nil** , Age: 10, Enfants : **set (nil)))**.

Si $O = \{o_1, o_2, o_3\}$, alors dans la figure 4 nous présentons le graphe **GCO** correspondant.

Le **GCO** peut être vu comme une "instanciation" du graphe des types **GT**.

L'extension d'une classe c est une valeur de type **set** (c), les nœuds correspondant

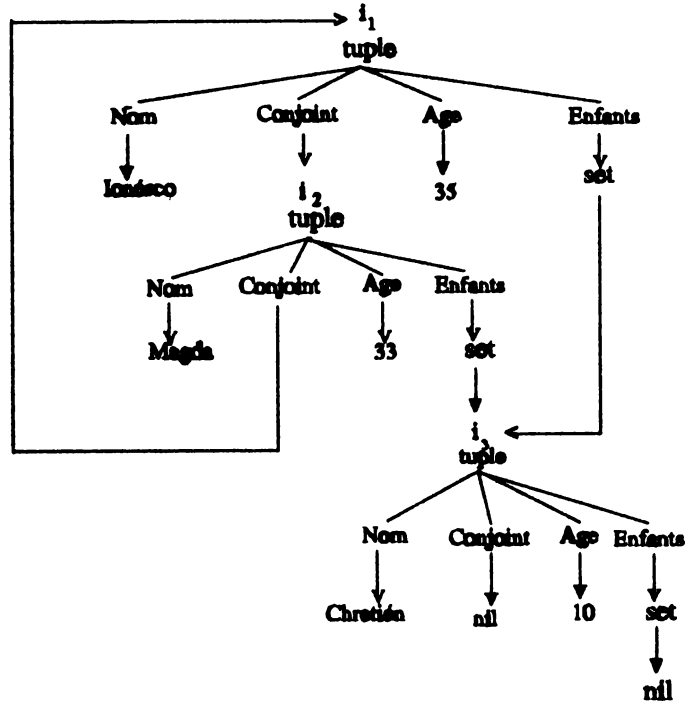

Figure 4. Graphe de composition d'objets

à ces objets seront présents dans le **CGO**. Le **CGO** décrit les connexions inter-objet d'un ensemble d'objets donné.

5. Conclusion

Nous avons présenté ici les concepts d'un modèle à objets complexes avec identité d'objet.

Dans les **SGBDOO** le concept d'objet est fondamental. Nous avons défini le concept d'objet complexe à partir d'objets atomiques et de constructeurs qui s'appliquent récursivement indépendamment du type des objets.

On représent souvent les liens entre les objets au moyen d'un graphe.

Nous avons construit le graphe des types GT, le graphe des types avec héritage GTH et le graphe de composition d'objets GCO.

Le graphe des types et le graphe de composition d'objets jouent un rôle prépondérant dans la mise en œuvre des stratégies de regroupement d'objets sur disque dans un système de bases de données orienté-objet.

BIBLIOGRAPHIE

[ABIT.87] S. ABITEBOUL, C. BEERI : On the power of languages for the manipulation of complex objects. In Proceedings of theInternational Workshop on Theory and Applications of Nested Relations and Complex Objects, Darmstadt,1987.

[ABIT.88] S. ABITEBOUL, S. GRUMBACH : Col : A logic - based Language for Complex Objects. In Proc. of EDBT International. Conf., 1988.

[ABIT.89] S. ABITEBOUL, P. KANELLAKIS : Object Identity as Query Language Primitive, In Proc. of the ACM SIGACT- SIGMOD Symp.on Principles of Database Systems, juin, 1989.

[ADIB.93] M. ADIBA, C. COLLET : Objets et Bases de Données. Le SGBD O2,Hermès, Paris, 1993.

[ATKI.87] M. ATKINSON, P.BUNEMAN : Types and Persistence in Database Programming Languages, ACM Computing Surveys, June, 1987.

[BANC.86] A. BANCILHON, S. KHOSHAFIAN : A calculus for Complex Objects, ACM PDDS Conference 1986.

[BENZ.89] V. BENZAKEN : Regroupement d'objets sur disque dans un système de bases de données orienté-objet, thèse de doctorat, Paris, 1990.

[BENZ.93] V. BENZAKEN, A. DOUCET : Bases de Données orientées objet. Origines et principes, Armand Colin, 1993.

[CODD.70] E.F.CODD : A Relational Model of Data for Large Shared Data Banks, CACM Vol 13 No 6, June, 1970.

[HUMB.91] M. HUMBERT : Les Bases de Données, Hermès, 1989.

[KHOS.86] S. KHOSHAFIAN, G. COPELAND : Object Identity, OOPSLA 86, Portland Oregon, Sept. 1986.

[LECL.88] C. LECLUSE, P. RICHARD, F. VELEZ : O2, an Object- Oriented Data Model, ACM 3/1988.

[LECL.89] C. LECLUSE, P. RICHARD : Modeling Complex Structures in Object-Oriented Databases, Prooc. of the ACM PODS Conference, Philadelphie, 1989.

[MAKI.77] A. MAKINOUCHI : A Consideration on Normal Form of Not-Necessarly-Normalized Relation in the Relational Model of Data, ACM VLBD Tokio, Japan, 1977.

[SCHE.82] H. SCHEK, G. JAESCHKE : Remarks on the Algebra of Non First Normal Form Relations, ACM PODS Los Angeles, 1982.

[RICH.89] P. RICHARD : Des Objets Complexes aux Bases de Données Orientées - objets, Thèse de doctorat, Paris, 1991.

[ZANI.85] C.ZANIOLO : The Representation and Deductive Retrieval of Complex Objects, VLBD, Stockholm, August, 1985.