Modélisation et recherche basées sur le contenu d'objets complexes : le système EMIR

Youssef Lahlou

Université Henri Poincaré – Nancy I

Département de Formation Doctorale en Informatique — École Doctorale IAE + M

Modélisation et recherche basées sur le contenu d'objets complexes. Le système EMIR.

THÈSE

présentée et soutenue publiquement le 14 Juin 1996

pour l'obtention du

Doctorat de l'Université Henri Poincaré – Nancy I

(Spécialité Informatique)

par

Youssef LAHLOU

Composition du jury

Président :	Monique GRANDBASTIEN, Professeur à l'Université Henri Poincaré
Rapporteurs :	Claude CHRISMENT, Professeur à l'Université Paul Sabatier, Toulouse Amedeo NAPOLI, Chargé de Recherche au CNRS, CRIN, Nancy NGUYEN Gia Toan, Directeur de Recherche à l'INRIA, Grenoble
E	T, Professeur à l'Université de Nancy 2
D	UADDIB, Maître de Conférences à l'Université Henri Poincaré

Centre de Recherche en Informatique de Nancy

Résumé

Dans cette thèse, nous présentons le modèle EMIR (Extended Model for Information Retrieval) destiné à modéliser des objets complexes à forte structure individuelle.

Dans le domaine des bases de données, l'étape conceptuelle est très lourde. Elle consiste à définir le schéma conceptuel de l'application, définition en intension de la structure des objets destinés à être gérés par la base. Dans certaines nouvelles applications (notamment documentaires), cette étape s'avère très restrictive, dans la mesure où les objets manipulés n'ont pas toujours une structure prévisible *a priori* par le concepteur de la base.

Le but du modèle EMIR est de fournir un environnement relâchant la contrainte du schéma conceptuel en autorisant les objets de la base à disposer de structures individuelles, qui s'ajoutent aux strutures prévues dans le schéma conceptuel.

Les objets du modèle ont une structure tri-partite : attributs, objets composants, liens entre composants. Le schéma conceptuel est formé de catégories d'objets (ayant aussi une structure tri-partite) et de relations sémantiques pouvant lier des objets. Le lien entre un objet et une catégorie est baptisé *lien de réalisation.* Il impose à l'objet d'implanter une structure issue de la catégorie, et l'autorise à posséder des structures individuelles supplémentaires.

Nous proposons des solutions adaptées au modèle, pour des questions classiques des bases de données telles que le maintien de l'intégrité et l'interrogation.

L'ensemble des propositions a été validé par un prototype réalisé en Smalltalk-80 et a été expérimenté sur des applications réelles (modélisation d'images scannerisées et de projets architecturaux).

Mots-clés: Systèmes d'information, Bases de données orientées objet, Recherche d'information, Instanciation flexible

Abstract

This thesis introduces the Extended Model for Information Retrieval (EMIR) whose objective is the modelling of complex objects with strong individual features.

Conceptual modelling of databases is a heavy task. Abstracting object structures into classes is not always easy. In some cases (e.g. Information Retrieval from document bases), it is rather unfeasible because of the individual characteristics of each object (document).

EMIR aims to relax the conceptual schema constraint by allowing objects to have individual features to be added to those abstracted in some related conceptual entities.

EMIR objects have a tri-partite structure: attributes, component objects and links between components. The conceptual schema is made of object categories (also described in the same tri-partite fashion) and semantic relationships. The link between an object and a category is baptized *realization link.* The object implements a structure inspired by the category and is allowed to have individual added features.

We propose specific solutions to classical database issues such as integrity management and querying.

A prototype implementing the approach has been realized in Smalltalk-80. It has been experimented in two real applications: semantic modelling of images and architectural projects description.

Remerciements

Je remercie Madame Monique GRANDBASTIEN de m'avoir fait l'honneur de présider le jury.

Je remercie Messieurs Claude CHRISMENT et NGUYEN Gia Toan d'avoir accepté d'être les rapporteurs de cette thèse, de l'intérêt qu'ils y ont porté et des commentaires constructifs qu'ils ont apportés.

Je remercie Monsieur Amedeo NAPOLI d'avoir accepté d'être le rapporteur interne de cette thèse, des discussions que nous avons eues et de l'intérêt qu'il a porté à nos travaux.

Je remercie Monsieur Noureddine MOUADDIB d'avoir dirigé cette thèse et de l'avoir fait dans un esprit de coopération, plutôt que de tutelle, chose à laquelle j'ai été très sensible.

Je remercie Madame Odile FOUCAUT d'avoir accepté de faire partie du jury et des conseils qu'elles m'a prodigués tout au long de mon stage doctoral.

Je remercie Madame Marion CREHANGE de m'avoir chaleureusement accueilli au sein de l'ancienne équipe EXPRIM, de ses commentaires judicieux sur mon travail et de l'amitié qu'elle m'a témoignée.

Je remercie Monsieur Nacer BOUDJLIDA pour son temps et ses remarques constructives, durant la préparation de la soutenance de cette thèse.

Je souhaite beaucoup de chance aux deux nouveaux groupes OCEAN et EXPRIM du CRIN, auxquels je suis lié d'une manière ou d'une autre.

Je remercie Madame Odile THIERY et Monsieur Franck DI SANTOLO d'avoir gentiment et patiemment relu mon manuscrit. Leurs remarques en ont significativement amélioré le contenu.

Merci Nathalie et Pascal pour la merveilleuse ambiance au bureau, pour vos encouragements et pour ta relecture soignée de mon manuscrit, Nath.

Je tiens aussi à remercier mes amis du labo (qui se reconnaitront) avec lesquels j'ai passé d'agréables moments pendant toutes ces années.

A Hind

A mes parents

A Mohammed RAKID

Table des matières

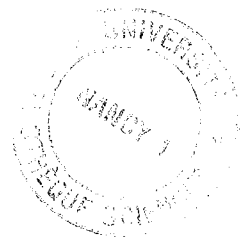


Table des figures

Introduction

Depuis la naissance de la gestion de bases de données en tant que discipline informatique à part entière, dans les années 60, les modèles de données n'ont cessé d'évoluer.

L'âge d'or des modèles hiérarchique et réseau qui se situe dans les années 70 a été marqué par l'explosion des programmes COBOL dans les entreprises, friandes de l'informatisation galopante et garante d'une accélération (sans pareil jusqu'alors) des traitements.

La déferlante relationnelle des années 80 a chassé "le hiérarchique" et "le réseau" du cœur des chercheurs et des chefs de projets, mais pas des fichiers COBOL, développés par millions de lignes durant la décennie d'avant. Une quantité insoupçonnée de ces programmes continue à hanter les disques durs des entreprises anciennement informatisées.

L'atout majeur du relationnel était l'indépendance qu'il assurait entre les données et les traitements, en particulier à travers son mode d'interrogation déclaratif et conversationnel, qui dispensait d'écrire, pour chaque requête, tout un programme pour la réaliser. Auparavant, avec le mode procédural, il était fréquent qu'un utilisateur soumette sa requête à l'administrateur de la base de données et attende quelques jours (voire quelques semaines), le temps que celui-ci écrive, débogue et exécute le programme de recherche correspondant !

Le relationnel avait cependant ses défauts ; le principal étant la célèbre "impedance mismatch", provenant de la différence entre la structure des données dans la base et celles des programmes qui la manipulent. Les années 90 ont alors donné lieu à la naissance de l'approche orientée objet, où données et programmes sont encapsulés dans une structure unifiée : l'objet. Un objet contient des données, mais aussi les méthodes qui permettent d'y accéder.

La transition entre le relationnel et l'objet ne s'est pas faite du jour au lendemain. Plusieurs approches, dites sémantiques ou d'objets complexes, ont assuré le relais.

L'approche objet se trouve être aujourd'hui l'avenir des bases de données. Elle a d'ailleurs conduit l'ouverture de cette discipline à des domaines d'application très divers : CAO, multimédia, bases de données documentaires (textes, images), bases de données géographiques, ...

Cependant, si cette approche est garante d'une diversité intéressante dans les types d'objets manipulés, chaque domaine possède ses propres particularités et nécessite un type de modélisation et des traitements appropriés.

Ainsi, les bases de données documentaires (et notamment les bases d'images auxquelles nous nous sommes intéressés plus particulièrement) nécessitent des traitements particuliers au niveau de la modélisation du contenu des documents (au delà de leur structure) et de l'interrogation.

En effet, même si les bases de données orientées objet permettent de modéliser efficacement des structures d'objets complexes, et d'effectuer des requêtes à partir des primitives de structuration (les attributs ou variables d'instance), elles ne sont pas très adaptées à la modélisation et l'interrogation des bases documentaires qui sont, pour leur part, basées sur le contenu des documents plus que sur leur structure.

Nous pensons que cette inadéquation résulte, en grande partie, du fait que le modèle de données utilisé dans les bases de données orientées objet est un modèle de classes. En effet, le schéma d'une base de données orientée objet est constitué d'un ensemble de classes reliées entre elles par des liens de généralisation, de composition, ... Chaque objet

de la base appartient à une classe et se voit ainsi conférer une structure précise qu'il doit implanter sans la transgresser.

Cette hypothèse est inadaptée aux bases de données documentaires, où les structures des objets sont très diversifiées. Dans une base d'images, par exemple, chaque image fait référence à des objets différents. Il est donc difficile de prévoir à un niveau conceptuel (dans le schéma de la base) une structure fixe que satisfait exactement chaque image.

La thèse que nous développons dans ce travail est que le rapprochement entre les bases de données et la recherche d'information dans les bases documentaires est possible au travers d'un modèle de données dédié à la modélisation du contenu individuel des objets, plutôt qu'à l'abstraction des objets en entités conceptuelles (classes).

Le modèle que nous proposons (intitulé EMIR) assouplit la contrainte d'un modèle de classes au niveau des objets. Il s'inspire de travaux effectués dans le domaine des langages de programmation par objets, autres que les langages de classes.

Dans le monde des langages orientés objet, le modèle de classes n'est qu'un paradigme parmi d'autres (modèles de frames, de prototypes, d'acteurs, ...). Il a été adopté dans le domaine des bases de données parce que la notion de classe fournit une primitive pour l'accès groupé à toutes ses instances. Cette fonctionnalité est nécessaire pour l'interrogation des bases de données, car la quantité d'objets manipulés est, par définition, importante.

Le modèle des prototypes adopte une approche intéressante qui confère aux objets une autonomie de structure et de comportement qui dispense de la notion de classe. Cette approche est intéressante pour modéliser des documents, mais elle ne fournit pas la primitive d'accès groupé aux objets, nécessaire pour une recherche d'information efficace.

Notre modèle s'inspire de travaux sur des modèles hybrides à mi-chemin entre classes et prototypes. Nous proposons le concept de *catégorie*, qui fixe un certain nombre de propriétés communes à un ensemble d'objets, et fournit la primitive d'accès groupé. Les catégories sont organisées en hiérarchie de généralisation. Tout objet relié à une catégorie dispose néanmoins d'une structure individuelle supplémentaire à la structure commune. Le lien entre un objet et sa catégorie est baptisé *lien de réalisation.*

Afin de mieux s'adapter au domaine des bases d'images, qui constitue son principal champ d'expérimentation, notre modèle décrit ses objets selon trois facettes : description (liste de valeurs d'attributs), composition (liste d'objets composants) et topologie (liste de liens entre composants).

Cette modélisation tri-partite et la différence de structure entre un objet et sa catégorie occasionne des traitements originaux pour maintenir l'intégrité de la base ou encore pour l'interroger.

Ce document est divisé en quatre chapitres et trois annexes.

Dans le chapitre 1, nous présentons le traditionnel "état de l'art". Nous y détaillons le cadre général de notre étude, en passant en revue les principales approches qui la motivent.

Après avoir décrit succinctement ce que nous entendons par "objet complexe", nous retraçons l'évolution des modèles de données, pour la gestion d'informations complexes dans les bases de données. Nous présentons ensuite les courants de programmation par objets qui nous ont inspiré, avant de présenter les particularités des systèmes d'information documentaires.

Le chapitre 2 expose en détail le modèle de données EMIR. Les concepts du modèle (catégories, objets, lien de réalisation ...) sont décrits informellement, puis formellement. Nous mettons l'accent en particulier sur l'aspect topologique, décrivant des relations internes à un objet, qui expriment l'agencement de sa composition.

Une partie de ce chapitre est consacrée à une discussion sur la nature des liens entre un objet et ses objets composants.

Dans le chapitre 3, nous proposons des mécanismes pour le maintien de l'intégrité des données dans EMIR, ainsi qu'un langage de requêtes adapté au modèle.

Un premier aspect de l'intégrité, appelé intégrité référentielle, assure la cohérence de la base vis-à-vis de la sémantique associée aux constructeurs du modèle. Nous avons, pour cela, défini les invariants d'une base de données EMIR, puis nous avons spécifié formellement un ensemble d'opérations possibles, qui respectent ces invariants.

Un second aspect de l'intégrité des données est lié aux applications du modèle. Nous avons défini un langage de spécifications de contraintes d'intégrité sémantiques qui tient compte de la définition tri-partite des objets EMIR ainsi que du lien de réalisation.

Par ailleurs, le langage de requêtes est capable d'exprimer des requêtes aussi bien sur la structure des objets (requêtes navigationnelles) que sur leur contenu (requêtes basées sur le contenu).

Nous présentons finalement le prototype réalisé pour valider notre approche, dans le chapitre 4, en fournissant quelques détails d'implantation.

Dans ce chapitre, nous évoquons en particulier deux applications réelles ayant servi à valider les concepts que nous avançons : une application décrivant une base d'images et une application pour modéliser des projets architecturaux.

Chapitre 1

Informations complexes et modèles de bases de données

La nature associative du raisonnement humain et de la perception humaine de l'environnement de manière générale, rend toute information complexe, puisqu'irrémédiablement liée à une multitude d'autres informations, qui en font un tout à la fois dissociable et indissociable ! En effet, toute entité du monde a une existence propre indépendamment de ce qui l'entoure ; mais son existence ne peut avoir de sens sans ses interactions avec d'autres entités.

Ainsi, une voiture stationnée dans la rue est perçue comme une entité du monde réel, ayant une existence propre et une certaine autonomie. Cependant, il suffit qu'elle soit dans sa rue pour que celui qui l'observe fasse immédiatement le lien entre elle et le voisin à qui elle appartient ("C'est la voiture de M. Untel !") ; il suffit d'un coup d'oeil sur la plaque minéralogique pour faire le lien entre elle et le pays d'où elle provient ("C'est une voiture syldave !"). Dans tous les cas, l'information véhiculée par l'objet voiture ne se limite jamais à l'objet en lui-même, mais le transgresse par des références à d'autres, qui lui sont liés d'une manière ou d'une autre, et qui eux-même ont une existence propre. Ces références peuvent elles-mêmes apporter de nouvelles unités de sens à la voiture en question : "M. Untel s'occupe bien de ses voitures !", "Les voitures syldaves sont solides !".

La complexité d'une entité ne provient pas uniquement de sa structure, mais aussi et surtout de ses références à d'autres entités, sans lesquelles son existence ne peut être justifiée.

Toute information est donc complexe ; mais selon les besoins, elle peut être ramenée à une information simple, en n'en considérant que les composants utiles au traitement auquel elle est destinée. C'est d'ailleurs ce qui s'appelle "faire de la modélisation". L'entité (du monde réel) qu'on modélise est distincte de l'entité (d'un monde virtuel) qui résulte de cette modélisation.

L'histoire des modèles de données dans le domaine des bases de données [DLR91], peut être vue comme la succession d'un certain nombre d'étapes, reflétant chacune l'état d'avancée, d'une part des applications des bases de données (et donc des procédés de modélisation) et d'autre part du matériel. On peut résumer cette évolution en disant qu'on est en train de passer d'une époque où *les applications suivent les modèles* (hiérarchique, réseau, relationnel), à une époque où *les modèles suivent les applications* (approches sémantiques, approche objet). Il n'y a nul doute que l'évolution du matériel a joué et joue un rôle crucial dans cette évolution. Elle a pour effet d'élargir les champs d'application et les techniques de modélisation.

Dans cette partie, une exploration des différentes caractéristiques d'un objet complexe est menée dans la section 1.1. L'évolution des modèles de données pour la gestion d'objets complexes est exposée dans la section 1.2. Nous y détaillons en particulier les extensions du modèle relationnel de données dans 1.2.1, quelques modèles sémantiques de données dans 1.2.2, ainsi que quelques modèles de bases de données orientées objet dans 1.2.3. Nous effectuons dans 1.2.4 "un petit tour" du côté des langages de programmation par objets. Enfin, nous évoquons dans 1.3 les particularités des bases de données documentaires (textuelles, d'images, multimédias) et de la Recherche d'Information basée sur le contenu.

1.1 Concepts généraux

Comme nous l'avons remarqué plus haut, tout objet du monde réel est complexe. Cependant, cette considération philosophique mise à part, la réalité pratique conduit à contourner cette complexité en ne considérant que les composantes informatives de l'objet, dignes d'intérêt dans le contexte.

Ceci ne résout pas dans tous les cas le problème de la complexité, puisque la diversification des champs d'intérêt et la prise en compte de plus en plus d'information, obligent les modèles de données à rendre compte du maximum de "phénomènes" possibles. Le besoin s'est alors fait ressentir de formaliser les approches de modélisation et d'isoler les constructeurs sémantiques essentiels à de telles informations complexes.

On peut définir informellement un objet complexe [HUL89, DLR91] comme étant un objet avec des caractéristiques intrinsèques, et nouant des liens avec d'autres objets (par opposition, un objet simple a seulement des caractéristiques intrinsèques). Un ensemble d'objets ayant les mêmes caractéristiques traduit une classe d'objets, et ces caractéristiques communes traduisent un type d'objets sous-jacent. Des liens peuvent exister entre les classes d'objets, traduisant des propriétés portant sur leurs types et/ou sur l'ensemble des objets qui en font partie.

Une notion sous-jacente, incontournable dans les modèles de données pour objets complexes, est la notion d'identité d'objet [KC86, COD79], qui différencie un objet d'un autre, indépendamment de leurs valeurs respectives. Cette notion traduit la séparation entre un objet en tant que référence à une entité, et sa valeur en tant que construction mathématique.

Dans la suite, nous allons rappeler certaines caractéristiques des objets complexes, de manière informelle d'abord, et formelle ensuite, en les illustrant par des exemples.

1.1.1 Identité d'objet

"Ces deux jumeaux se ressemblent trait pour trait, on dirait qu'il s'agit de la même personne !"

Cette phrase simple montre que la notion d'identité d'objet est essentielle, puisqu'elle permet de distinguer un objet des autres, même d'un autre objet ayant exactement les mêmes caractéristiques (les deux jumeaux ne sont pas une même personne).

Un objet O est donc modélisé par un identificateur unique $i(O)$ invariant pendant toute la durée de vie de l'objet, et une valeur $v(O)$ sujette à d'éventuelles modifications. La valeur $v(O)$ sert, en particulier, à désigner les liens entre l'objet O et ses différents objets reliés.

Ainsi, l'objet V_1, représentant une voiture, aura un identificateur $i(V_1)$ qu'on peut assimiler à V_1 (car en l'appelant V_1, on lui attribue déjà un identificateur), ainsi qu'une valeur $v(V_1)$ qui traduira par exemple les liens suivants : entre V_1 et son propriétaire (une

personne), entre V_1 et sa roue avant gauche, entre V_1 et la classe *Voiture* à laquelle il appartient.

1.1.2 Agrégation et groupement

"Il habite au numéro 264 de l'avenue de la plage, à Szôhod, en Bordurie."
"Le roi Ottokar 1er a trois fils : Ottokar 2nd, Choffar et Plummar."

L'agrégation [SS77] permet de construire un objet comme étant la juxtaposition d'autres objets, chacun y jouant un rôle précis. Par exemple, une adresse est obtenue par la juxtaposition d'un nombre (le numéro) et de trois chaînes de caractères (le nom de l'artère, le nom de la ville et le nom du pays).

Le groupement permet de construire un objet comme un ensemble d'autres objets. Par exemple, les héritiers d'une personne forment un ensemble de personnes.

Il y a donc trois types d'objets : des objets *simples* dont la valeur est une valeur simple (nombre entier, réel, chaîne de caractères, ...), des objets *tuples* dont la valeur est un n-uplet de valeurs et des objets *ensembles* dont la valeur est un ensemble de valeurs.

Ceci nous permet de définir formellement la notion de type d'objet : étant donné un ensemble de types dits "terminaux" Ter (en pratique, ce sont les entiers, les chaînes de caractères, ...), un type T est défini récursivement par les trois règles suivantes :

- $T \in Ter$, est un type.
- Si les T_i sont des types ($1 \leq i \leq n$), alors $T = T_1 \times ... \times T_n$ représente le type obtenu par agrégation des types T_i.
- Si T est un type, alors $T' = set(T)$ représente le type obtenu par groupement du type T.

Des variantes intéressantes du constructeur *set* représentent des ensembles ordonnés (constructeur *list*) ou des ensembles à répétition d'éléments (constructeur *bag*).

Exemple :
Le type d'un objet représentant une adresse est $E \times S \times S \times S$ (E pour le numéro, S pour la rue, S pour la ville et S pour le pays) ; E et S désignant respectivement les types "Nombre entier" et "Chaîne de caractères".

De plus, si une personne peut avoir plusieurs adresses, le type de l'objet qui représente les adresses d'une personne est obtenu en appliquant le constructeur *set* au produit cartésien précité.

1.1.3 Classes d'objets

"C'est un oiseau ! Tous les oiseaux ont un bec d'une certaine longueur, des pattes et un plumage d'une certaine couleur."

La classification est un processus naturel chez l'être humain. Tout objet de l'environnement est perçu comme tel, mais aussi (et inexorablement) comme faisant partie d'un groupe d'objets dont il partage certaines caractéristiques.

Une classe d'objets est un groupe d'objets ayant le même type. Les deux notions de classe et de type sont indissociables. Un type n'a de légitimité qu'à travers les individus qui le représentent ; et inversement un groupe d'individus ne mérite le nom de groupe que grâce aux points communs qui lient ses membres (dont le premier est ... d'appartenir au groupe !).

Ainsi chaque objet O contient dans sa partie valeur $v(O)$ la référence à une classe $c(O)$, à laquelle il "appartient".

Une classe C est alors définie comme étant un objet dont la valeur $v(C)$ comporte en particulier deux parties : l'ensemble $I(C)$ des objets qui lui "appartiennent" et son type associé $t(C)$. Si un objet O "appartient" à la classe C, le type de O est $t(O) = t(c(O))$.

Le lien qui permet de lier un objet O à sa classe $c(O)$ est ce qu'on appelle le lien d'*instanciation*. On dira que O est une instance de $c(O)$.

Exemple :
Soit la classe $Entreprise$ ayant pour instances les objets $E_1, ...E_n$. Donc, $I(Entreprise) = \{E_1, ..., E_n\}$.
Soit le type $TE = t(Entreprise)$.
Si chaque entreprise est représentée par un nom, une adresse, un chiffre d'affaire, une date de création, et un chef d'entreprise, alors : $TE = S \times t(Adresse) \times E \times t(Date) \times t(Personne)$, où $Adresse$, $Date$ et $Personne$ sont des classes.
Le type associé aux instances E_i est TE : $t(E_i) = t(c(E_i)) = t(Entreprise) = TE$.

1.1.4 Généralisation / spécialisation

"Tous les hippopotames sont des mammifères."

La notion de sous-classe est une caractéristique importante du mécanisme de classification qu'opère l'être humain dans la perception des entités de son environnement.

On dit qu'une classe C est une *généralisation* (ou "super-classe") d'une classe SC si toute instance de la classe SC est obligatoirement instance de la classe C. On dit aussi que SC est une *spécialisation* (ou "sous-classe") de C.

Ainsi l'ensemble des instances d'une classe contient la réunion des ensembles d'instances de ses sous-classes.

Au delà de l'équivalence des deux termes "généralisation" et "spécialisation" au niveau des classes, deux mécanismes différents du point de vue de l'approche de modélisation sont à considérer [DLR91, AH87] :

- Le mécanisme de *généralisation* est le fait de regrouper plusieurs classes SC_i, $i \in [1,n]$ en une seule classe C, parfois appelé *classe abstraite*. Dans ce cas $I(C) = \bigcup\{I(SC_i), i \in [1,n]\}$. On part donc des sous-classes. Ainsi, la généralisation des deux classes $Voiture$ et $Camion$ donnera la classe $Véhicule$, dont les seules instances seront soit des voitures soit des camions.

- Le mécanisme de *spécialisation* permet, au contraire, de distinguer parmi les instances d'une classe C, quelques unes répondant à certains critères supplémentaires, en les regroupant en sous-classes SC_i, $i \in [1,n]$. Dans ce cas $I(C) \supseteq \bigcup\{I(SC_i), i \in [1,n]\}$. On part donc de la super-classe. Ainsi, la spécialisation d'une classe *mammifère* en une sous-classe *hippopotame*, selon des caractéristiques précises, ne signifie pas que tous les mammifères sont des hippopotames.

Le lien qui permet de lier une sous-classe SC à une super-classe C est appelé *lien IS-A (est-un)*.

Exemple :
Si on reprend l'exemple de la classe *Entreprise*, définie précédemment, on peut en définir une spécialisation, *Multinationale*.
$I(Multinationale) = \{M_1, ..., M_p\}$, avec $p \leq n$ et $\forall i \in [1,p], \exists j \in [1,n], M_i = E_j$;
d'où, $I(Multinationale) \subseteq I(Entreprise)$.

1.1.5 Association

"Isaac est sous l'arbre. Archie est dans son bain."

L'association, composante essentielle de la perception humaine de l'entourage, permet normalement de lier deux ou plusieurs entités sémantiques. Dans certains cas, l'existence même d'un objet se "résume" à ses associations : dans la locution "La chèvre de M. Seguin", l'existence de la chèvre est conçue à travers le fait qu'elle appartient à M. Seguin.

Les associations entre objets peuvent avoir des dénotations différentes. Elles peuvent traduire des liens spatiaux [BRI92] (dans ce cas elles améliorent la localisation spatiale des objets) ; elles peuvent traduire des liens de composition physique, ou tout simplement des liens référentiels [WCH87, ODE94]. Dans tous les cas, il est nécessaire d'avoir la possibilité de modéliser des liens sémantiques.

Il est parfois utile de considérer l'association entre un certain nombre d'objets comme un objet particulier ; ce qui permet, entre autres, de lui associer d'autres objets (attributs).

1.2 Evolution des modèles de données

Dans cette section, nous décrivons l'évolution des modèles de données dans le cadre de la gestion des informations complexes dans les bases de données, depuis le modèle relationnel, en passant par ses extensions, par les modèles sémantiques et jusqu'aux modèles de bases de données orientées objet. Nous évoquons également l'approche objet des langages de programmation.

1.2.1 Le modèle relationnel et ses extensions

Le modèle relationnel de données [COD70, DA82] est le premier modèle de base de données à avoir introduit une assise théorique forte et assuré l'indépendance des données vis à vis des traitements qui les utilisent. Il repose sur la notion de *relation.*

Une relation est définie en extension par un sous-ensemble du produit cartésien d'un certain nombre de domaines de base $D_1 \times ... \times D_n$ (de type string, real, integer, ...). Le *schéma* d'une relation est sa définition en intension, il est constitué :

- du nom de la relation,
- d'un ensemble d'attributs ; par exemple, $A_1, ..., A_n$, ayant respectivement $D_1, ..., D_n$ comme domaine de valeurs,
- d'un certain nombre de contraintes d'intégrité, portant chacune sur les attributs de la relation, et désignant une condition nécessaire que les valeurs de ces attributs, pour un n-uplet donné (élément du produit cartésien), doivent vérifier pour que celui-ci appartienne à la relation.

La notion de *forme normale* [DA82] permet de définir des catégories de relations ayant des propriétés nécessaires pour assurer la cohérence des données, par rapport à des opérateurs qui les manipulent.

L'algèbre relationnelle est formée d'un ensemble minimal d'opérateurs, permettant de manipuler un ensemble de relations. Ces opérateurs sont la projection (d'une relation sur un ou plusieurs attributs), la sélection (de certains éléments (*tuples*) d'une relation selon un critère), l'union (de deux relations compatibles), la différence (de deux relations compatibles), et le produit cartésien (de deux relations quelconques).

Certains opérateurs utiles, tels que l'intersection, la division ou la jointure, sont déduits des opérateurs minimaux.

Le calcul relationnel est un langage prédicatif qui repose sur la logique des prédicats du premier ordre. Les formules sont basées sur le fait que si R est une relation et si $\{x_1, x_2, ..., x_n\}$ est un n-uplet du domaine de R, $R(x_1, x_2, ..., x_n)$ est une formule, vraie si le n-uplet appartient à la relation, fausse sinon.

L'équivalence au niveau de l'expressivité, entre l'algèbre relationnelle et le calcul relationnel est un résultat classique, si on limite les formules du calcul à une certaine catégorie, dite des formules *saines* [DA82]. Les données peuvent alors être manipulées (en particulier

interrogées) indifféremment d'une manière algébrique ou prédicative.

Le modèle relationnel a un intérêt certain dans le domaine des bases de données de par son assise théorique solide et, surtout, parce qu'il a permis une grande indépendance des données par rapport aux traitements. Cependant, il présente des limitations importantes.

Le modèle relationnel ne permet de modéliser que des objets dont la structure est relativement plate, puisque toutes les relations sont des n-uplets de valeurs simples. Ainsi, des informations de nature plus complexe, du type de celles rencontrées en Conception Assistée par Ordinateur (CAO), ne peuvent être prises en compte de manière naturelle. De plus, le modèle étant basé sur la notion de relation (donc un ensemble de tuples), il est fondamentalement basé sur les valeurs et non sur la notion d'identité d'objet, puisque deux objets du monde réel ayant les mêmes valeurs d'attributs correspondent au même tuple.

Des études ont donc essayé de pallier ce manque, donnant ainsi naissance à plusieurs extensions du modèle relationnel. L'un des principaux objectifs de toutes ces extensions, était d'étendre l'algèbre et le calcul relationnels de manière à toujours maintenir l'équivalence d'expressivité entre eux.

Tout d'abord, des fonctions d'agrégation (somme, moyenne ou maximum des valeurs d'un attribut sur un ensemble de tuples) sont venues compléter l'algèbre et le calcul dans [KLU82].

La prise en compte du cas où un attribut a plusieurs valeurs possibles pour un seul tuple, fait l'objet de [JS82]. Les opérateurs de groupement (*Nest*) et de dispersion (*Unnest*), sont introduits pour la première fois.

Dans [FT83], une algèbre généralisant les opérateurs de groupement et de dispersion à plusieurs attributs et plusieurs niveaux d'abstraction, est introduite. Des propriétés importantes (telle que la symétrie des deux nouveaux opérateurs l'un par rapport à l'autre) ne sont pas toujours assurées. La notion de *relations imbriquées (Nested Relations)* était née.

Cette notion a depuis été explorée dans plusieurs travaux [AB86, SS86, OOM87, RKS88, COL90]. En particulier, la notion de fonctions d'agrégation introduite dans [KLU82] est étendue dans [OOM87] ; une classe de relations imbriquées appelées relations *partitionnées (Partitioned Normal Form)* est introduite dans [RKS88] ; cette nouvelle forme normale assure en particulier la propriété de symétrie des opérateurs de groupement et de dispersion.

D'un point de vue conceptuel, le fait de pouvoir représenter des données structurées (non atomiques) fut une avancée incontestable. Cependant, la structure des données reste figée, dans la mesure où tout doit être exprimé en termes de relations. Une grande partie de la sémantique des données n'est pas prise en compte par cette distribution homogène de l'information (une relation est toujours un ensemble d'objets de même type). En particulier, les notions de types génériques de données, de relations sémantiques, de nature des liens entre objets, d'identité d'objet, ... n'étaient pas prises en compte.

Les modèles sémantiques de données, que nous développons dans la section 1.2.2 ont tenté d'apporter des solutions pour la modélisation de ces notions. Ils ont notamment mis

à jour l'importance de la notion d'identité d'objet [COD79, KV84, KC86, AK89].

Cet aspect sémantique de l'évolution des modèles de données s'est accompagné par une évolution plus structurelle ; donnant lieu à l'émergence de modèles pour objets complexes [HY84, KV84, BK86, AB88, AK89]. A la différence des relations imbriquées, où un objet est un tuple appartenant à une relation, qui est un ensemble de tuples, les constructeurs tuple et ensemble sont utilisés dans un séquencement quelconque pour former un objet complexe à partir d'objets plus simples, comme mentionné dans [AH87, HUL89, SS90].

Ces modèles représentent une base structurelle pour les modèles de bases de données orientées objet, dont certains sont présentés dans 1.2.3, et dont la principale caractéristique est l'encapsulation de la structure et du comportement d'un objet dans la même coquille : sa classe. En plus de l'aspect structurel des objets, ces modèles prennent en compte les opérations qui permettent d'accéder aux structures, à l'instar des types abstraits de données. Il est à noter à ce propos, que l'un des premiers modèles à avoir souligné cette dualité dans la définition des objets est le modèle fortement typé Galileo [ACO85], où la notion de classe d'objets est obtenue par la réunion d'un type concret de données (structure) et d'un type abstrait (opérations).

1.2.2 Les modèles sémantiques

Les modèles relationnels ont une approche assez stéréotypée des informations à modéliser, puisque la finalité est d'obtenir un certain nombre de relations, pour modéliser les informations du monde réel. La complexité de celles-ci, dont certains aspects ont été discutés dans la partie 1.1, ne se prête pas toujours à cet aplatissement.

Les approches sémantiques de modélisation de données analysent les traits de complexité des objets du monde réel et en font les bases de leurs modèles. La philosophie est donc différente : au lieu d'adapter la complexité des informations aux constructeurs d'un modèle, on va définir des modèles dont les constructeurs s'inspirent de cette complexité, sans se soucier de la manière dont le niveau physique sera structuré.

1.2.2.1 Concepts fondamentaux

Les concepts décrits auparavant pour les objets complexes, sont des concepts communs aux modèles sémantiques de données. Ainsi, les notions d'agrégation, de groupement, de généralisation, sont présentes dans tous ces modèles. Cependant, certains modèles se distinguent par la prise en compte de concepts spéciaux (la généralisation par alternatives par exemple dans [LEO89]).

Nous ferons la distinction entre des modèles "généraux", dont le champ d'action n'est pas spécifique à un domaine, et des modèles qui sont destinés à des applications particulières (données temporelles, statistiques, ...).

Dans [HK87] et [PM88], une revue d'ensemble des modèles sémantiques est effectuée. Elle met l'accent sur leurs points communs et les spécificités de chacun. Nous empruntons à [HK87] le formalisme de la figure 1.1, tiré d'un modèle "générique", le modèle GSM (Generic Semantic Model).

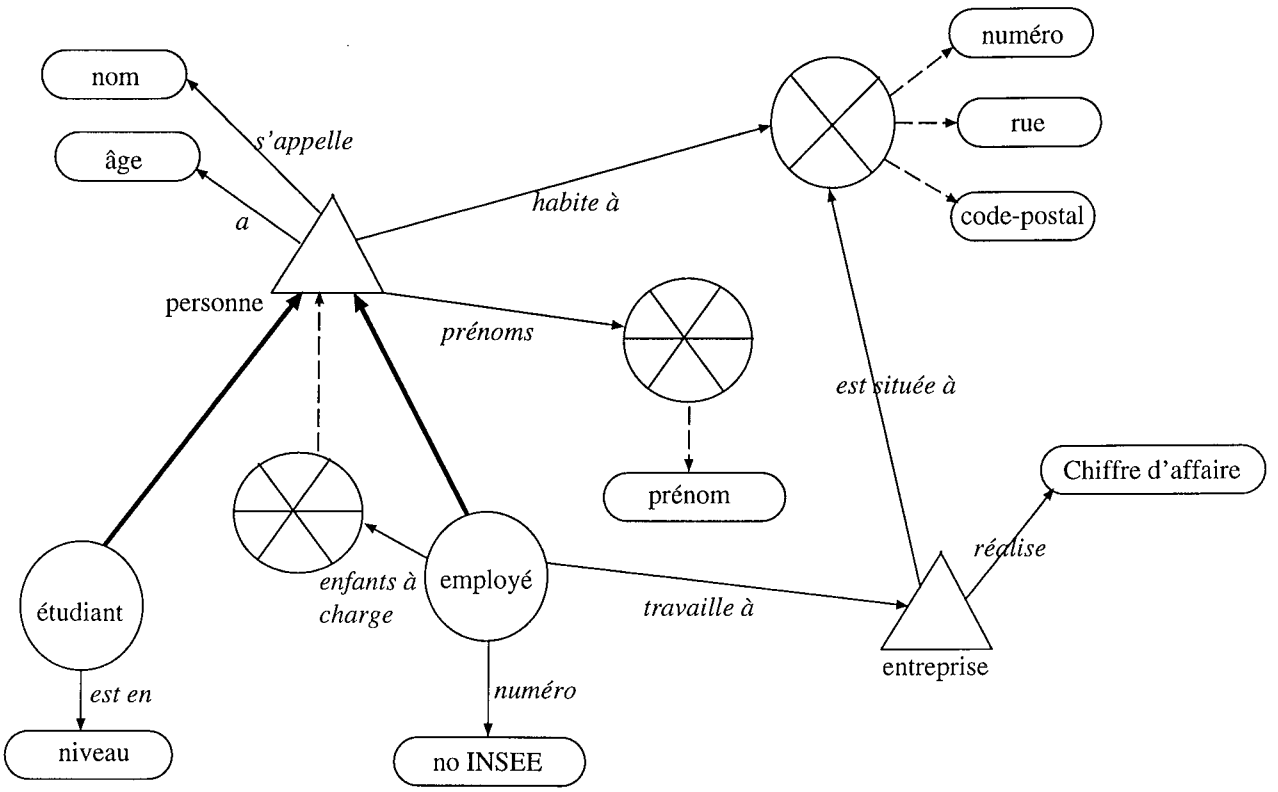


FIG. 1.1 – *Un exemple de schéma GSM pour objets complexes*

Légende :

△ types d'objets, dits types abstraits [HK87, AH87] correspondant à des entités du monde réel (personne ou entreprise, par exemple).

○ types d'objets obtenus par spécialisation de types abstraits, appelés types représentés (étudiant ou employé par exemple).

⬭ types dits imprimables ; ce sont les types de valeurs de base (entiers, réels, matrices de pixels, ...).

⊗ types obtenus par agrégation d'autres types (une adresse est l'agrégation du numéro, de la rue et du code postal).

⊛ types obtenus par groupement d'instances d'un autre type (les enfants à charge d'un employé forment un groupe de personnes).

⟶ relient un type représenté à un type abstrait, ce sont des liens de généralisation.

– – → relient un type agrégé à ses types subordonnés ou un type groupé à son type élémentaire.

⟶ relient un type (non imprimable) à un autre type, par un lien de référence (attribut).

Ici, l'amalgame est fait entre la notion de type et celle de classe. L'application modélisée par la figure 1.1, servira d'exemple pour tous les modèles présentés ci-après.

Dans les paragraphes qui suivent, nous allons détailler quelques modèles importants, soit par leurs concepts originaux, soit par leur intérêt théorique. Tous ces modèles sont des modèles généraux, en ce sens qu'ils ne sont pas destinés à un domaine d'application particulier. Parmi les modèles non cités, on trouve le modèle Entité /Association [CHE76], des modèles destinés à des applications spécifiques (les statistiques, par exemple, avec SAM* [SU83]), le modèle TAXIS [MBW80], des modèles d'investigation théorique (le modèle logique de données LDM [KV84] ou le modèle FORMAT [HY84]), ou encore le modèle fortement typé Galileo [ACO85].

1.2.2.2 Quelques modèles sémantiques importants

1.2.2.2.1 Le modèle relationnel RM/T Le modèle RM/T (Relational Model of Tasmania)[1] [COD79] est une extension du modèle relationnel, destinée à "saisir plus de sémantique". De ce point de vue, il s'intégre parfaitement dans la famille des modèles sémantiques.

La notion importante de "surrogate" introduite dans ce modèle permet d'une part de tenir compte de la notion d'identité d'objet, et d'autre part de définir la notion de classe. En effet, contrairement à l'uniformité du modèle relationnel initial, plusieurs types de relations sont distingués, dont: les E-Relations et les P-Relations. Les premières, où la lettre E signifie "Entité", sont des relations monocolonnes, contenant des identificateurs des entités d'un certain type. A chaque type d'entité correspond donc une E-Relation. A chaque E-Relation (à chaque type d'entité), correspond un certain nombre de P-Relations, où la lettre P signifie "propriété". Chaque P-Relation relie une entité (désignée par son surrogate) à certaines de ses propriétés, qui prennent leurs valeurs dans un domaine de base ou dans le domaine des surrogates). C'est une manière de modéliser les attributs.

L'une des particularités du modèle RM/T est que c'est l'un des premiers modèles à avoir distingué, dans la notion de généralisation, le cas particulier de la généralisation par alternative. Par exemple, il est possible de distinguer la généralisation de "secrétaire" en "employé", de celle de "client" en "personne physique" et "personne morale". Dans le premier cas, une secrétaire est toujours un employé, alors que dans le second, un client est soit une personne physique soit une personne morale (entreprise cliente).

Le lien de généralisation est donc représenté à travers deux types particuliers de relations: les UGI-Relations et les AGI-Relations. Les deux premières colonnes de ces relations, respectivement intitulées SUB et SUP, prennent leurs valeurs dans le domaine des noms de relations RN (Relation Name). Elles désignent les E-Relations qui entrent en jeu dans le lien de généralisation. La colonne SUP (resp. SUB) indique la E-Relation générique (resp. spécifique). Les UGI-Relations traduisent des liens de généralisation *non contraints*, comme celui de "secrétaire" en "employé". Les AGI-Relations, quant à elles, traduisent des liens de généralisation *par alternative*, comme ceux de "client" en "personne

1. Sa première présentation a été faite lors d'une conférence qui a eu lieu en Tasmanie !

physique" et "personne morale".

Remarquons que dans la terminologie de RM/T [COD79], la notion de généralisation traduit l'inclusion (ensembliste) entre les E-Relations. La différenciation n'est donc pas faite entre généralisation et spécialisation.

Une autre particularité intéressante est qu'il est possible de définir des types agrégés ou des valeurs agrégées simplement. En d'autres termes, il est possible de définir des types comme étant des agrégations et d'attacher un surrogate à leurs entités, les autorisant ainsi à avoir des attributs (propriétés), comme il est possible de définir uniquement une valeur agrégée (dite "nonentity aggregation").

La figure 1.2 représente des relations traduisant l'application générique du schéma GSM de la figure 1.1.

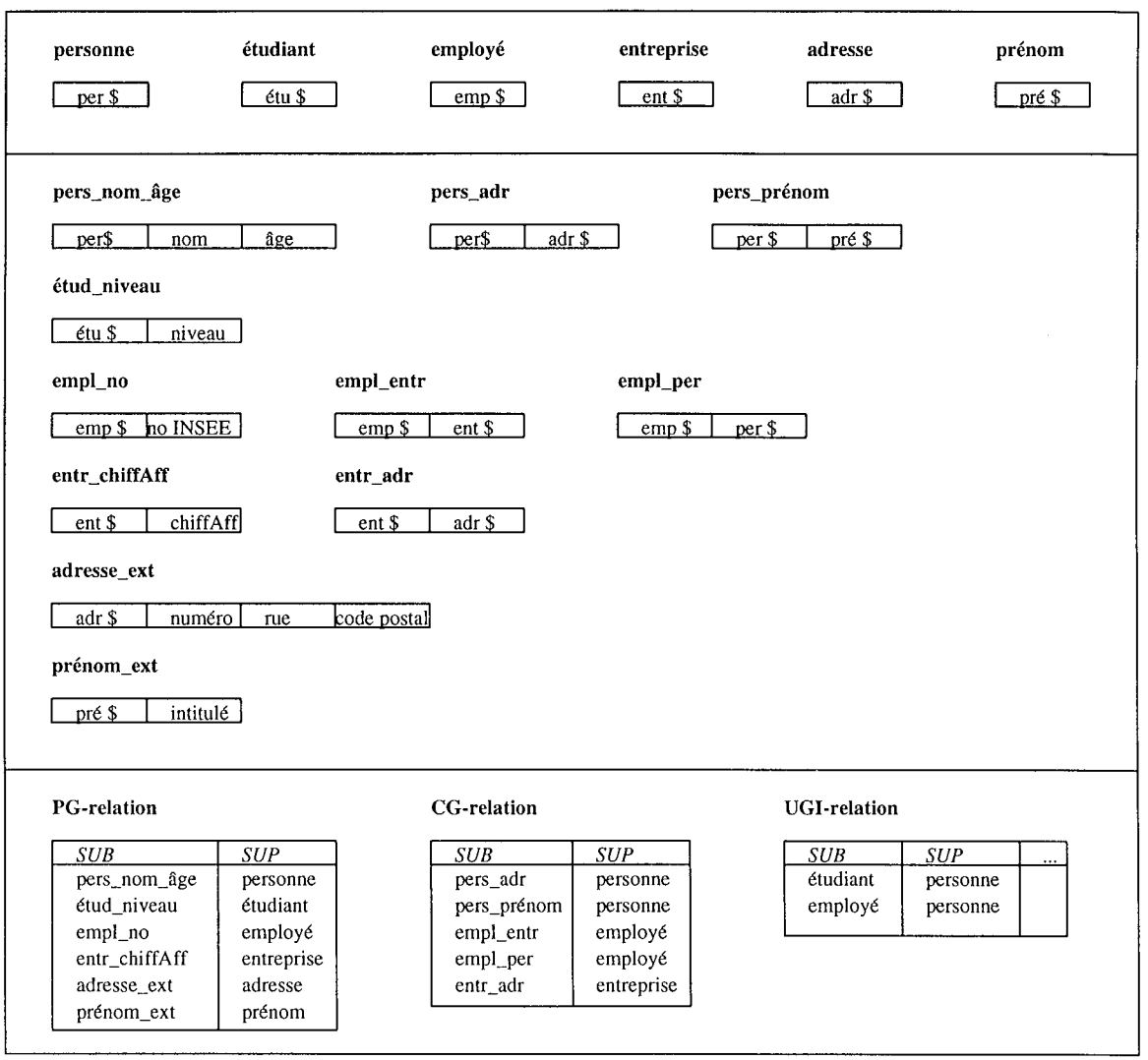


FIG. 1.2 – *Un schéma de relations RM/T.*

La première partie de la figure représente les E-Relations de l'application. La seconde

représente les P-Relations. La dernière représente des relations particulières du modèle RM/T. La PG-Relation relie une E-Relation (SUP) à ses propriétés (attributs imprimables). La CG-Relation relie une E-Relation (SUP) à ses "caractéristiques" (attributs entités) avec éventuellement des attributs imprimables. Ceci permet de modéliser les associations entre entités. Enfin, la UGI-Relation représente le graphe de généralisation (non contrainte).

L'intérêt du modèle RM/T est qu'il repose sur le modèle relationnel, en l'étendant avec des considérations sémantiques. Il peut être considéré comme la définition d'un modèle sémantique à travers les composants du modèle relationnel. Le fait de reposer sur le modèle relationnel représente à la fois un avantage et un handicap. C'est avantageux en raison de la grande notoriété du modèle relationnel dans le domaine des bases de données ; et c'est contraignant à cause de la nature "unidimensionnelle" du modèle relationnel, ramenant tous les concepts à celui de relation. D'autre part, les informations concernant une entité, sont dispersées sur plusieurs relations, ce qui alourdit les traitements.

1.2.2.2.2 Le modèle fonctionnel FDM FDM (Functional Data Model) [SHI81] est un modèle à la fois simple et puissant. Sa simplicité vient de ses primitives, qui se résument à des fonctions (et des entités considérées comme des fonctions à ensemble de départ vide). Sa puissance vient du fait qu'il permet de tenir compte de presque tous les concepts des modèles sémantiques de données.

Les domaines des fonctions sont soit des domaines de base (integer, string, ...), soit le domaine "entity" (pour les fonctions représentant des entités), soit l'ensemble des instances d'une entité particulière définie auparavant. Ainsi, l'application citée jusqu'à présent en exemple peut être modélisée par le graphe de la figure 1.3.

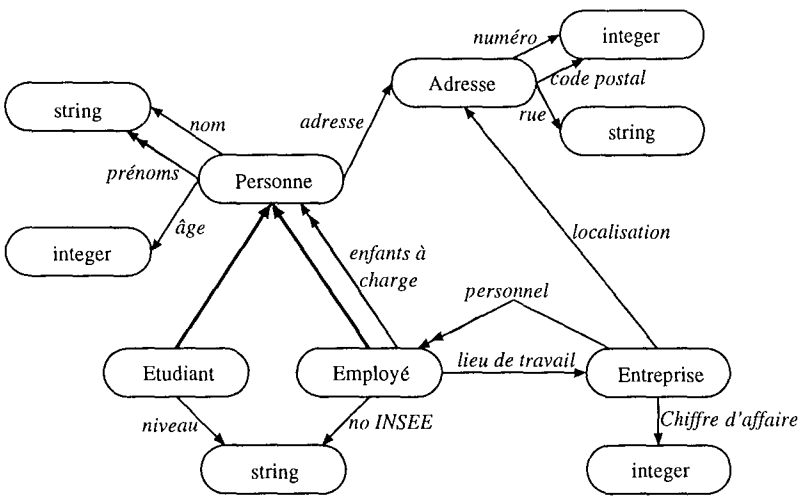


FIG. 1.3 – *Un schéma fonctionnel selon le modèle FDM.*

Les entités, ainsi que les domaines de base, sont représentés par des rectangles à bords arrondis. Les flèches simples indiquent des fonctions (références) monovaluées. Les flèches doubles indiquent des fonctions multivaluées. les flèches en gras représentent les liens de

généralisation / spécialisation. Les domaines de base sont répétés pour la lisibilité du schéma.

Il est possible de définir une fonction comme l'inverse d'une autre. Par exemple, la fonction "personnel" de l'entité "Entreprise" associe à toute instance E de cette entité, les instances de l'entité "Employé" dont la valeur de la fonction "lieu de travail" vaut E. D'autres exemples de fonctions calculées à partir d'autres sont possibles.

Le langage de description de données est une partie du langage de gestion de base de données DAPLEX [SHI81]. Voici quelques exemples illustratifs et représentatifs de la définition du schéma de la figure 1.3 :

```
declare Personne() => entity
declare Etudiant() => Personne
declare nom(Personne) => string
declare prénoms(Personne) =>=> string
declare lieudetravail(Employé) => Entreprise
declare personnel(Entreprise) =>=> inverse of lieudetravail(Employé)
```

La simplicité d'utilisation du modèle fonctionnel de données et du langage DAPLEX, ainsi que son intégration de la majorité des concepts des modèles sémantiques de données sont un grand atout. La limitation de ses concepts aux entités et fonctions, et la possibilité de définir des fonctions calculées, permettent l'expression de plusieurs constructions. Ainsi, la notion de vue peut être facilement explicitée. Par exemple :
`declare Nometudiant() => nom(Etudiant())` permet de définir une entité dont les instances sont les noms des étudiants présents dans la base.

1.2.2.2.3 Le modèle sémantique SDM Basé sur la sémantique des applications à modéliser, le modèle SDM (Semantic Data Model) [HM81] se fixe pour but de raccourcir l'écart entre la perception de l'utilisateur du monde réel et la manière dont elle est représentée, en rompant avec la tradition des modélisations par enregistrements (record-based) et en offrant la possibilité d'avoir une "vue relativiste du monde réel", par différentes vues des mêmes informations.

Une application est représentée par un ensemble d'entités (correspondant aux objets réels), groupées en classes. Chaque classe a un ensemble d'instances (entités associées), un ensemble d'attributs d'instances (attributs répercutés dans toutes les instances, comme le nom d'une personne), un ensemble d'attributs de classe (attributs concernant la classe en tant qu'ensemble d'instances, comme l'âge moyen de la classe "étudiant"), et un type (de base ou non).

Toute classe, autre que les classes de base, dépend d'une autre classe par l'intermédiaire d'une "interconnexion". Deux types d'interconnexions permettent soit de relier une sous-classe à sa super-classe, soit de relier deux classes, telles que les instances de la première soient des ensembles d'instances de la seconde. L'agrégation n'est pas définie explicitement dans SDM.

Un avantage de SDM, en ce qui concerne la généralisation / spécialisation, réside dans le fait que 4 possibilités sont offertes lors de la définition d'une sous-classe.

- Une sous-classe SC d'une classe C peut être définie comme étant l'ensemble des instances de C vérifiant une certaine condition sur certains attributs. Par exemple, la classe "navires marchands" est une sous-classe de la classe "navires", dont les instances sont des instances de "navires" avec l'attribut "type" valué à "marchand".
- Une sous-classe SC peut être définie par rapport à une classe C, avec un mode d'insertion / suppression d'instances, laissé au soin de l'utilisateur. Par exemple, la classe "navires bannis" est une sous-classe de "navires", dont les instances sont des instances de "navires" indiquées au fur et à mesure par l'utilisateur.
- Une sous-classe SC de C peut être définie par des opérations ensemblistes sur d'autres sous-classes de C (intersection, réunion, différence). Par exemple, la sous-classe des "navires marchands bannis" est l'intersection des deux sous-classes précédentes de la classe "navires".
- Enfin, une sous-classe SC de C peut être définie comme étant l'ensemble des instances de C, valeurs d'un attribut A d'une autre classe D, dont le domaine est la classe C. Par exemple, la classe "navires à problèmes" est une sous-classe de "navires", dont les instances sont des valeurs de l'attribut "navire" de la classe "accidents".

Il est possible de définir dans SDM la notion de méta-classe, qui est une classe dont les instances sont des classes.

Les attributs sont aussi définis de plusieurs manières (attributs inverses, attributs calculés à partir d'autres, ...).

Il n'y a pas de formalisme graphique particulier correspondant à SDM ; cependant, voici la description en syntaxe SDM, des classes "Etudiant" et "Entreprise" de notre exemple d'application.

```
ETUDIANT
   description: sous-classe de PERSONNE
   interclass connection: subclass of PERSONNE where specified
   member attributes:
      Niveau
         value class: STRING

ENTREPRISE
   description: petites et moyennes entreprises
   member attributes:
      Adresse
         value class: ADRESSE
      Chiffre_d'affaire
         value class: STRING
      Personnel
```

```
value class : EMPLOYE
inverse : Travaille_à
multivalued
```

Ainsi, le modèle SDM offre une large panoplie d'outils sémantiques de modélisation, à travers la notion centrale de *classe*.

1.2.2.2.4 Le modèle IFO Parmi les modèles de données existants, le modèle IFO [AH87] offre un intérêt particulier. En effet, ce modèle a pour origine l'étude des structures inhérentes aux modèles sémantiques. C'est le plus récent, et celui qui se rapproche le plus du modèle générique GSM, défini dans 1.2.2.1. D'ailleurs le formalisme graphique d'IFO diffère très peu de celui de GSM (cf figure 1.4).

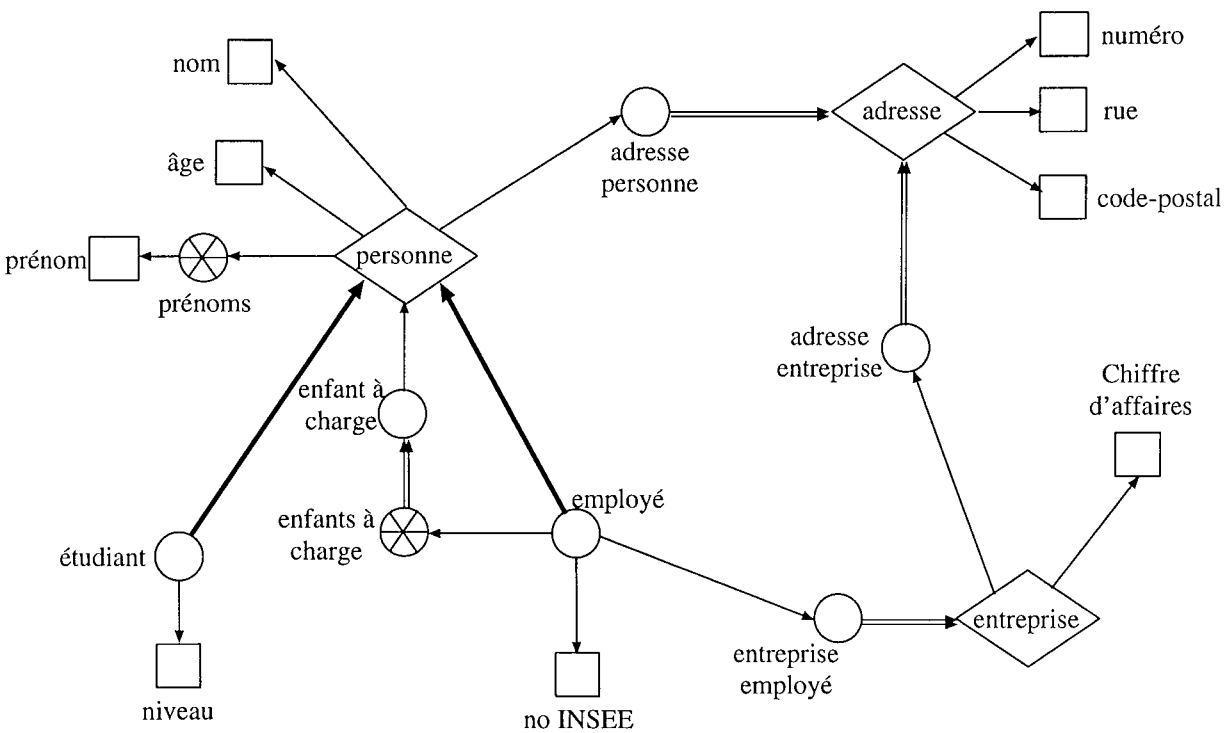


FIG. 1.4 – *Un schéma IFO.*

Le modèle IFO a été construit afin de pouvoir servir de base pour des recherches rigoureuses et mathématiques, à propos de problèmes liés aux bases de données sémantiques (reposant sur un modèle sémantique), du point de vue structurel de la base, et non de la manipulation de données ou encore de leur intégrité. Il s'intéresse en particulier aux propagations des mises à jour dans de telles bases de données.

Le modèle IFO considère trois types atomiques de données : le type imprimable (□), le type abstrait (◇) et le type libre (○). Le premier concerne les types de base, imprimables (string, integer, ...). Le second sert à référencer des objets sans structure interne, dans la base. Le dernier permet de définir des types par l'intermédiaire des liens IS-A de

généralisation / spécialisation.

IFO offre les deux constructeurs de type : l'agrégation ($\otimes$) et le groupement ($\circledast$), pouvant être récursivement combinés à volonté. Il introduit en outre la notion de fragment permettant de modéliser la notion d'association d'objets par l'intermédiaire de fonctions (flèches simples). Par exemple, dans la figure 1.4, le fragment "personne" est l'arbre ordonné, issu du type abstrait personne, et suivant le sens des flèches simples, vers les associations induites par ce type (nom, âge, prénoms et adresse personne). Il est possible d'associer des propriétés (attributs) à une association (par exemple une date au type "adresse personne").

Si on observe la figure 1.4, on remarquera que des types supplémentaires (principalement des types libres) ont été rajoutés au schéma GSM de la figure 1.1. Ainsi, le type libre "entreprise employé" est un sous-type du type abstrait "entreprise", contenant les entreprises référencées par un employé. Cela rejoint la quatrième et dernière possibilité de définition de sous-classe dans le modèle SDM (valeurs d'un attribut d'une autre classe).

Une originalité d'IFO est que la différenciation entre les deux concepts de généralisation et de spécialisation est visible sur le schéma graphique de l'application (cf figure 1.4). Des flèches doubles désignent une spécialisation du type qui se situe à la pointe de la flèche vers le type origine ("adresse" en "adresse entreprise", par exemple). Des flèches grasses désignent une généralisation du type qui se situe à l'origine vers le type destination ("employé" et "étudiant" en "personne", par exemple).

En outre, les particularités des graphes IFO ont conduit à imposer des règles de construction de graphes corrects [AH87].

Le modèle IFO nous semble être, sinon le plus complet, du moins celui qui regroupe d'une manière synthétique et génératrice, les concepts fondamentaux des modèles sémantiques. Son formalisme graphique riche permet d'avoir des graphes très "parlants".

Outre ses utilisations diverses (en conception et description de modèles de données, en particulier [HEU89]), le modèle IFO a donné lieu à un certain nombre d'études, parmi lesquelles nous citerons [PON93], qui étudie en particulier l'implantation de l'approche IFO sur une base de données orientée objet, à travers un modèle de transition appelé IFO_2.

1.2.2.3 Discussion

Les modèles sémantiques ont émergé compte tenu de l'insuffisance des modèles antérieurs (hiérarchique, réseau et relationnel) pour traiter des données de plus en plus complexes. Ils ont apporté un certain nombre de solutions à des problèmes de modélisation, notamment en introduisant la notion d'identité d'objet, de classes d'objets, de types, ...

Les modèles sémantiques de données ont trouvé, en l'approche objet, le prolongement logique dans la mesure où celle-ci prend en compte un certain nombre de leurs concepts fondamentaux, dont notamment la notion d'identité d'objet (rompant ainsi de manière définitive avec l'approche basée sur les valeurs), la notion de classes d'objets, de spé-

cialisation (à travers l'héritage). Les bases de données orientées objet ont donc ouvert une ère nouvelle dans le domaine des banques de données, en y introduisant le concept d'*encapsulation* des données : les données d'un objet ne sont accessibles qu'à travers une interface de procédures, qui lui est propre, et qui en fait partie intégrante.

1.2.3 Les modèles de bases de données orientées objet

Un *objet* peut être vu comme étant la réunion de trois éléments :

- un *identificateur* unique et inchangé durant tout le cycle de vie,
- une *structure* représentant sa partie "interne", strictement confidentielle vis à vis des autres objets,
- une *interface* représentant sa partie "extérieure" ; seul moyen pour l'objet de communiquer avec l'extérieur.

La seule manière de communiquer avec un objet étant son interface (constituée en fait d'un certain nombre de procédures appelées "méthodes", effectuant des traitements spécifiques en réaction à des requêtes envoyées par d'autres objets sous forme de messages), un objet a un comportement défini par ses réactions aux envois de messages.

Les objets d'une base de données orientée objet sont regroupés dans des classes d'objets. Les objets d'une même classe ont la même structure et surtout la même interface, d'où un comportement similaire.

Cette nouvelle manière de voir les choses (ayant ses racines dans les recherches en langages de programmation et en représentation de connaissances pour l'intelligence artificielle) a donné lieu à une nouvelle structuration des bases de données et donc une nouvelle vague d'approches de modélisation. La dispersion modulaire de l'information en unités de sens (objets, classes) a, par exemple, donné naissance à la "conception pilotée par les responsabilités" [WBJ90], où un objet est vu comme une entité qui assure un certain nombre de responsabilités au sein de l'application. La manière de concevoir les bases de données a radicalement évolué, en parallèle avec la manière de les organiser.

Si l'encapsulation influence considérablement les méthodes de conception, la structure des objets, quant à elle, influence l'organisation interne de la base de données.

Chaque objet, dans une Base de Données Orientée Objet (BDOO), est instance d'une classe et possède un certain nombre de variables d'instance, qui jouent le rôle des attributs et qui sont eux même des objets d'une certaine classe.

La notion de généralisation / spécialisation de classes en sous-classes est présente et est contrôlée par le mécanisme d'héritage. Une sous-classe hérite de sa super-classe sa structure et son interface. Elle a la possibilité de rajouter des variables d'instance et des méthodes, mais aussi de redéfinir les méthodes héritées en fonction de ses spécificités. Ceci permet en particulier de différer la définition des actions d'une méthode, d'une classe vers ses sous-classes, dans le cas où les différentes sous-classes implémentent différemment cette méthode.

La notion d'héritage multiple permet de désigner une classe comme étant une sous-classe de plusieurs autres classes et non pas d'une seule. Ceci pose le problème des conflits de noms, provenant de l'héritage de deux méthodes ayant le même nom, de deux super-classes différentes, par exemple.

L'approche orientée objet est à la fois simple et riche puisqu'elle permet par exemple de définir des associations entre objets par l'intermédiaire des variables d'instances, une confidentialité des données à travers l'encapsulation, ...

Pour gérer des BDOO, deux approches ont été proposées [KIM90] : étendre un langage orienté objet avec des paradigmes de bases de données (tels que la persistence, la gestion d'accès concurrents, ...) ou définir des vues objet dans un Système de Gestion de Bases de Données (SGBD) hôte (relationnel, en général).

Cette dernière approche, comme pour les bases de données classiques, pose le problème de l'*impedance mismatch* [CM84]. En effet, le fait d'avoir recours à deux langages différents pour gérer la base de données introduit des *incohérences conceptuelles* dues au fait que chaque langage a sa propre nature (déclaratif, procédural, ...), ainsi que des *incohérences structurelles* dues au fait que chaque langage manipule des structures de données différentes. Un autre problème réside dans le fait que les programmeurs doivent se familiariser avec deux langages, différents sous plusieurs points de vue.

Quant à la première approche, la programmation orientée objet persistente, les principaux problèmes qu'elle engendre concernent la gestion en mémoire centrale des objets.

Une troisième approche consiste à définir des Systèmes de Gestion de Bases de Données Orientées Objet (SGBDOO) qui intégrent à la fois les fonctionnalités d'un SGBD (gestion des concurrences d'accès, sécurité des données, persistence, etc) et celles des langages de programmation à base d'objets (héritage, encapsulation, etc) [DLR91].

Chaque SGBDOO a sa propre technique pour la représentation des identifiants d'objets (identificateur d'instances et/ou identificateur de classe), des hiérarchies d'héritage, des références et de la gestion de l'évolution du schéma de la base. Le langage de requêtes présente un autre critère de comparaison des SGBDOO, ainsi que la manière de gérer l'indexation des objets, des classes, des hiérarchies, ou encore la gestion des autorisations utilisateurs, intéressante du fait de la nature cyclique des références inter-classes (droit de consulter une classe, une hiérarchie, de créer une sous-classe d'une classe, une instance, ...). Enfin, chaque SGBDOO a sa propre technique de résolution des concurrences d'accès.

En outre, la technique de typage des objets est un point important pour la protection des données. Un système de types performant permet de détecter un maximum d'erreurs dès la compilation des programmes, évitant de nombreux problèmes et incohérences à l'exécution [DLR91]. Un typage statique assigne un type à tout individu (objets, classes, résultats de méthodes, ...) permettant une compilation puissante, mais coûteuse. Un typage dynamique, quant à lui, ne vérifie l'existence d'une méthode pour un objet que lors de l'exécution effective, renforçant ainsi la flexibilité du système.

Dans la suite, nous allons considérer quelques SGBDOO parmi les plus courants et en détailler certaines caractéristiques.

1.2.3.1 Gemstone

Le SGBDOO Gemstone [CM84] est une extension directe du langage Smalltalk [GR83], avec des paradigmes de programmation persistente. Il est considéré comme étant un pionnier dans le domaine des SGBDOO, et l'un des premiers prototypes ayant donné lieu à un produit commercial.

Le langage de description des données (LDD) ainsi que le langage de manipulation de données (LMD) sont des dérivés directs de Smalltalk, avec pratiquement la même syntaxe. Ainsi la création d'une nouvelle classe "Personne" se fait par l'envoi d'un message de création de sous-classe, à la classe prédéfinie "Object".

Object subclass: #Personne
instVars: 'nom prénoms âge adresse'
classVars: ''
immutable: false
constraints: #[#[nom String]
#[prénoms Chaînes]
#[âge Float]
#[adresse Adresse]]

"String" et "Float" sont des classes prédéfinies, alors que "Chaînes" et "Adresse" sont des classes définies ailleurs, dans la base.

Le langage de requêtes STDM [CM84] est basé sur des ensembles étiquetés de valeurs hétérogènes, pouvant être elles-même des ensembles ou de simples valeurs. Il s'agit d'un langage général, à structure déclarative, indépendant de Smalltalk.

Exemple :
Soit l'ensemble E1 suivant :

```
E1: {Personne: {P1: {nom: 'toto', prénoms: {p1}, âge: 24, adresse: a1},
                P2: {nom: 'titi', prénoms: {p2}, âge: 42, adresse: a2}},
    Chaînes: {p1: {'jean', 'jules'},
              p2: {'jacques', 'joe'}},
    Adresse: {a1: {no: 12, rue: 'des roses', cp: 54000},
              a2: {no: 1, rue: 'des lilas', cp: 54506}}}
```

Cet ensemble représente deux personnes P1 et P2, deux groupes de prénoms p1 et p2 et deux adresses a1 et a2. Les étiquettes correspondent soit à des noms d'attributs (rue) ou à des noms d'objets (P1).

L'accès aux différentes informations contenues dans un ensemble STDM se fait par l'intermédiaire des étiquettes, en en formant un chemin ; par exemple, 'E1!Personne!P2!nom' a pour valeur 'titi'. STDM emploie alors, pour les requêtes, une syntaxe déclarative proche de celle de SQL.

La nature hétéroclite des ensembles STDM est un atout pour la prise en compte de données structurellement hétérogènes comme les instances de différentes sous-classes d'une même classe. Elle permet en outre de saisir différentes structures de données (tableaux, ...).

Le langage OPAL de programmation du SGBDOO Gemstone est le résultat de l'intégration des deux langages Smalltalk et STDM, par l'intermédiaire de l'analogie entre objets et ensembles et entre variables d'instances et couples étiquette-valeurs.

1.2.3.2 Orion

Le système Orion [BCG+87, KBG89, KIM89] est aussi l'un des tous premiers SGBDOO ; il s'est cependant fixé des objectifs plus précis que Gemstone, en termes de domaines d'utilisation, dans la mesure où il se présente comme étant plus particulièrement destiné à des applications de CAO, d'IA, de bureautique et à des environnements multimédias.

Son modèle de données est donc un peu différent de celui de Gemstone. Les concepts de classes, d'instances, d'envoi de message, d'héritage sont bien sûr présents. De plus, certaines particularités distinguent Orion du reste des SGBDOO.

En premier lieu, les références entre classes d'objets à travers les variables d'instances sont répertoriées en plusieurs types, selon la nature du lien entre les deux classes d'objets [KBG89]. Une référence simple traduit simplement le fait qu'un objet fait intervenir un autre dans sa définition. Par exemple, pour une classe "voiture", la référence vers l'instance de la classe "personne" qui en est la propriétaire est une référence simple. Une référence de composition traduit en plus le lien "partie-de" entre les deux objets. Par exemple, le lien entre un livre et ses chapitres est un lien de composition.

De plus, le fait que l'existence de l'objet référencé soit liée à celle de l'objet référençant (soit parce qu'il s'agit d'un lien physique, soit simplement parce que l'objet référencé fait partie intégrante de l'objet référençant) différencie une catégorie particulière de liens de composition, dits de *dépendance*, des autres.

La distinction de ces différents types de liens intervient dans diverses considérations : la gestion de la dynamique, les versions d'objets, l'indexation, les autorisations, ...

La gestion de la dynamique des applications, du point de vue du schéma de la base de données, est un point auquel Orion a accordé une attention particulière. Dans [BCG+87], différents cas d'évolution de schéma ont été répertoriés en taxinomie, complétée plus tard dans [KBG89] par les cas particuliers de nouveaux types de liens de composition introduits. Ceci a dégagé un certain nombre d'invariants à respecter lors de ces évolutions.

Le respect de ces invariants a donné lieu à des règles précises d'évolution de schéma. Par exemple : "A toute procédure (resp. variable d'instance) héritée de plusieurs superclasses avec le même nom, on affecte le code (resp. la valeur) de celle correspondant à la

classe ayant la plus faible étiquette sur le treillis d'héritage".

En ce qui concerne le langage de requêtes [KIM89], il s'agit d'une extension de Lisp. Une de ses particularités réside dans le fait qu'une requête s'applique à une seule classe, ou alors à la hiérarchie d'héritage de classes qui en est issue. Une requête sur une classe est définie par un prédicat (simple ou avec opérateurs) de sélection de certaines de ses instances. Le résultat d'une requête est donc toujours un ensemble d'instances de la classe cible. Cependant, lorsque la requête s'applique à une hiérarchie de classes, le résultat peut être un ensemble d'instances de différentes sous-classes de la classe cible. Dans tous les cas, le résultat d'une requête donne lieu à la *création de nouvelles classes d'objets* avec de nouveaux identificateurs classés, le cas échéant, selon les sous-classes de la classe cible impliquées dans le prédicat.

Orion est un système intéressant de par ses particularités et son orientation vers des applications particulières (le multimédia, par exemple).

1.2.3.3 O_2

Le système O_2 [Da90, DLR91], contrairement à Orion, n'a pas d'applications cibles particulières. Il se veut, au contraire, un système généraliste regroupant un certain nombre de fonctionnalités déjà introduites par d'autres précurseurs. Son approche est donc qualifiée d'*horizontale* [DLR91].

le modèle de données d'O_2 offre une particularité intéressante par rapport aux autres SGBDOO : il n'a pas une approche "tout-objet" des informations. Il différencie donc la notion d'objet de la notion de valeur.

Une valeur peut être simple ou structurée par l'un des constructeurs *tuple, set* ou *list*. Les valeurs simples peuvent être des identifiants d'objets, ou simplement des entiers, des caractères, ... Un objet est une paire *(identifiant, valeur)*.

La différenciation entre objets et valeurs a conduit naturellement à différencier entre classes (d'objets) et types (de valeurs). Un type est un ensemble de valeurs ayant une structure commune. Comme les valeurs, il peut être simple ou structuré par les constructeurs précités, sur d'autres types. Une classe est un triplet *(nom, type, méthodes)* où *type* désigne le type obligé pour la partie valeur des instances et où *méthodes* désigne un ensemble de méthodes applicables à toutes les instances de la classe.

Par exemple, la structure d'une classe "Personne" est définie de la manière suivante (sans les méthodes) :

```
class Personne
type tuple (nom : string,
            prénoms : list(string),
            âge : float,
            adresse : Adresse)
end
```

La persistence des objets dans O_2 se fait à travers le mécanisme de nommage des objets et des valeurs par l'utilisateur, et des références entre objets. Ainsi, tout objet nommé par l'utilisateur est persistent et tout objet référencé par un objet persistent l'est aussi. Les noms des objets sont des variables globales de l'application ; ils permettent en particulier de simuler la notion d'identité d'objet vis-à-vis de l'utilisateur puisque les identificateurs d'objets sont en principe inaccessibles aux utilisateurs.

En ce qui concerne l'héritage, il est autorisé à être multiple et repose sur la notion de sous-typage entre les types associés aux classes en question. Un mécanisme de vérification de cohérence de types et d'inférence de sous-types est activé lors de toute notification de message de définition de sous-classe.

Les langages de définition et de manipulation de données sont définis en fonction des syntaxes de langages hôtes, principalement celles de C (O_2C) et de Basic ($BasicO_2$), en les enrichissant d'une couche orientée objet.

Le langage de requêtes (O_2SQL), quant à lui, est fonctionnel et a une syntaxe semblable à celle de SQL. La particularité du modèle de données d'O_2 se retrouve au niveau du résultat des requêtes. En effet, les résultats de requêtes sont soit des valeurs existantes, soit des valeurs inférées (jointures, projections, ...), soit des objets existant déjà dans la base.

Le système O_2 est intéressant de par son système de types évolué et son orientation généraliste. Cependant, ce typage structurel fort implique bien sûr une certaine rigidité des applications.

1.2.3.4 Discussion

Si on met de côté les spécificités de chaque SGBDOO, leurs aspects fondamentaux communs fournissent des mécanismes qui permettent de gérer des informations complexes. Ainsi, les notions de classe, d'instance, de références, d'héritage, permettent d'avoir une base solide pour la prise en compte d'informations complexes.

Si on excepte le cas des systèmes destinés initialement à certains domaines d'application, les SGBDOO fournissent en général un environnement de modélisation plus qu'un modèle de données. En effet, leurs modèles à la fois simples et riches permettent de définir de nouvelles primitives de modélisation, par l'intermédiaire notamment de la notion d'héritage et de définition par affinements successifs.

Les systèmes actuellement commercialisés prouvent que l'approche est riche et se prête à des utilisations d'autant plus diverses que sa flexibilité n'est plus à prouver. Une théorie unique standardisée et bien spécifiée (comme pour le relationnel) est nécessaire. Un premier pas dans ce sens a été franchi par l'ODMG (*Object Database Management Group*) en 1993 [CAT94].

1.2.4 Y a-t-il des objets sans classes?

Les modèles de BDOO trouvent leur origine dans les réflexions soulevées par les modèles sémantiques à propos de l'insuffisance du modèle relationnel, ainsi que dans l'avancée

des langages de programmation par objets.

Au début des années 80, deux grandes tendances se confrontaient dans ce dernier domaine. L'approche basée sur les classes à la Smalltalk [GR83] (qui a été adoptée dans les modèles pour bases de données) et l'approche basée sur les *prototypes* (résumée par exemple dans [DMC92]).

Cette dernière approche constitue une alternative intéressante aux classes. Un prototype est un objet particulier qui possède lui-même tout ce qui le concerne (structure, comportement) et peut être dupliqué pour créer de nouveaux objets. Chaque objet évolue alors indépendamment des autres.

Dans [BOR86], Borning avance qu'en contre-partie de l'avantage d'accès groupé aux objets, les langages de classes présentent plusieurs inconvénients liés à la présence de méta-classes (et la difficulté que cela représente pour comprendre le langage), la détention des primitives de comportement par les classes et non par les objets, ... Il conclut que malgré certains inconvénients (représentation d'objets basiques comme les entiers, partage de l'information), les prototypes sont d'une utilisation plus intuitive que les classes.

1.2.4.1 Les langages à prototypes

Dans [LIE86], Lieberman remarque que la conceptualisation mentale repose sur la technique des individus typiques. Lorsqu'on veut représenter le concept d'éléphant, par exemple, il suffit de représenter un éléphant particulier (appelons-le Clyde). Pour représenter un autre éléphant, il suffit de représenter ce qui le différencie de Clyde. Si on désire créer une représentation d'un éléphant quelconque, il suffit donc de recopier la représentation de Clyde et ensuite modifier ce qui doit l'être pour obtenir la représentation du nouvel éléphant.

Dans les langages à prototypes, l'information est représentée par des objets. Chaque objet possède un certain nombre de *slots* qui modélisent sa structure (slots de données) et son comportement (slots d'opérations). Les objets sont appelés *prototypes* car chacun d'eux peut être utilisé pour en créer un nouveau grâce au mécanisme de clonage. Chaque objet contient en lui-même tout ce qui le caractérise (structure et comportement), à la différence des langages de classes où le comportement des objets est encapsulé au sein de la classe.

L'un des principaux paradigmes de la programmation par objets en général est le partage de l'information entre objets. Dans les langages de classes, le partage de l'information est effectué selon deux axes orthogonaux.

- Le partage de l'information entre les objets d'une même classe se fait à travers la notion même de classe. Tous les objets d'une classe partagent la même structure et le même comportement.
- Le partage de l'information entre différentes classes se fait au travers du mécanisme d'héritage. Une classe spécifique hérite (donc partage) toutes les caractéristiques de sa classe générique.

Dans les langages à prototypes, le partage de l'information se fait par l'intermédiaire du mécanisme de *délégation* [LIE86]. Tout objet a la possibilité de déléguer la réponse

à un message (valeur d'un slot de données, exécution d'un slot d'opération) à un autre prototype. Ces deux prototypes partagent la réponse au message en question.

Lieberman montre dans [LIE86] que ce mécanisme est plus puissant que le mécanisme d'héritage car il peut simuler celui-ci, sans que la réciproque soit vraie ! Cependant, en "co-signant" le *traité d'Orlando* quelques années plus tard [SLU89], il admet que dans certains cas (lorsque l'application à modéliser est assez mûre - les signataires utilisent l'expression "large, relatively routine software production" -), classes et héritage peuvent être un moyen de modélisation plus approprié.

Ce traité stipule néanmoins que, dans le cas général, un langage orienté objet doit implémenter les notions d'*empathie* (partage de l'information) et de *template* (moule) pour la création d'objets. Des mécanismes de partage dynamique, explicite et individuel de l'information (délégation), sont préférables à un partage statique, implicite et groupé (héritage). Il se prononce ainsi, sans équivoque, en faveur des prototypes et de la délégation.

L'un des langages à prototypes les plus célèbres est le langage Self [US87, US91]. Le mécanisme de délégation y est basé sur un lien appelé *parent*. Un objet peut avoir plusieurs parents, auxquels il délégue la réponse à des messages qu'il ne comprend pas. Les parents sont donc considérés comme des parties de l'objet [CUCH91]. La multiplicité des parents pose, dans le cas de la délégation, des problèmes similaires à l'héritage multiple. Une technique de désambiguïsation des messages, adaptée pour Self, est présentée dans [CUCH91].

La figure 1.5, issue de [US87], montre un exemple de prototypes dans le langage Self, concernant deux points avec des coordonnées cartésiennes sur un plan, une méthode d'addition de points et une méthode d'impression générale d'objets. La modélisation de ces points en objets Self est comparée à une modélisation en instances de classes Smalltalk.

Dans cet exemple, du côté Smalltalk, chaque point possède des valeurs de coordonnées x et y, ainsi qu'une référence vers la classe *Point*. Cette classe, comme toute classe Smalltalk (instance de la classe *Metaclass*), possède un nom, une super-classe (éventuellement nulle), des variables d'instances (implémentées par toutes ses instances) et des méthodes. La création d'un nouveau point s'effectue en envoyant un message de création de nouvelle instance à la classe *Point*. Le point est alors créé avec une référence vers cette classe et une implémentation de ses variables d'instance.

Du côté Self, tous les objets ont le même statut : un ensemble de slots (dont certains, les slots parents, guident la délégation) ainsi que la possibilité d'être "clonés" (dupliqués) pour créer de nouveaux objets. Chaque point possède donc deux slots pour les coordonnées x et y ainsi qu'un slot parent qui renvoie vers un objet contenant les caractéristiques communes de tous les points (méthodes de mise à jour des coordonnées, opération d'addition, lien parent vers un objet qui sait comment imprimer un objet quelconque). La création d'un nouveau point s'effectue en dupliquant l'un des points existants. Le nouveau point possède alors la même structure, mais évolue indépendamment de son point créateur.

Une particularité du langage Self réside dans la notion de *traits* [UCCH91]. Un *trait* de Self est le regroupement, au sein d'un objet, de tous les slots communs à un ensemble d'objets, qui deviennent des enfants du *trait* (d'où partage par délégation). Par exemple, l'objet qui possède les caractéristiques communes de tous les points dans la figure 1.5 est

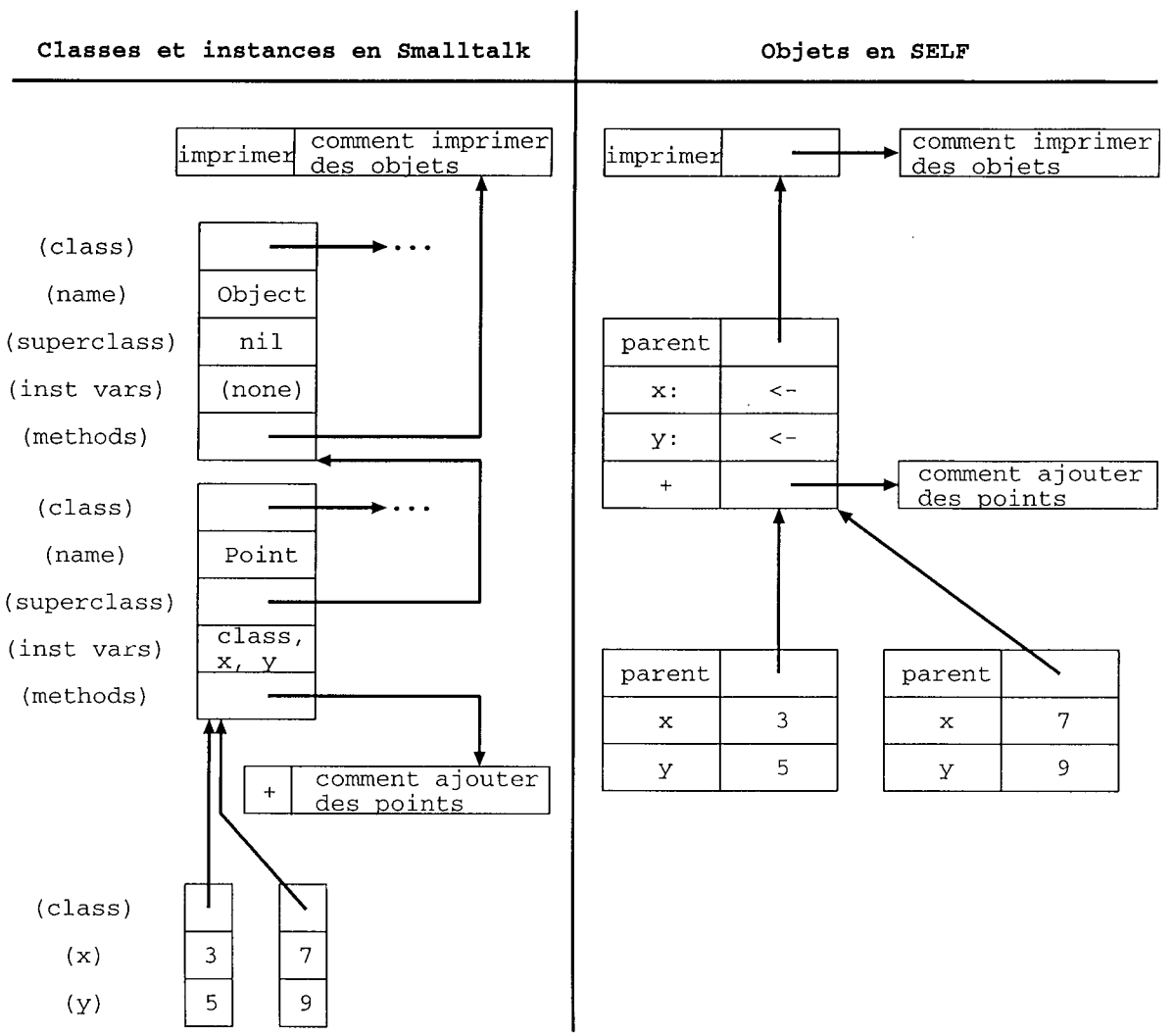


FIG. 1.5 – *Quelques prototypes dans le langage Self*

un *trait* de Self.

La notion de *trait* peut être considérée de plusieurs manières. En comparant conceptuellement avec les langages de classes, on peut dire que les *traits* remplissent le rôle factorisant des classes. Si, par contre, on désire faire une analogie (prototypes / classes), on peut comparer les *traits* à des classes abstraites (classes sans instances, servant uniquement à factoriser des propriétés communes à plusieurs classes plus spécifiques).

1.2.4.2 Quelques approches hybrides

Des approches intéressantes combinent les deux paradigmes de classes et de prototypes [LTP86, SCI89]. Les auteurs argumentent que la modélisation à base de classes est rigide (elle exige que toutes les instances d'une classe aient la même structure - celle de leur classe -) et que la modélisation à base de prototypes souffre de l'absence d'accès groupé aux objets.

Ils proposent une conciliation des deux aspects au sein d'un modèle qui combine, d'une part, hiérarchie de classes avec héritage, d'autre part, ce qu'ils appellent *hiérarchie d'objets* (*exemplaires* dans [LTP86]) avec délégation.

Un objet possède des caractéristiques (attributs, méthodes) ainsi que deux types de liens : vers sa classe et vers ses objets parents (respectivement appelés *schema* et *super* dans [SCI89]).

Dans ces approches, le mécanisme principal de réponse aux messages reste quand même la délégation. Les classes sont simplement un moyen d'accès groupé aux objets, et leur hiérarchie peut être simplement interprétée comme une contrainte sur la structure de la hiérarchie d'objets. Sciore annonce d'ailleurs la règle suivante dans [SCI89] : *Si* o_1 *est un objet de la classe* c_1 *et si* c_1 *est une sous-classe de* c_2, *alors* o_1 *doit avoir un objet parent* o_2 *de la classe* c_2.

Cette règle résume la philosophie de ces approches : les classes sont présentes dans le seul but de l'accès groupé aux objets. Elles ne possèdent de ce fait qu'une partie de la structure de leurs objets. Chaque objet possède sa propre structure individuelle.

Ces approches résolvent certains problèmes d'héritage multiple et favorisent la multi-représentation d'objets, puisque des objets de structure différente peuvent être attachés à la même classe, pourvu qu'ils respectent la hiérarchie de classes qui est associée à celle-ci.

Dans la figure 1.6, une hiérarchie de quatre classes est présentée, ainsi que quatre hiérarchies d'objets qui lui sont reliées. La classe *Moniteur* hérite des deux classes *Enseignant* et *Etudiant*, qui héritent toutes les deux de la classe *Personne*.

Un enseignant appelé *Alain* est représenté par une hiérarchie de deux objets ; le premier étant relié à la classe *Enseignant*, avec un parent relié à la classe *Personne*. Ce dernier représente *Alain* en tant que personne, et est complété par le premier pour ce qui est des caractéristiques d'*Alain* en tant qu'enseignant.

Un étudiant du nom de *John* est représenté d'une manière similaire, mais avec la classe *Etudiant*.

Deux moniteurs du nom de *Salim* et *Salima* sont aussi représentés. La différence au niveau de leurs hiérarchies d'objets respectives est qu'un objet de la classe *Enseignant*

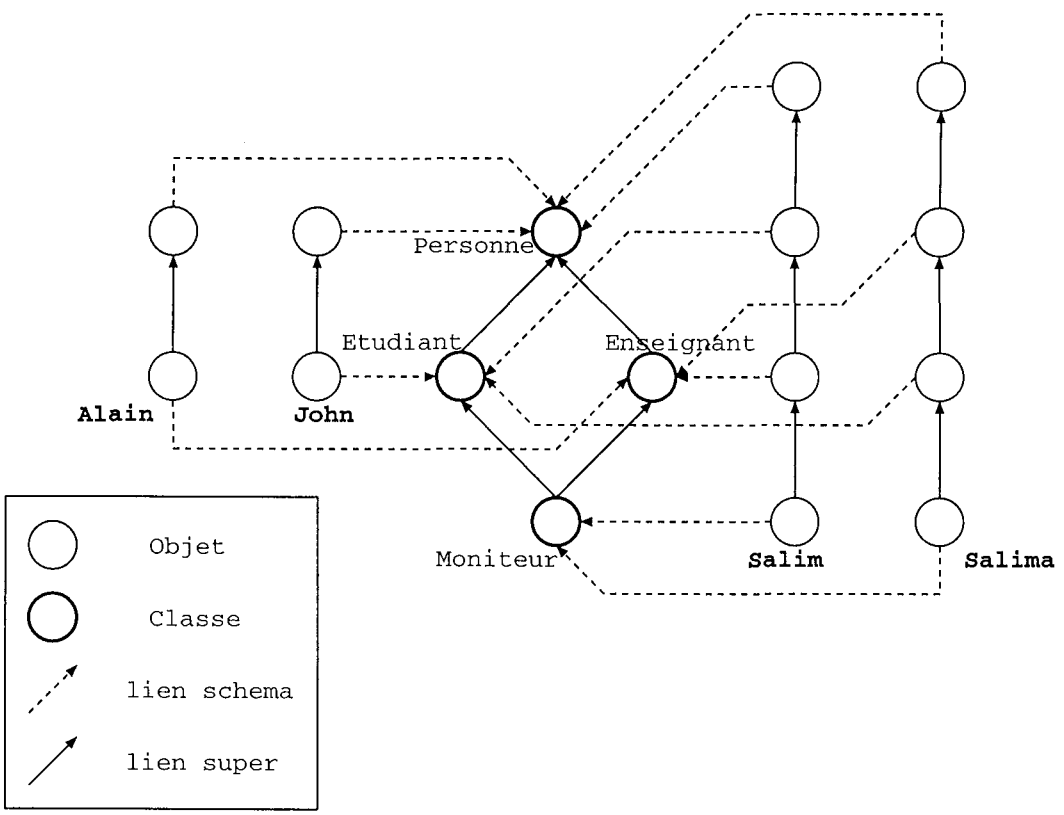


FIG. 1.6 – *Plusieurs hiérarchies d'objets asociées à une même hiérarchie de classes*

apparait en premier dans la hiérarchie d'objets de *Salim*, alors qu'un objet de la classe *Etudiant* apparait en premier dans la hiérarchie d'objets de *Salima*. Cela fixe en particulier la priorité dans la recherche des possesseurs de méthodes le long de la hiérarchie d'objets.

Ainsi, si les deux classes *Etudiant* et *Enseignant* ont toutes les deux une variable d'instance *grade* qui représentent respectivement l'année d'étude et l'ancienneté, l'envoi du message *grade* à *Salim* donnera son ancienneté en tant qu'enseignant. L'envoi du même message à *Salima* donnera son année d'étude en tant qu'étudiante.

1.2.4.3 Discussion

Les langages à objets fournissent une variété considérable de paradigmes. Nous avons situé les langages de prototypes par rapport aux langages de classes, mais d'autres paradigmes objet existent dans la littérature: langages de frames, d'acteurs, ... [MNC+89, NAP92].

Si les langages de classes ont été très vite adoptés par les systèmes de gestion de bases de données orientées objet, c'est principalement parce qu'ils favorisent l'accès groupé aux objets à travers les classes et du fait de leur typage fort des objets. Ces deux notions sont essentielles dans la gestion efficace de larges quantités de données et ne sont pas fournies par les langages de prototypes.

Cependant, de nouvelles applications, notamment dans le domaine documentaire, nécessitent des primitives de modélisation plus flexibles, comme celles des approches hybrides, présentées dans la partie 1.2.4.2. Les bases d'images par exemple, nécessitent l'utilisation de classes pour stocker des structures communes à tous les objets, mais nécessitent aussi des primitives de modélisation de structures individuelles, propres à chaque image.

Dans la suite, nous détaillons les caratéristiques des applications documentaires. Nous évoquerons les sytèmes d'informations textuelles, les bases d'images et quelques notions des systèmes multimédias.

1.3 Les bases de données documentaires

Dans certains domaines d'application, les informations ont une structure difficilement modélisable selon la terminologie des modèles de bases de données, puisque les objets manipulés représentent des entités réelles ayant des structures très différentes, même au sein d'un même groupe ou catégorie. Ceci est dû au fait qu'une notion supplémentaire vient s'ajouter à la structure, dans la représentation des objets, c'est la notion de *contenu*. C'est le cas des systèmes d'informations textuelles, des bases d'images, de sons, et en général des nouvelles applications multimédias.

Par exemple, la modélisation d'un article scientifique, comme entrée d'une base de données textuelle, peut se faire en tenant compte de l'aspect structure (titre, sections, paragraphes, ...) ou de l'aspect du contenu (ce que contient l'article comme information), qui peut revêtir une forme objective (suite de mots, de groupes de mots, de phrases, ..., relations entre ces primitives, ...), référentielle (références explicites à des ouvrages, des domaines d'étude, ou implicites, comme les références culturelles) ou purement subjective

(ce qu'inspire sa lecture, ...).

Dans ce genre d'informations (textes, images, ...), on distingue en général, aux désignations près, trois niveaux de description des objets manipulés [ORI92, WNM+95] :

- données brutes ou de représentation ; elles constituent le contenu des objets tel qu'il est représenté physiquement, sans analyse ni indexation (suite de caractères, matrice(s) de pixels, ...),
- données contextuelles ou de catalogage ; elles représentent des données concernant le contexte des objets (longueur ou largeur d'images, nombre de mots de textes, date de production, nom de l'auteur, ...),
- données sémantiques ou de description ; elles concernent l'ensemble des informations englobées, référencées ou connotées par le contenu de l'objet. Ce sont les plus difficiles à modéliser (puisqu'elles diffèrent d'un objet à un autre, au sein de la même classe d'objets) et donnent lieu à des traitements divers et variés. Elles peuvent revêtir un aspect *objectif* lorsqu'elles concernent des informations du sens commun, ou *subjectif*, lorsque leur interprétation varie avec l'utilisateur.

Contrairement aux modèles de bases de données où la modélisation d'un objet se fait à travers sa structure (modèles dits structurés), la modélisation de ce genre d'informations nécessite une phase d'indexation de la base par un certain nombre de primitives permettant de caractériser le contenu des objets. Ces primitives peuvent être des mots-clés, des indices graphiques élémentaires, ... Elles peuvent être attachées manuellement à l'objet, ce qui pose le problème des informations subjectives [KAT92], ou encore extraites automatiquement des données brutes, par des techniques de reconnaissance des formes, d'analyse du langage naturel, ...

Le modèle de données est en fait un modèle d'indexation manuelle, plus qu'un modèle de structuration, et ce qui le détermine principalement est l'objectif du système en terme de requêtes et services. La catégorisation des objets manipulés étant difficile, vu la différence notable de leurs structures, la notion de classe est presque inexistante et les requêtes, tout autant que les modèles, mettent l'accent sur la structure individuelle de chaque objet.

Dans une base de données structurée (relationnelle, orientée objet), une entrée de la base (un tuple, un objet) correspond à une construction (relation, classe) ayant une structure fixe, représentée par des attributs qui désignent chacun une partie bien définie de la structure et qui font référence à une entité d'un certain type (valeur d'un domaine, objet d'une classe). La structure des objets manipulés est donc connue à l'avance simplement si on connait à quelle entité conceptuelle (relation, classe) ils appartiennent.

Dans les bases de données documentaires, par contre, les documents (textes, images, vidéos, ...) n'ont en général pas de structure figée. S'il est possible par exemple de structurer un texte en sections, paragraphes, ... il est difficile, en revanche, de prévoir cette structure, a priori, à un niveau supérieur d'abstraction (niveau conceptuel). On peut prévoir une structure *générique* partagée par un certain nombre de textes, mais la structure réelle (*spécifique*) de chacun n'est connue qu'une fois qu'on dispose réellement de l'objet

texte : on voit alors qu'il est composé, par exemple, de deux sections, chacune composée à son tour de trois paragraphes. Un problème important réside d'ailleurs dans la granularité des descriptions ; en d'autres termes, jusqu'à quel degré aller dans la granularité des composants [RT88] ?

Au delà de la modélisation des applications, cette structure, une fois obtenue, peut servir de primitive d'indexation de la base de données. Cette dernière opération revient à définir des modes de stockage de l'information en vue de certains types de traitements. Ainsi, les données sont organisées en fonction des traitements qui sont amenés à être effectués sur la base.

Une base de données structurée, relationnelle par exemple, est indexée par rapport aux attributs de relations, qui sont le plus fréquemment utilisés au niveau des requêtes [DA82]. Ainsi, dans une base comprenant la relation "Cours" formée des attributs "intitulé", "date", "professeur" et "numéro de salle", et utilisée le plus souvent pour éditer les emplois de temps d'utilisation des salles (cours qui vont y avoir lieu), les tuples de la relation cible "Cours" sont indexés selon l'attribut cible "numéro de salle". La base sera donc organisée de telle manière que les tuples ayant le même "numéro de salle" soient rapidement accessibles dans leur ensemble. La structure des relations sert donc de base à l'indexation.

L'indexation d'une base de données documentaire est quelque peu différente, en raison du manque de structuration évoqué précédemment. Il demeure néanmoins qu'elle est toujours guidée par les requêtes. La base est organisée de telle manière à accéder rapidement à tous les documents faisant référence à un concept (textes comprenant un mot ou un groupe de mots, images de voitures, ...), si on s'attend à formuler des requêtes portant sur ce concept.

Cette indexation basée sur le contenu est une indexation *a posteriori* des documents de la base. En effet, en l'absence d'une structure fixe des documents pour prévoir leur indexation *a priori*, cette indexation n'est effectuée qu'une fois que le contenu réel des objets est connu. Il s'agit là d'une indexation par rapport au contenu et non à la structure.

Les requêtes opérées sur des bases de données structurées ont pour cible des entités conceptuelles (relations, classes) et sont, dans le cas le plus simple, des conditions du type "*attribut = valeur*" (ex : Tous les cours donnés par M. Dupont). Des requêtes plus compliquées sont obtenues par assemblage de sous-requêtes plus élémentaires. Comme la structure des objets manipulés dans la base est connue, elle est utilisée pour formuler les requêtes.

Dans une base de données documentaire, les requêtes peuvent être formulées selon au moins deux aspects : l'aspect structurel (ex : tous les textes ayant 3 sections, dont la seconde contient 5 paragraphes) et l'aspect contenu (ex : toutes les images à prédominance de bleu). Des requêtes mixtes, faisant intervenir les deux aspects en même temps, peuvent aussi être utiles (ex : tous les textes dont les titres contiennent le mot "database" et qui contiennent moins de 6 sections).

Dans la suite de cette partie, nous exposerons les particularités des informations mises en jeu dans les bases de données textuelles (1.3.1) et les bases d'images (1.3.2). Dans les deux cas, nous aborderons les différents niveaux de description, les types de requêtes et

éventuellement, les techniques d'indexation.

1.3.1 Systèmes d'informations textuelles

Les informations manipulées dans de tels systèmes sont majoritairement des unités textuelles. Ce genre d'informations pose des problèmes particuliers tant au niveau de la modélisation qu'à celui de l'accès et de la recherche d'information [LOE94, LNM94].

Dans ce qui suit, nous exposons deux types d'approches pour la modélisation de textes. Les approches fondées sur le contenu (*full-text*), décrites dans 1.3.1.1, s'intéressent au contenu réel des textes (mots, groupes de mots, ...). Les approches structurelles, décrites dans 1.3.1.2, s'intéressent plus à leur structure (chapitres, sections, ...).

1.3.1.1 Approches fondées sur le contenu

Un texte peut être vu simplement comme étant une suite de mots. L'indexation se fait par des mots-clés et les requêtes sont aussi spécifiées par des mots-clés, avec éventuellement quelques opérateurs logiques. Il s'agit là d'une manière simple de modéliser, et les données et les requêtes.

Pour ce qui est de la classification structurelle dans ce genre d'approche, on dispose de deux types d'objets : le mot et le texte. Toutes les entités intermédiaires sont en général occultées et on ne s'intéresse qu'au contenu des textes.

Après indexation, un texte est représenté par un ensemble de mots-clés. Une classe de contenu, reliée à un ensemble C de mots-clés, est l'ensemble de tous les textes représentés par un sur-ensemble de C. Nous désignerons dans la suite l'ensemble C des mots-clés et la classe correspondante par le même nom : C.

D'après [FAL85], et dans un souci de réutilisation de techniques d'accès pour bases de données structurées, ces mots-clés peuvent être vus comme des valeurs d'attributs dans une base de données structurée (relationnelle), à la différence qu'il y a autant d'attributs que de mots-clés (donc un nombre variable) et que ces attributs n'ont pas de nom spécifique.

Si l'on oblige les mots-clés à être dans un ensemble prédéfini, on peut représenter le texte par un vecteur dans l'espace engendré par la base de mots-clés possibles. Les coordonnées du vecteur peuvent être binaires, ou variant dans l'intervalle [0,1], si l'on autorise des indexations par poids de pertinence (approches à base de thésaurus de mots, par exemple). Dans ce dernier cas, les requêtes peuvent être flexibles, dans le sens où le résultat d'une requête n'est pas l'ensemble des textes qui la vérifient, mais une liste de texte ordonnée par ordre d'adéquation à la requête.

D'autre part, un thesaurus de termes peut tenir compte des liens entre termes. Ceci permet, en particulier, de structurer la base de l'espace de description et de pouvoir inférer des poids de pertinence dans l'indexation, ou des degrés d'adéquation dans les requêtes, selon le type de relation qui existe entre mots-clés (synonymie, subsomption, ...) [HCK90].

A ce titre, si la notion de sous-typage sert de base pour les notions de généralisation et spécialisation des classes structurelles, la notion de subsomption de sens de mots-clés peut

faire de même pour les classes de contenu. Une classe C généralise une classe C' si pour chaque mot-clé de C', il existe au moins un mot-clé de C qui le subsume (qui le généralise).

Le langage naturel n'étant pas simplement une suite de mots, il est normal que le domaine de l'interrogation de bases de données textuelles se soit orienté petit à petit vers des traitements plus compliqués que la simple recherche par mots-clés, et donc vers l'analyse du langage naturel.

Dans [SCH90], par exemple, les différents liens syntaxiques entre mots ou groupes de mots sont explorés ; l'indexation des textes est effectuée en conséquence. Les requêtes sont formulées en langage naturel et passent par un compilateur qui en extrait les structures grammaticales porteuses de sens, analogues à celles servant de primitives d'indexation de la base (groupe de mots, compléments de noms, ...).

Dans [MAR91], en plus des liens syntaxiques, des liens sémantiques et pragmatiques, dépendant du contexte, sont pris en compte entre les différentes portions du texte. Un texte est vu comme un réseau dont les noeuds (portions de texte) sont liés par trois types de liens (syntaxique, sémantique, pragmatique). La logique modale permet de définir un modèle de requêtes basé sur des prédicats et des propositions, tenant compte, en particulier, du profil de l'utilisateur (pour l'inférence de liens pragmatiques).

1.3.1.2 Approches structurelles

La structure des documents (des textes, en particulier), a fait l'objet de plusieurs études et a donné lieu à des normes internationales : ODA (Office Document Architecture [ISO87]), SGML (Standard Generalized Markup Language [ISO86]) et son application aux pages du World Wide Web : HTML (Hyper-Text Markup Language), ou encore HyTime (Hypermedia/Time-based structuring language [ISO92]). Si ODA s'intéresse à la dualité des structures physique (régions, localisations) et logique (sections, paragraphes, titres), SGML offre un langage de description hiérarchique de la structure et du contenu d'un document. Sa principale caractéristique est la notion de balisage qui permet de désigner le début et la fin de chaque partie de la composition, et en particulier d'assurer la hiérarchisation de la description grâce à l'emboîtement des balises [DSA91]. Les lignes qui suivent donnent un exemple simple de description d'un texte par SGML.

```
<livre>
<titre>Le vieil homme et la mer</titre>
<auteur>E. HEMINGWAY</auteur>
<paragraphe>Il \'etait une fois un vieil homme, tout seul dans son bateau
qui p\^echait au milieu du Gulf Stream. En quatre-vingt-quatre ...
</paragraphe>
<paragraphe>Chaque soir le gamin avait la tristesse de voir le vieux
rentrer avec sa barque vide. Il ne manquait ...
</paragraphe>
...
</livre>
```

Il est effectivement possible de traiter les données textuelles d'une manière structurée, mais les modèles pour bases de données structurées souffrent de certaines limitations dans la représentation de ce type de données.

La représentation de données textuelles dans une base de données relationnelle par exemple, occasionne un certain nombre de difficultés conceptuelles, telles que la structure des textes ou encore les traitements de données. Avant de saisir les textes, il faut prévoir leur structure dans un schéma de relation ; ce qui est difficile du fait de la structure variable et complexe des textes. On a souvent recours à plusieurs relations. Les traitements s'en ressentent. Afin de réunir les informations disséminées sur plusieurs relations et concernant une seule structure complexe, comme un texte, on est obligé d'avoir recours à des opérateurs complexes et coûteux en temps d'exécution [MAC91b, GD92].

Une manière efficace de représenter la structure générale d'une catégorie de textes est d'utiliser une grammaire formelle, à l'instar des DTDs de SGML (Document Type Definition) [ISO86]. Une grammaire peut représenter des agrégations, des groupements, des alternatives, ... autant de concepts utiles à la représentation des structures complexes des données textuelles. Ce type d'approche est suivi dans plusieurs systèmes d'informations textuelles, dont le système Maestro [MAC90, MAC91a, MBHL91], le système TOMS [DWL92] ou encore le modèle des P-Strings (*Parsed Strings*) [GT87] qui a la particularité de reposer sur une algèbre d'opérateurs grammaticaux [GPVG89].

Dans Maestro, par exemple, un texte est décomposé récursivement en parties, chacune ayant des caractéristiques intrinsèques (attributs), un contenu et éventuellement des sous-parties. Un contenu est une suite de mots. Un texte peut éventuellement faire référence à d'autres textes.

Une classe (ou type) de textes n'étant pas définitivement structurée, on fait appel à des grammaires formelles pour décrire sa structure. L'exemple suivant est extrait de [MAC91a] et donne une structure type, où les suffixes * et + signifient respectivement liste et liste non vide, du terme qui les précède. Chaque ligne représente un type de texte (définitions hiérarchiques). Le symbole * initial signifie simplement que la structure peut être partagée, par plusieurs types.

doctype : MyType

```
[< *          (Title, Abstract, Section*, Citation+) >
 < Abstract   (Paragraph+) >
 < Paragraph  (Sentence+) >
 < Section    (SectionHeading,
              (Paragraph+ | (Paragraph*, SubSection+))) >
 < SubSection (SectionHeading, Paragraph+) >
]
```

Cette définition permet en particulier de désigner les composants d'un type de textes par des rôles (qui sont aussi leur type). Ainsi "Abstract **of** MyType" désigne le composant "Abstract" d'un objet de type "MyType".

Les requêtes sont assez diverses : sélectionner l'ensemble des instances d'un type de textes ; sélectionner les instances d'un type dont le contenu vérifie certaines conditions ; sélectionner les instances d'un type qui vérifient certaines conditions de structure ; ...

Les requêtes complexes sont des imbrications de requêtes élémentaires. Un exemple de requête complexe est de rechercher les textes scientifiques ayant 2 sections avec 3 paragraphes chacune et tels que le second paragraphe de la première section contient le mot "théorie".

Un autre type d'approche est représenté par des modèles dits modulaires (Grif [QV89] ou MULTOS [LUT89], par exemple) dont les notions de base sont : la décomposition conceptuelle de l'information en modules de plus en plus spécifiques, le typage des objets en modules, l'agrégation, le groupement, et éventuellement le lien de généralisation entre modules.

Dans le modèle Grif [QV89], par exemple, un langage appelé S permet de définir la structure des types de données manipulés sous forme de modules, comme dans l'exemple suivant :

```
STRUCTURE Book;
    Book=BEGIN
        Title=TEXT;
        Author=TEXT;
        Body=LIST OF(Chapter);
        END;
END

STRUCTURE Chapter;
    Chapter=BEGIN
        ChapterHead=TEXT;
        ChapterBody=LIST OF(Paragraph);
        END;
END

STRUCTURE Paragraph;
    Paragraph=CASE OF
        SimpleParag=TEXT;
        Item-List=LIST OF(Item=TEXT);
        Quotation=TEXT;
        END;
END
```

Deux autres langages P et T permettent de gérer respectivement les caractéristiques de présentation et de transfert des textes.

Dans les exemples précédents, la séquence "BEGIN .. END" permet de définir une agrégation d'une structure modulaire en sous-structures modulaires plus élémentaires,

chacune repérée par un rôle : un Chapitre est l'agrégation d'un texte (son entête) et d'une liste de paragraphes (son corps). Le constructeur "LIST OF" permet de définir une sous-structure comme un groupement d'objets d'une structure donnée : le corps d'un livre est une liste de chapitres. Le constructeur "CASE OF" définit une alternative dans la composition d'une structure : un paragraphe peut être du texte simple, ou une liste de textes, ou encore une référence (Quotation).

Il est possible d'assigner des attributs (langue d'un livre, ...) à chaque structure, par une option du langage S.

Les approches modulaires permettent de définir des structures arbitrairement complexes de documents textuels, au même titre que les approches grammaticales. Elles ont cependant l'avantage de la réutilisabilité et du partage de composants, facilités par le fait de désigner les différents composants d'un module par des rôles particuliers (*entête* et *corps* d'un livre, par exemple).

Un aspect important de la modélisation de textes est la prise en compte des liens entre entités textuelles. Les types de liens cités jusqu'à présent entre objets textuels, ont été principalement hiérarchiques : un objet est composé d'autres. Mais, d'autres types de liens sont nécessaires dans les bases de données textuelles, notamment en ce qui concerne la navigation. Les techniques d'hypertexte [CON87, DV90, cac90] identifient ces différents types de liens, en définissant des points de branchement, des techniques de retour, et des traitements plus spécifiques (personnalisation, ...). On y distingue en particulier les liens unidirectionnels avec un document source "pointant" vers un document cible, des liens bidirectionnels où les deux documents protagonistes ont des rôles similaires.

Les systèmes présentés jusqu'à présent ne permettent pas de modéliser naturellement différents types de liens entre entités textuelles. Par exemple, dans Maestro, non destiné initialement à représenter des liens non hiérarchiques, un lien est représenté artificiellement par un document sans contenu [MBHL91]. Les deux parties du lien sont représentées par des attributs du document lien, chacun faisant référence à l'endroit des documents concernés où les références sont définies. Le type "Link" est défini comme suit :

document Link **is null with** Source, Target **reference**, LinkType **text**.

Cette représentation peu naturelle présente néanmoins certains avantages. Ainsi, la manipulation des liens profite des opérations bien définies sur les documents. Une requête sur des liens est exprimée de la même manière qu'une requête sur des documents. En effet, pour rechercher, par exemple, les documents citant un document particulier, il suffit de sélectionner l'attribut "Source" de tous les documents de type "Lien" où l'attribut "Target" est contenu dans le document en question.

1.3.1.3 Discussion

Nous avons présenté quelques caractéristiques des systèmes d'informations textuelles, ainsi que la manière dont elles sont gérées dans un certain nombre de systèmes.

Il apparait dès l'abord, qu'au sein de ce type de données, co-existent deux notions parallèles : la structure et le contenu. Elles sont étroitement liées au sein d'un document

textuel sans pour autant être dépendantes. Elles donnent lieu à des traitements différents, et donc à des modèles et systèmes différents.

Les modèles documentaires délaissent la structure pour se focaliser sur le contenu des textes. Les modèles structurels tiennent compte de la structure dans la modélisation du contenu des documents, et apportent des solutions aux problèmes posés par la complexité de la structure des documents textuels, mais offrent peu d'outils pour la manipulation basée sur le contenu.

Les approches grammaticales offrent un cadre puissant de modélisation des structures, mais présentent des limitations quant au partage, à la réutilisation et l'intégrité des données manipulées ; les approches modulaires qui apportent des solutions à ces problèmes, pêchent par leur structure trop hiérarchique, qui fait que les modules de bas niveau dépendent fortement des modules de haut niveau.

Enfin, la modélisation des liens non hiérarchiques (de type hypertexte) reste un handicap sérieux.

1.3.2 Bases d'images

L'expression "base d'images" désigne une collection d'images stockées sur un support numérique. Chacune de ces images, dans sa forme la plus simple, est une matrice de pixels dans laquelle chaque pixel est affecté d'une valeur représentant sa luminosité. Cette valeur peut se situer dans différents référentiels (valeurs booléennes pour le cas le plus simple des images en noir et blanc, valeurs entières prises dans un intervalle pour le cas des images à niveau de gris ou encore n-uplets de valeurs entières de couleurs primaires pour le cas des images en couleurs).

Les images numériques peuvent être obtenues de deux manières différentes : en scannerisant des images réelles ou en créant de toutes pièces (par programme) ce qu'on appelle des images virtuelles.

Une image ne se limite pas à la définition très simpliste de matrice de pixels. Elle met en jeu des informations de natures bien diverses. Hormis les données brutes ou *de représentation* (pixels, fonction de luminosité) et les données externes ou *de catalogage* (contextuelles : photographe, date, ...), les images véhiculent des données sémantiques ou *de description* ou encore *d'indexation.* En outre, et au delà de l'aspect informationnel (plus ou moins objectif), une image intervient aussi à un niveau émotionnel (purement subjectif).

La modélisation et la recherche d'information dans une base d'images doit donc tenir compte d'éléments appartenant à des univers de discours très différents.

En se limitant aux images réelles scannerisées, le contenu d'une image peut représenter des scènes très diverses : scènes de rue, d'intérieur, paysages, photographies de personnes, de bâtiments, logos de compagnies, documents écrits, images médicales, ...

Les systèmes de recherche d'information dans des bases d'images sont, en général,

orientés vers une application particulière parce que chaque type d'application fait intervenir des besoins, et donc des traitements, particuliers.

Divers systèmes "orientés application" sont décrits dans la littérature. Le système présenté dans [WLS95] est dédié à la représentation de logos de compagnies. Il est utilisé, en particulier, pour déterminer, lors de l'arrivée d'un nouveau logo dans la base, s'il n'y a pas de similarité frappante avec d'autres logos existants. Le système présenté dans [HHLC92] est dédié aux images médicales. Il permet de détecter des configurations anormales dans certaines radiographies.

En général, on distingue, dans la modélisation des données sémantiques des images, les données *objectives* (indices graphiques) des données *subjectives* (connotations personnelles, culturelles, ...). Certaines études [KAT92] tentent même d'inférer certaines données subjectives à partir de primitives purement graphiques !

Comme nous l'avons mentionné plus haut, ces deux types de données sémantiques d'indexation viennent compléter les données brutes de représentation et les données externes de catalogage de l'image.

1.3.2.1 Représentation des indices graphiques

Si on se limite à l'aspect objectif des images (celui des indices graphiques), le contenu d'une image peut être indexé de différentes manières : croquis, polygones, régions, zones homogènes de pixels, ...

Dans certains cas, l'indexation est automatique, grâce à l'extraction automatique d'indices visuels, qui fait appel à des techniques de traitement d'image et de reconnaissance de formes [TOU91, HK92, RS92a, RS92b].

D'un autre côté, certaines approches reposent sur une indexation manuelle, plus ou moins assistée, du contenu des images [HLMM92, HHLC92] ; on parle alors d'indexation sémantique.

Dans [HK92], les images traitées sont des tableaux de maîtres. Elles sont représentées, du point de vue du contenu, par des croquis formant une sorte "d'ébauche" de la structure générale de l'image.

Les requêtes sont également spécifiées par des croquis, à main levée. Après avoir représenté ces croquis selon la même technique que pour les croquis des images de la base, le système recherche les images dont les croquis ressemblent au croquis de la requête (algorithme de mise en correspondance des modèles de croquis).

Dans [TOU91], l'analyse de l'image repose sur les pixels. L'image est découpée selon la méthode des Quadtrees inversés, en régions homogènes. Pour cela, elle est divisée récursivement en quatre régions rectangulaires et la division s'arrête au niveau des régions ayant une luminosité uniforme.

Les requêtes sont spécifiées par des primitives graphiques, modélisées de la même manière. Les index sont parcourus, afin de trouver les images dont certaines régions correspondent approximativement à ces primitives.

Dans [RS92a, RS92b], un type d'images est représenté par une structure contenant des composants positifs (types d'objets obligatoires) et négatifs (types d'objets interdits), et des liens entre composants. Le contenu d'une image est représenté par un "graphe relationnel attribué" (ARG) dont les sommets sont des objets composants et les arcs des relations. Chaque composant appartient à un type d'objets, et possède un certain nombre d'attributs vis à vis de l'image (degré de reconnaissance, ...). Les liens entre composants traduisent des relations de positionnement des objets les uns par rapport aux autres.

Les requêtes s'adressent à un type d'images particulier, et contiennent des conditions (clauses en forme normale conjonctive) sur les composants, et des relations entre ces clauses (entre les objets répondant à ces clauses).

Dans [HHLC92], une image est composée de régions disjointes. Chaque région a un barycentre et un poids. Le centre de masse de l'image est calculé en fonction de ces valeurs sur toutes les régions. Chaque région est alors repérée par sa distance au centre de masse de l'image, et par un angle avec la région de plus grand poids. Ces valeurs servent de base pour l'indexation et les requêtes.

Dans le système IIDS [CYDT88], les composants de l'image sont des régions de formes diverses. A chaque région est associé son rectangle englobant et des relations spatiales sont calculées sur ces rectangles. Les auteurs utilisent une panoplie complète de relations spatiales possibles entre deux rectangles quelconques dans un plan.

Le système a une architecture de système expert basé sur le "raisonnement spatial" : les faits sont des relations spatiales entre régions et les règles sont des règles générales d'induction de relations spatiales les unes des autres. Le système permet ainsi de décrire les images d'une manière minimale et d'inférer de nouvelles descriptions lors de l'interrogation.

1.3.2.2 Interprétation du contenu des images

Nous entendons par "interprétation" des images la représentation de leur données sémantiques subjectives. Cette représentation peut se baser sur les indices graphiques ou encore s'en passer.

Dans certaines approches, le contenu des images est décrit par du texte: liste de mots-clés dans [HAL89, SMA94, KKH94], phrases en langage naturel dans [KKL91] ou semi-naturel dans [HLMM92]. Les indices graphiques sont sans importance et on se retrouve dans une situation similaire aux bases de données textuelles (indexation vectorielle, degrés d'adéquation, éventuellement analyse du langage naturel, ...).

Dans [HLMM92], par exemple, le contenu d'une image est décrit, selon un modèle linguistique (adjectif, nom, verbe), par un certain nombre de primitives :

Primitive	Exemple
objet	cheval
attribut objet	blanc cheval
relation()	court()
relation(objet)	court(cheval)
relation(objet,objet)	mange(enfant,pomme)

Le système est implémenté au dessus d'un SGBD relationnel, et les requêtes sont traduites en SQL.

Dans d'autres approches, l'interprétation des images s'appuie sur les indices graphiques. A la manière de la norme ODA [ISO87, DSA91] qui distingue, dans le contenu des documents, une hiérarchie conceptuelle de la hiérarchie structurelle, chaque fragment de contenu est associé à un objet conceptuel. Le niveau conceptuel consiste en un certain nombre d'objets, reliés entre eux par des liens de natures différentes.

Ainsi, dans le système MULTOS [MRT91] (qui fait clairement référence à ODA), chaque fragment de contenu de l'image (région élémentaire) est associé à un objet décrit dans un modèle sémantique (cf. partie 1.2.2) fournissant les notions de classes d'objets, d'agrégation et de généralisation. A une image peuvent être associés plusieurs graphes du modèle sémantique représentant chacun une interprétation possible de l'image. Les requêtes peuvent à la fois s'adresser au contenu des images ou à leur structure logique (au niveau du modèle sémantique).

Les idées véhiculées par MULTOS sont reprises dans [MEG95] dans un cadre très formel. Les deux niveaux de la description d'une image sont formalisés séparément. Le niveau graphique est représenté par un ensemble π de régions et une fonction totale de cet ensemble vers une distribution de couleurs. L'aspect de l'interprétation est représenté par un ensemble O d'objets, un ensemble C de classes et de liens entre eux représentant, en fait, un graphe sémantique. L'image en entier est considérée comme étant la réunion de ces deux niveaux de description ainsi que d'une fonction partielle de l'ensemble des parties de π vers O, associant à chaque ensemble de régions une interprétation sous la forme d'un objet.

Une approche similaire, également basée sur un modèle sémantique, est suivie dans le système VIMSYS [GWJ91]. Elle se distingue par la prise en compte d'événements pouvant survenir au cours de la vie des objets d'interprétation du contenu des images.

Encore une fois, le niveau graphique est séparé du niveau de l'interprétation. L'application est décrite selon quatre plans différents : le plan IR pour les aspects graphiques des images, le plan IO pour les objets d'interprétation, le plan DO pour la description conceptuelle du domaine et la plan DE pour les événements.

Le formalisme du système CORE [WNM+95] est sans doute l'un des plus complets pour décrire des objets multimédias, en particulier les images. Comme dans [MEG95], une formalisation des concepts de primitives graphiques, d'objets d'interprétation ... est effectuée. Elle est même complétée par la notion d'état du document multimédia (persistent,

transient, complet, incomplet, ...).

CORE offre cependant la particularité d'être un système extensible, en ce sens qu'il ne s'agit pas d'un système unique, mais d'un moteur de recherche d'information basée sur le contenu, pour les systèmes d'information multimédias. Plusieurs instanciations de ce modèle ont donné lieu à des sytèmes divers (reconnaissance de visages, de logos commerciaux, ...).

Enfin, dans le système EMIR^2 [MEC95], les deux niveaux de description des images (graphique et d'interprétation) sont à leur tour divisés en plusieurs parties, si bien que la description de l'image se trouve décomposée en cinq vues.

La vue principale est la vue *structurelle*; elle est composée d'*objets images* qui sont en fait des régions imbriquées de l'image liées par un lien de composition spatiale. Autour de cette vue gravitent les quatre autres, à savoir la vue *physique* (description matricielle de l'image), la vue *perceptive* (éléments de texture, de couleur et de brillance), la vue spatiale (éléments géométriques) et la vue *symbolique* (interprétation sémantique en termes de classes d'objets et de propriétés sémantiques).

Dans ce dernier modèle, comme dans tous les autres, se retrouvent les deux aspects fondamentaux de la modélisation des images : l'aspect graphique objectif (dans les vues autres que symbolique) et l'aspect plus subjectif de l'interprétation (dans la vue symbolique).

1.3.3 Systèmes d'informations multimédias

Dans cette section, nous considérons le cas particulier des bases de données multimédias. Nous aborderons leurs caractéristiques singulières, du point de vue de la modélisation et de l'accès.

Dans le terme "multimédia", le mot "média" est le pluriel de "medium", et fait référence aux moyens de communication. Un document multimédia est un assemblage de données de différentes formes : données structurées alphanumériques, données statistiques (tableaux), textes, données sonores ou visuelles (graphes, images, séquences vidéo) [OD91].

La nature hétérogène de ce genre d'informations requiert des moyens de stockage particuliers pour chaque forme ou média, des moyens d'accès différents, des utilisations différentes, même si un document multimédia consiste en fait en un seul objet, composé de plusieurs parties hétérogènes.

Le domaine des bases de données s'est ouvert au multimédia pour différentes raisons [MAS87] : l'évolution des supports de stockage, des interfaces homme-machine, la standardisation des protocoles de communication, l'évolution des modèles sémantiques de données, l'approche objet, ... Cependant, la taille des données, notamment sous certains médias (image, son), reste le problème principal ; aussi, une attention particulière doit être accordée au modèle de données retenu et aux techniques de stockage et de recherche dans la base.

Dans la suite, nous nous intéresserons à la modélisation d'informations multimédias sous deux angles différents : l'aspect structurel et l'aspect fonctionnel. Alors que le premier

s'intéresse aux primitives de modélisation de la structure des documents multimédias, le second s'intéresse, lui, aux primitives de traitement (évolutivité, accès, recherche, ...).

1.3.3.1 L'aspect structurel

Les documents multimédias ont une structure d'une complexité physique et conceptuelle. La complexité physique provient de la différence des supports et donc de la décomposition physique des documents, alors que la complexité conceptuelle rejoint la difficulté plus générale, de représentation d'objets à structures complexes.

Le modèle de données doit donc fournir les primitives essentielles de représentation d'objets complexes (classes, instances, généralisation / spécialisation, agrégation, groupement, identité d'objet, ...), et en plus la possibilité de définir des liens sémantiques spéciaux entre les entités (composition, dépendance, traduction inter-média, position relative, ...), en particulier pour des applications hypertexte (ou hypermédia) [CCJ+93, CPC94].

L'approche objet, de par sa flexibilité structurelle (permettant en particulier la gestion d'informations complexes), et de par son encapsulation des données et des traitements (permettant en particulier une gestion plus souple de l'évolutivité des applications), offre le plus d'avantages pour la modélisation de documents multimédias.

La notion de composition d'objets revêt, dans ce cadre, une importance primordiale, puisque les documents sont fondamentalement composites. Il est donc très utile de gérer deux hiérarchies "orthogonales" pour le schéma de l'application : la hiérarchie d'héritage classique et la hiérarchie de composition. Il est en outre intéressant de différencier entre les différents types de liens de composition (exclusive, dépendante, ...) [WKL86, KBG89].

Un exemple de modèle de données multimédias, défini pour le SGBDOO Orion, est présenté dans [WKL86, WLK87]. Il se base sur la notion de classe et d'instances, d'objets intrinsèques et d'objets relations.

- Les classes sont des collections d'objets, et sont organisées en deux hiérarchies orthogonales (héritage et agrégation). Deux types d'agrégations sont considérés au niveau des classes : "est-composé-de" et "peut-être-composé-de", pour traduire des notions de composition obligatoire et optionnelle. Les opérations sur les objets sont gérées par des méthodes. Le lien de composition optionnelle est particulièrement utile dans le domaine des documents composites ou hypermédias [CPC94].
- Les objets intrinsèques sont les feuilles de la hiérarchie d'agrégation (composition) et contiennent des données terminales stockées dans l'un ou l'autre des médias.
- Les objets relations traduisent des relations sémantiques spécifiques, entre d'autres objets, à la manière du modèle E/A, permettant ainsi de saisir des liens de natures diverses.

1.3.3.2 L'aspect fonctionnel

La structure particulière des documents multimédias, nécessite des fonctionnalités particulières, que le système de gestion de ce genre d'informations doit être capable d'assurer

[WLK87]. Ces fonctionnalités concernent par exemple, le partage de données, la gestion des versions d'un même objet, l'accès aux données de la base (requêtes et navigation), la gestion des transactions concurrentes, ...

En ce qui concerne le partage des objets, la taille souvent importante des données multimédias fait qu'une attention particulière doit être accordée à l'optimisation de l'espace disque ; (d'après [WKL86], une image fixe digitalisée peut nécessiter 4 Mo, et une minute de bande sonore digitalisée, jusqu'à 480 ko).

Le partage d'objets peut être fait soit par l'intermédiaire de la notion de copie différée ou de la notion de référence. La différence réside au niveau de la propagation des changements : la première crée une nouvelle copie physique de l'objet partagé lorsqu'un objet partageant le modifie, l'autre gardant l'ancienne copie ; la seconde répercute les modifications dans tous les objets, en même temps.

Dans le cadre particulier de la conception de documents multimédias, la notion de version est primordiale. Le système doit fournir la possibilité de modéliser d'une part les versions temporelles d'un objet, d'autre part ses versions alternatives [CPC94].

Les versions temporelles d'un objet traduisent son évolution linéaire d'un état initial à un autre état. Les versions alternatives, quant à elles, traduisent une bifurcation dans l'historique de l'objet ; elles correspondent à des implantations différentes du même objet, dues par exemple à une remise en cause d'une implémentation précédente.

L'histoire d'un objet peut donc être modélisée par un arbre où les versions alternatives sont représentées par des noeuds frères, et les versions linéaires par des noeuds parents.

La gestion des transactions concurrentes à un même objet doit assurer en particulier la transparence des mises à jour non encore validées (au sein d'une même transaction) vis à vis des autres utilisateurs. Comme pour les bases de données classiques, la notion de transaction est aussi une primitive de reprise sur pannes, puisque les différentes actions entreprises doivent être validées ensemble ou abandonnées ensemble.

Enfin, en ce qui concerne la navigation et les requêtes, et outre les fonctionnalités de présentation des données sous différents médias, la notion "d'accès associatif" est très importante, puisqu'elle permet la navigation à travers les relations entre objets (les relations de composition, de référence, ou les relations sémantiques définies par l'utilisateur).

Les requêtes peuvent s'adresser à la structure du document (documents contenant du son, par exemple), son type (documents d'une certaine classe, par exemple) ou encore son contenu (document contenant le mot "multimédia", par exemple). Il est aussi utile de souligner l'importance de modules de détection d'indices (reconnaissance optique de caractères, reconnaissance de mots dans un discours, ...), pour des requêtes nécessitant des translations de média (les documents écrits digitalisés en images et contenant le mot "multimédia", par exemple), ce qui représente une utilisation avancée des informations multimédias.

1.4 Discussion

Après avoir passé en revue quelques approches de modélisation pour les bases de données en général, et dans les applications documentaires en particulier (avec un détour du côté des langages de programmation par objets), nous énonçons ici un certain nombre de remarques qui constituent les motivations de notre approche.

Flexibilité du schéma d'application

Une première remarque sur les modèles de bases de données cités dans les paragraphes précédents, concerne le schéma de l'application. Dans la majorité des modèles, le lien entre classes et instances, entre types et valeurs, conduit à un mécanisme habituellement appelé "instanciation". Les objets appartiennent à une classe qui fixe leur structure, leurs liens, et même leur comportement.

Dans certains cas, les bases d'images par exemple, les structures des objets manipulés ne peuvent être ainsi abstraites dans des classes (ou alors il faudrait créer une classe par objet !). Ainsi, une image peut être composée d'un arbre comme d'une personne ou encore d'une planète. Les modèles de données décrits dans la partie 1.3, permettent de tenir compte de cette diversification du contenu en différenciant clairement la structure (générique) des objets documentaires de leur contenu réel (spécifique à chaque objet). Cette différenciation conduit notamment à la définition de processus d'interrogation propres, qui rendent ces modèles difficiles à utiliser dans d'autres domaines que celui pour lequel ils ont été conçus. Un problème important réside dans leur interopérabilité avec d'autres applications, ou simplement leur inefficacité à gérer d'autres types de données que ceux qu'ils gèrent d'habitude (textes, images, ...).

Il est donc utile de disposer d'un modèle flexible du point de vue de la structure des objets, permettant de gérer aussi bien des données dont la structure est fixe (informations usuelles dans les bases de données) que des objets dont la structure est plus variée (textes, images, ...). Les classes d'objets doivent simplement représenter des primitives d'accès groupé aux objets. Ces derniers doivent pouvoir disposer d'une structure individuelle supplémentaire. Ainsi, il est intéressant de faire en sorte que les classes d'objets servent simplement à attacher une structure minimale aux objets, et non une structure fixe.

Une telle approche ressemble à ce qui a été présenté dans la partie 1.2.4.2, concernant des approches hybrides dans les langages de programmation par objets, situées à mi-chemin entre les langages de classes et les langages de prototypes. Nous souhaitons adapter les idées véhiculées par ces approches à la modélisation d'objets complexes à forte structure individuelle dans les bases de données.

Distinction valeurs / objets

Un autre point important à notre avis n'est pas toujours pris en compte dans les modèles de bases de données. Il s'agit de la différenciation claire entre la notion de valeur et la notion d'objet, au niveau des concepts fondamentaux du modèle de données.

En introduisant la notion d'identité d'objet comme primitive de modélisation [KC86] et d'interrogation [AK89], les modèles orientés objet, ont rompu définitivement avec les

problèmes dus à la non duplication des valeurs dans les bases de données relationnelles, et des modèles orientés valeur en général. Il a donc été possible de faire référence à une entité sans faire intervenir ses valeurs ou avoir recours à des artifices de clés [HUL89] ; il a encore été désormais possible d'avoir les mêmes valeurs pour des objets différents.

Cependant, la notion d'identité d'objet ne doit pas être omniprésente. Il est parfois utile de pouvoir faire référence à de simples valeurs. Peu de modèles [AK89, Da90] offrent cependant cette possiblité. Il est à notre sens important de faire cohabiter dans un même environnement de modélisation, les notions de valeur et d'objet.

Nous définirons les notions de *description* et *composition*. La description d'un objet est un ensemble de caractéristiques dont les valeurs ne sont pas des objets à part entière (la couleur d'une voiture, l'adresse d'une personne, ...). La composition d'un objet est un ensemble d'objets de la base auxquels il fait référence.

Typologie des liens de composition

Le lien entre deux objets qui font référence l'un à l'autre, est désigné sous plusieurs appellations : on parlera d'agrégation, de composition, d'association, d'utilisation, de référence, ... Nous regrouperons toutes ces appellations sous le terme générique de lien de *connaissance*. Un objet connait un autre s'il y fait référence dans sa structure.

A part le modèle du SGBD Orion [KBC+87, KBG89] (pour ce qui est des objets composites), peu de modèles fournissent comme primitive de modélisation une différenciation des types des liens de connaissance. Ceci est plutôt fait au niveau des méthodes de conception [SM88, DES90, BOO91, BCMS93].

Nous pensons qu'une différenciation doit être faite (au niveau du modèle de données) entre plusieurs types de liens de connaissance, et que cette différenciation doit reposer sur la réaction de l'un des objets à la disparition de l'autre. Par exemple, le concepteur doit pouvoir différencier la nature du lien entre un livre et ses chapitres, de celle du lien entre une image et ses objets composants, du moment que la disparition du livre entraîne celle de ses chapitres et que celle de l'image n'entraîne pas celle des objets qu'elle référence.

Le modèle de données doit donc fournir la possibilité de faire cette distinction, en offrant un certain nombre de types de liens ; chacun doté d'une sémantique claire et précise, qui repose sur la réaction d'un objet à la disparition de l'autre.

Prise en compte de relations sémantiques entre les composants d'un objet

Le concept de relation sémantique s'est imposé avec le modèle E/A comme l'un des outils de base de la modélisation sémantique des données. Il était particulièrement destiné à des utilisations dans le cadre d'un modèle structuré, orienté valeur (Record-Based) [KEN79].

Ce genre de modèles n'accorde pas à la relation sémantique le même statut que l'entité ; il faut alors recourir à des artifices de modélisation, menant à la dissémination de la structure d'une relation à travers plusieurs entités (tables en relationnel).

Dans le cadre de l'approche objet des bases de données, une relation sémantique entre deux classes se traduit en général par un attribut dans chaque classe ; les deux attributs

Chapitre 2

Le modèle EMIR

Les remarques émises dans le paragraphe 1.4 du chapitre 1 identifient un certain nombre de caractéristiques nécessaires pour faire cohabiter, dans un même environnement, des données classiquement gérées dans les bases de données et des données issues des systèmes de recherche d'information dans les bases documentaires.

L'objectif du modèle EMIR [HM92, MLH94] est de contribuer au rapprochement entre les systèmes de recherche d'information et les bases de données. Ce modèle fournit les caractéristiques évoquées à la fin du chapitre 1, à savoir la flexibilité du schéma d'application, la distinction entre valeurs et objets, la typologie des liens de composition et la prise en compte de liens sémantiques entre les composants d'un objet.

Dans la suite, nous présentons en détail le modèle EMIR. Nous donnons d'abord des définitions informelles des concepts dans la partie 2.1, avant de présenter formellement le modèle, progressivement dans les parties 2.2, 2.3, 2.4 et 2.5.

2.1 Présentation informelle du modèle EMIR

Dans cette section, nous présentons informellement les concepts de base du modèle, avant de les formaliser dans les sections suivantes.

Notre modèle reposant, en particulier, sur la distinction entre valeurs et objets, nous définissons des *domaines* de valeurs et des *catégories* d'objets. Chaque valeur appartient à un domaine et chaque objet est relié à une catégorie.

Nous définissons également la notion de relation pour traduire une association possible entre des objets de deux catégories ou plus.

2.1.1 Domaines et catégories

Un domaine de valeurs est un ensemble de valeurs ayant une structure identique. Cette structure est appelée *type de valeurs* du domaine. Tout domaine a un nom qui le distingue des autres ; deux domaines de valeurs pouvant avoir le même type de valeurs (structure).

Une catégorie d'objets regroupe des objets partageant une structure commune. Cette structure est appelée *type d'objets* de la catégorie. Toute catégorie a un nom qui la distingue des autres ; deux catégories d'objets pouvant avoir le même type d'objets (structure).

A la différence des valeurs d'un domaine, les objets attachés à une catégorie n'ont pas tous la même structure. Seule une partie de leur structure est commune et chaque objet possède une structure individuelle propre supplémentaire ; ce qui représente la principale caractéristique du modèle. Cette caractéristique est fondamentale pour assurer d'une part l'accès groupé aux objets (à travers les catégories), d'autre part la flexibilité de structure des objets (à travers leurs structures individuelles).

Les types d'objets sont organisés en treillis, selon un ordre partiel basé sur leurs structures. En revanche les catégories sont organisées en une hiérarchie de généralisation /spé-

cialisation, définie par l'utilisateur (donc non calculée), mais cependant soumise à certaines conditions au niveau de la comparaison des types d'objets correspondants.

A ce propos, et dans toute la suite, nous désignerons par le terme "utilisateur" l'utilisateur final du modèle, c'est-à-dire la personne ou le programme qui manipulera des données selon le formalisme du modèle.

La structure d'une catégorie se divise en trois volets :

- une description, qui est une liste d'attributs, relié chacun à un domaine de valeurs,
- une composition, qui est une liste de composants, relié chacun à une catégorie d'objets,
- une topologie, qui est une liste de liens, représentant des relations sémantiques entre les composants.

La distinction entre description et composition d'une catégorie permet de différencier, parmi les caractéristiques d'un objet, celles qui sont de simples valeurs de celles qui sont des objets à part entière. La topologie permet de saisir des liens entre les composants des objets.

Dans la figure 2.1, nous présentons un exemple de catégorie EMIR : la catégorie *Pièce_rectangulaire*, issue d'une application architecturale du modèle, détaillée dans la partie 4.3.

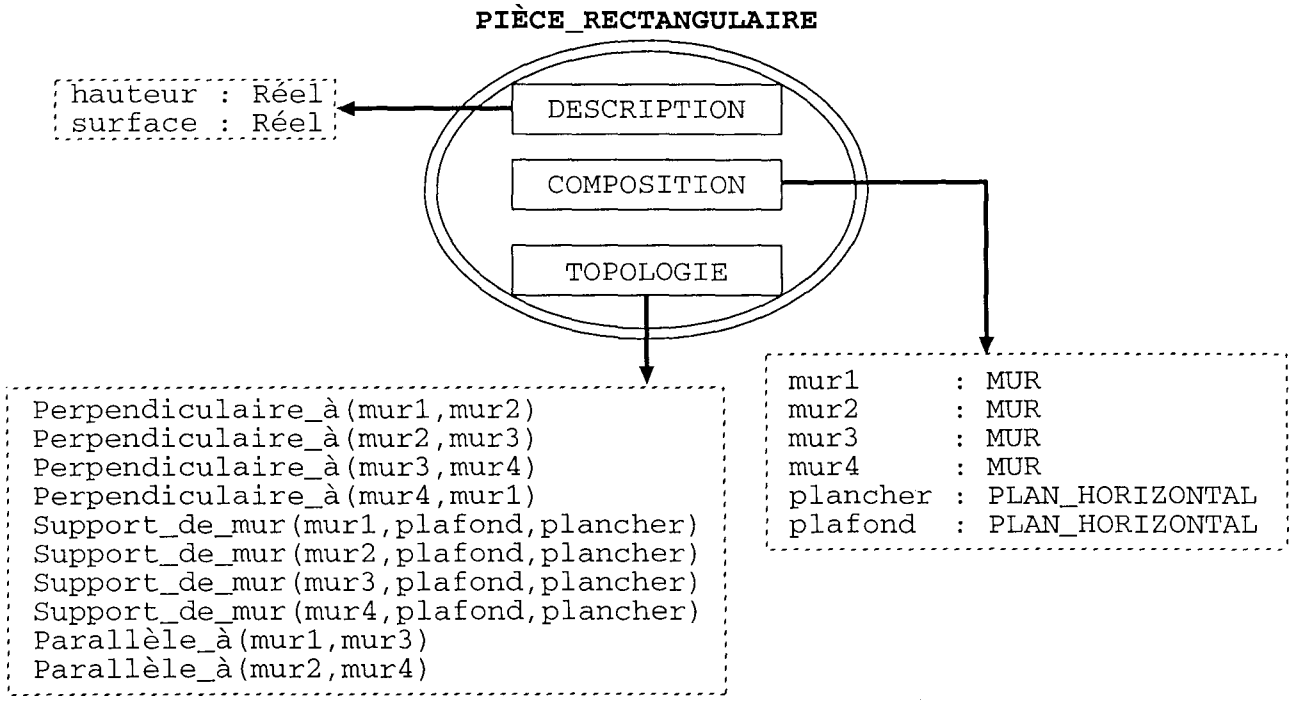


FIG. 2.1 – *Un exemple de catégorie EMIR*

2.1.2 Valeurs et objets

On suppose l'existence d'un ensemble de valeurs, de différentes structures, correspondant à des valuations possibles de caractéristiques d'objets du monde réel (couleur d'une voiture, âge d'une personne, ...).

Un objet est la représentation "en machine" d'une entité du monde réel. Il est désigné par un nom unique, qui le distingue des autres, indépendamment de ce que contient sa structure. Le type (structure) d'un objet est la manière dont sont structurés les valeurs, les objets ou les liens sémantiques, auxquels l'objet fait référence.

Tout objet est relié à une catégorie. Contrairement aux modèles de bases de données orientées objet classiques, le type d'un objet n'est pas exactement celui de sa catégorie (classe) ; il peut être plus spécifique que celui-ci, au sens de l'ordre partiel des types d'objets cité précédemment. Autrement dit, la structure d'un objet peut être plus riche que celle de la catégorie à laquelle il est relié. On dira que l'objet "réalise" (et non "instancie") une catégorie.

La figure 2.2 présente un exemple d'objet EMIR, réalisant la catégorie *Pièce_rectangulaire*. Les composants et liens représentés en gras sont propres à la pièce en question et ne se retrouvent donc pas dans la modélisation générale d'une pièce rectangulaire, donnée dans la figure 2.1.

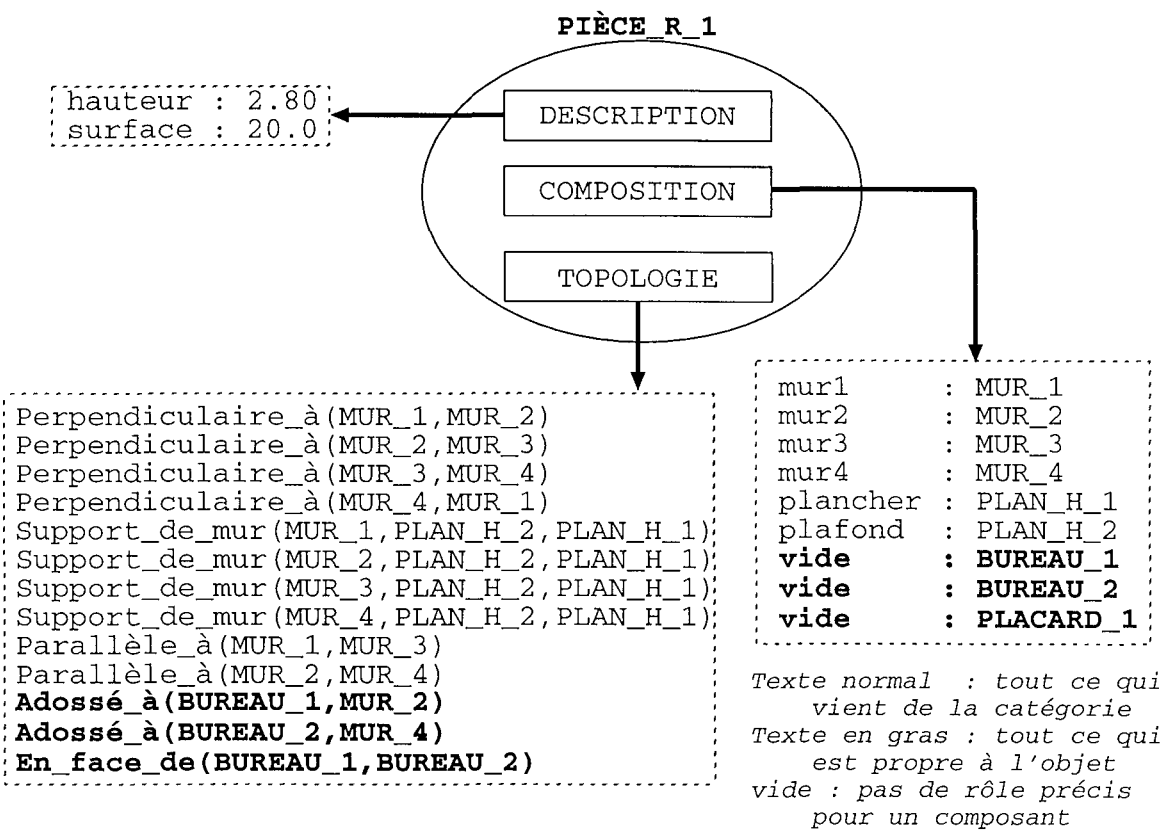


FIG. 2.2 – *Un exemple d'objet EMIR*

2.1.3 Relations

Une relation est une association définie par l'utilisateur entre deux ou plusieurs catégories (avec une sémantique qu'il choisit lui-même). Chaque catégorie mise en jeu par la relation y joue un rôle particulier. Les différents rôles constituent les *composants* de la relation. Comme les associations dans le modèle Entité / Association [CHE76, EWH85, TYF86], une relation peut avoir des attributs.

A la différence de ce modèle, les relations ne traduisent pas des propriétés des catégories, dans notre cas. Elles sont définies entre deux ou plusieurs catégories, mais ont un statut indépendant. Elles peuvent donc être utilisées à loisir pour relier des composants d'objets quelconques, à condition que ces composants soient reliés aux catégories mises en jeu par la relation. Ceci est particulièrement utile pour représenter des relations qui n'ont lieu que dans le cadre de la composition d'un objet particulier (localisation des composants pour les images). C'est d'ailleurs la seule utilisation possible des relations dans notre modèle : pour relier les composants d'un objet.

Pour ce qui est des relations permanentes entre catégories (entre une voiture et son propriétaire, par exemple), nous pensons comme dans [CAT94] qu'elles doivent être représentées par des composants de catégories.

Les relations sont aussi organisées en hiérarchie de généralisation / spécialisation, traduisant un affinement de la structure (attributs et composants).

Lorsqu'une relation est effective au niveau des objets, elle donne lieu à ce que nous appellons une relation concrète.

La figure 2.3 présente des exemples de relations (binaire et ternaire) et de relations concrètes associées.

2.2 Formalisation des concepts de base

Dans cette section, nous donnons une description formelle des concepts de base du modèle EMIR. Dans un but de lisibilité, nous laissons de côté, pour l'instant, l'aspect "topologie" des objets EMIR, pour nous concentrer sur les aspects "description" et "composition". La formalisation sera complétée par les relations sémantiques dans 2.3.

Le modèle EMIR se base sur une distinction entre la notion d'objet et la notion de valeur. Une valeur est une unité informationnelle qui fournit un renseignement sur une caractéristique (un attribut) d'un objet du monde réel (la surface d'une pièce, l'âge d'une personne, ...). Elle n'a pas d'existence propre. Un objet, par contre, est la représentation en machine d'un objet du monde réel. Il a donc une existence propre, indépendante de ce qui l'entoure, et des valeurs de ses caractéristiques.

Les valeurs sont prises dans des domaines, qui sont simples, multivalués ou agrégés (composés d'autres domaines). Chaque domaine est associé à un type de valeurs. La

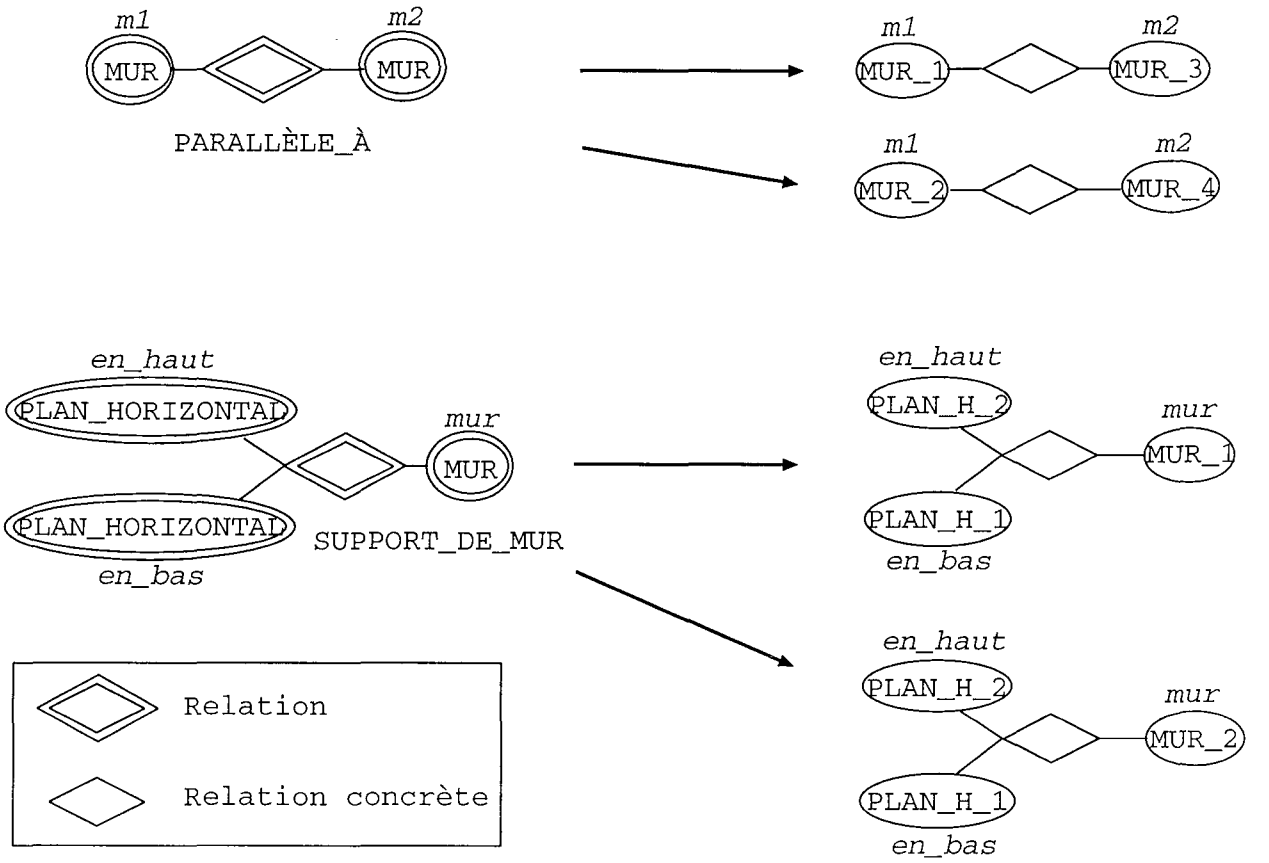


FIG. 2.3 – *Des exemples de relations EMIR*

différence entre un domaine et un type de valeurs réside dans le fait que le domaine porte un nom. Un type de valeurs représente uniquement l'aspect structurel des valeurs qui lui sont associées, alors qu'un domaine, de par son nom, renvoie en outre à une dimension sémantique.

2.2.1 Types de valeurs

2.2.1.1 Définition

Un type de valeurs $tv \in \mathbf{Tv}$ (ensemble de tous les types de valeurs) est une structure, associée à un ensemble de valeurs possibles.

On suppose l'existence des types de valeurs de base : B, I, R, S, M, définis par :

$B = \{V, F\}$, valeurs Booléennes,

I = Ensemble d'entiers naturels (partie de $\mathbb{N}$),

R = Ensemble de nombres réels (partie de $\mathbb{R}$),

S = Ensemble des chaînes obtenues à partir d'un jeu de caractères donné,

M = Ensemble des suites d'octets interprétables par un outil "multimédia" (gestionnaire d'images, de sons, de séquences vidéo, ...).

Les autres types de valeurs sont définis récursivement à partir de ces types de base, de la manière suivante :

- si tv est un type de valeurs, alors : $tv^* = \mathcal{P}(tv)$ (ensemble des parties de tv) est un type de valeurs, appelé *type ensemble*,

- si $tv_1, ..., tv_n, (n \geq 2)$ sont des types de valeurs, alors $tv_1 \times ... \times tv_n$ est un type de valeurs, appelé *type tuple*.

Exemples :
$B \times I$
$B \times R^* \times S$
$(B \times R) \times I = B \times R \times I$
$R \times (S^* \times M) = R \times S^* \times M$
$(R \times S)^* \times I$

2.2.1.2 Ordre partiel sur les types de valeurs

On définit récursivement un ordre partiel $\leq_{tv}$ sur les types de valeurs comme suit :

- $\forall tv \in \mathbf{Tv}, tv \leq_{tv} tv$

- $\forall tv_1, tv_2 \in \mathbf{Tv}$, si $tv_1 \leq_{tv} tv_2$ alors $tv_1^* \leq_{tv} tv_2^*$

- $\forall tv_1, ..., tv_n \in \mathbf{Tv}$, si $\exists i \in [1, n], tv_i \leq_{tv} tv$ alors $tv_1 \times ... \times tv_n \leq_{tv} tv$

- $\forall tv_1, ..., tv_m, tv'_1, ..., tv'_n \in \mathbf{Tv}$, si $n \leq m$ et $\forall i \in [1, n], tv_i \leq_{tv} tv'_i$ alors $tv_1 \times ... \times tv_m \leq_{tv} tv'_1 \times ... \times tv'_n$

Exemple :
$R \times S^* \times M \leq_{tv} R \times M$.
En effet, $R \times S^* \times M = R \times (S^* \times M)$,
et l'on a : $R \leq_{tv} R$ et $S^* \times M \leq_{tv} M$.

Une interprétation intuitive de cet ordre partiel est qu'un type de valeurs est "inférieur" à un autre si sa structure est au moins aussi riche.

Cet ordre partiel servira notamment à définir un ordre partiel sur les types d'objets, et à définir la notion d'héritage entre les catégories, du point de vue d'EMIR.

2.2.2 Valeurs

Soit $\mathbf{V}$ l'ensemble de toutes les valeurs. A chaque élément v de $\mathbf{V}$, on associe un type de valeurs $[\![v]\!]$. Les éléments de $\mathbf{V}$ sont définis de la manière suivante :

- les éléments de B, I, R, S et M sont des valeurs, dont les types sont ces ensembles respectifs,
- si v_1, ..., v_n sont des valeurs ayant un même type tv, alors $\{v_1, ..., v_n\}$ est une valeur dont le type est tv^*,
- si v_1, ..., v_n sont des valeurs et si a_1, ..., a_n sont des noms d'attributs (chaînes de caractères) alors $[a_1 : v_1, ..., a_n : v_n]$ est une valeur et l'on a : $[\![[a_1 : v_1, ..., a_n : v_n]]\!] = [\![v_1]\!] \times ... \times [\![v_n]\!]$

Exemple :
$[cours : \{[heure : 11, intitulé : "thermo"], [heure : 14, intitulé : "info"]\}, coef : 5]$ est une valeur dont le type est $(I \times S)^* \times I$.

2.2.3 Domaines de valeurs

2.2.3.1 Définition

- Un *domaine simple* $d : tv$ est, en général, obtenu en associant un nom d à un type de valeurs de base tv (B, I, R, S ou M).

  Exemples :
  $Effectif : I$,
  $String : S$,
  $Integer : I$, ...

  A chaque domaine d est associée une valeur par défaut, notée $defaut(d)$. Pour les domaines simples définis sur le type de valeur B (respectivement I, R, S ou M), la valeur par défaut est V (respectivement 0, 0.0, la chaîne vide ou une suite vide d'octets).

  Des domaines simples particuliers (continus ou discrets) sont construits à partir des types de valeurs de base I, R, S, seulement.

  Un domaine simple *continu* correspond à un intervalle de valeurs. Il est caractérisé par des bornes inférieure et supérieure différentes, ainsi que par le fait que les bornes sont incluses ou pas. L'ordre utilisé est l'ordre numérique pour I, R, et un ordre alphanumérique (du type de celui utilisé pour les dictionnaires électroniques) pour S.

Exemples :
$Angle : R/[0.0, 360.0[$ (R restreint à cet intervalle),
$Age : I/[1, 140]$, ...

La valeur par défaut d'un domaine continu est la moyenne des bornes.

Un domaine simple *discret* est aussi construit à partir d'un des types de base I, R ou S, et correspond à un ensemble fini non vide, de valeurs fixées à l'avance, du type utilisé.

Exemple :
$Couleur : S/\{"rouge", "vert", "jaune", "bleu", "noir"\}$, ...

La valeur par défaut d'un domaine discret est une valeur choisie parmi l'ensemble de ses valeurs.

- Un *domaine multivalué* $d' : \{d\}$ est obtenu, à partir d'un domaine d, en prenant des valeurs constituées d'ensembles de valeurs de d.

Exemple :
$Adresses : \{Adresse\}$, ...
$Strings : \{String\}$, ...

La valeur par défaut d'un domaine multivalué est un ensemble contenant la valeur par défaut du domaine sur lequel il est défini : $defaut(d' : \{d\}) = \{defaut(d)\}$.

- Un *domaine agrégé* $d : [a_1 : d_1, ..., a_n : d_n]$, $n \geq 2$ est construit par agrégation de plusieurs domaines $d_1, ..., d_n$, en associant à chacun un rôle a_i.

Exemple :
$Adresse : [numéro : Integer, rue : String, localité : Localité]$
avec $Localité : [code : Code\_postal, ville : String]$
et $Code\_postal : I/[01000, 99999]$.

La valeur par défaut d'un domaine agrégé est formée des valeurs par défaut de ses sous-domaines : $defaut(d : [a_1 : d_1, ..., a_n : d_n]) = [a_1 : defaut(d_1), ..., a_n : defaut(d_n)]$.

Afin de pouvoir associer à chaque domaine un type de valeurs précis (cf. 2.2.3.2), la définition récursive des domaines ne doit pas comporter de cycles. Autrement dit, un domaine d ne peut dépendre d'un autre domaine d' qui dépend lui-même de d. La notion de dépendance est assez intuitive et est définie par les règles suivantes :

- un domaine simple ne dépend d'aucun autre domaine,
- un domaine $d' : \{d\}$ dépend du domaine d,
- un domaine $d : [a_1 : d_1, ..., a_n : d_n]$ dépend de tous les domaines d_i,
- si d_1 dépend de d_2 et si d_2 dépend lui-même de d_3, alors d_1 dépend aussi de d_3 (propriété de transitivité).

Exemple : Le domaine $d_1 : \{d_2\}$ tel que $d_2 : [a_1 : d_3, a_2 : d_4]$ avec $d_4 : \{d_1\}$ est inconsistant car d_1 dépend de d_2 et vice-versa (à travers d_4).

Tous les domaines sont regroupés dans l'ensemble **D**.

2.2.3.2 Type d'un domaine

A chaque domaine d est associé un type de valeurs noté $[\![d]\!]$, défini récursivement de la manière suivante :

- Domaines simples :
  $[\![d : tv]\!] = [\![d : tv/[(])v_1, v_2](])]\!] = [\![d : tv/\{v_1, ..., v_n\}]\!] = tv$, où tv est l'un des types de base.
- Domaines multivalués :
  $[\![d' : \{d\}]\!] = [\![d]\!]^*$
- Domaines agrégés :
  $[\![d : [a_1 : d_1, ..., a_n : d_n]]\!] = [\![d_1]\!] \times ... \times [\![d_n]\!]$

Remarque : Cette définition est possible du fait de l'obligation faite précédemment, de ne pas avoir de cycle dans la définition des domaines.

Exemple :

$$
\begin{aligned}
[\![Adresses]\!] &= [\![Adresses : \{Adresse\}]\!] \\
&= [\![Adresse]\!]^* \\
&= [\![Adresse : [numéro : Integer, rue : String, localité : Localité]]\!]^* \\
&= ([\![Integer]\!] \times [\![String]\!] \times [\![Localité]\!])^* \\
&= ([\![Integer : I]\!] \times [\![String : S]\!] \times [\![Localité : [code : Code\_postal, ville : String]]\!])^* \\
&= (I \times S \times [\![Code\_postal]\!] \times [\![String]\!])^* \\
&= (I \times S \times [\![Code\_postal : I/[01000, 99999]]\!] \times [\![String : S]\!])^* \\
&= (I \times S \times I \times S)^*
\end{aligned}
$$

On associe à chaque domaine d un ensemble de valeurs possibles, noté $V(d)$. En général, l'ensemble des valeurs possibles pour un domaine est égal à son type, sauf pour les domaines discrets et continus, où les ensembles de valeurs associés sont respectivement l'ensemble discret et l'intervalle correspondants.

Exemples :
$V(Angle) = \mathbb{R}$ restreint à l'intervalle $[0.0, 360.0[$
$V(Age) = \mathbb{N}$ restreint à l'intervalle $[1, 140]$
$V(Couleur) = \{"rouge", "vert", "jaune", "bleu", "noir"\}$

2.2.3.3 Ordre partiel sur les domaines de valeurs

On pouvait dire simplement que l'ordre partiel sur les domaines de valeurs est déduit de l'ordre partiel sur les types de valeurs associés, mais les domaines discrets et continus posent certains problèmes.

On définit donc un ordre partiel $\leq_d$ sur les domaines comme suit :

- $d : X/[(])x_1, x_2](]) \leq_d d' : X$ (où X est I, R ou S)
- $d : X/\{x_1, ..., x_n\} \leq_d d' : X$ (où X est I, R ou S)
- si $[(])x_1, x_2](]) \subseteq [(])x'_1, x'_2](])$, alors $d : X/[(])x_1, x_2](]) \leq_d d' : X/[(])x'_1, x'_2](])$
- si $\{x_1, ..., x_n\} \subseteq \{x'_1, ..., x'_m\}$, alors $d : X/\{x_1, ..., x_n\} \leq_d d' : X/\{x'_1, ..., x'_m\}$
- si $d_1 \leq_d d_2$, alors $d'_1 : \{d_1\} \leq_d d'_2 : \{d_2\}$
- si $\exists i \in [1, n], d_i \leq_d d'$, alors $d : [a_1 : d_1, ..., a_n : d_n] \leq_d d'$
- si $n \leq m$ et $\forall i \in [1, n], d_i \leq_d d'_i$, alors $d : [a_1 : d_1, ..., a_m : d_m] \leq_d d' : [a'_1 : d'_1, ..., a'_n : d'_n]$

On peut voir aisément qu'un domaine est en général "inférieur" à un autre si les types de valeurs associés respectifs le sont ; à moins que les règles ci-dessus concernant les domaines discrets et continus soient utilisables ; auquel cas, elles prévalent.

Exemples :
$Adresse \leq_d Integer$, $Age \leq_d Integer$, ...
Si $Adresse\_restreinte : [numéro : Integer, rue : String, ville : String]$,
on a : $Adresse \leq_d Adresse\_restreinte$.

2.2.4 Types d'objets

2.2.4.1 Définition

Un type d'objets *simple to* est défini par la donnée de deux listes $A(to)$ et $C(to)$, éventuellement vides. On note $to = (A, C)$.

La première liste (A) est une liste d'attributs, correspondant chacun à un type de valeurs. La seconde liste (C) est une liste de composants, correspondant chacun à un type d'objets.

$A = \langle tv_1, ..., tv_n \rangle$ et $C = \langle to_1, ..., to_m \rangle$

où $tv_i \in \mathbf{Tv}$ et $to_j \in \mathbf{To}$, pour $1 \leq i \leq n$ et $1 \leq j \leq m$. **To** étant l'ensemble de tous les types d'objets.

Si $C(to)$ est vide, to est dit *type d'objets terminal.* Si en plus, $A(to)$ est vide, to est le *type d'objets vide*, noté $\top$.

D'autre part, si to est un type d'objets simple, alors to^* est aussi un type d'objets, appelé type d'objet *ensemble.*

2.2.4.2 Ordre partiel sur les types d'objets

- $\forall to \in \mathbf{To}, to \leq_{to} \top$

- si $to = (\langle tv_1, ..., tv_n \rangle, \langle to_1, ..., to_m \rangle)$ et $to' = (\langle tv'_1, ..., tv'_{n'} \rangle, \langle to'_1, ..., to'_{m'} \rangle)$, alors $to \leq_{to} to'$ si et seulement si :

  - $n' \leq n$ et $m' \leq m$
  - si $n' \geq 1$, alors il existe une injection σ_v de $[1, n']$ dans $[1, n]$ telle que $\forall i \in [1, n'], tv_{\sigma_v(i)} \leq_{tv} tv'_i$
  - si $m' \geq 1$, alors il existe une injection σ_o de $[1, m']$ dans $[1, m]$ telle que $\forall j \in [1, m'], to_{\sigma_o(j)} \leq_{to} to'_j$

- $to^* \leq_{to} to'^*$ si et seulement si $to \leq_{to} to'$

Exemples :
Soient les types d'objets $to = (\langle S^* \times I, R, M \rangle, \langle to_1, to_2 \rangle)$ et $to' = (\langle M, I \rangle, \langle to'_1 (= to_2) \rangle)$.
Ici, on est dans le deuxième cas ci-dessus, et l'on a $n = 3$, $m = 2$, $n' = 2$ et $m' = 1$.
Soient les injections σ_v et σ_o, définies par :

$$\sigma_v : \begin{array}{ccc} [1,2] & \longrightarrow & [1,3] \\ 1 & \longmapsto & 3 \\ 2 & \longmapsto & 1 \end{array} \quad \text{et} \quad \sigma_o : \begin{array}{ccc} [1,1] & \longrightarrow & [1,2] \\ 1 & \longmapsto & 2 \end{array}$$

On a :
$tv_{\sigma_v(1)} = tv_3 = M \leq_{tv} M = tv'_1$,
$tv_{\sigma_v(2)} = tv_1 = S^* \times I \leq_{tv} I = tv'_2$,
$to_{\sigma_o(1)} = to_2 \leq_{to} to_2 = to'_1$.

D'où $to \leq_{to} to'$.

2.2.5 Catégories

2.2.5.1 Définition

Une catégorie représente une ensemble d'objets. Elle possède un nom et elle est reliée à un type d'objets. Les objets qui seront reliés à cette catégorie devront avoir un type égal à son type d'objets, ou plus spécifique (inférieur au sens de l'ordre partiel sur les types d'objets).

Une *catégorie simple* $c : (A, C)$ est définie par la donnée d'un nom c et de deux listes $A = \langle a_1 : d_1, ..., a_n : d_n \rangle$ et $C = \langle r_1 : c_1, ..., r_m : c_m \rangle$, éventuellement vides, et notées respectivement $A(c)$ et $C(c)$.

Les a_i sont des noms d'attributs et les d_i des domaines de valeurs. Les r_j sont des rôles de composants et les c_j des catégories.

Si dans une catégorie simple c, $C(c)$ est vide, c est dite *catégorie terminale.* Si de plus, $A(c)$ est vide, c est dite *catégorie vide.*

Si $c : (A, C)$ est une catégorie simple, alors c^* est aussi une catégorie, dite *catégorie ensemble.* Elle sera reliée à des ensembles d'objets reliés à c.

Toutes les catégories sont regroupées dans l'ensemble **C**.

Exemples :

$Personne : (\langle nom : String, prénom : String, âge : Age \rangle, \langle\rangle)$
$Image : (\langle titre : String, date : Date, lieu : String, caractéristiques : Strings \rangle,$
$\langle photographe : Personne \rangle)$

avec $Date : [jour : Jour, mois : Mois, année : Integer]$, domaine agrégé,
et $Jour : I/[1, 31]$, $Mois : I/[1, 12]$, domaines continus.

2.2.5.2 Type d'une catégorie

A chaque catégorie c est associé un type d'objet noté $[\![c]\!]$, défini récursivement de la manière suivante :

- Catégories simples :
  Si $A(c) = \langle a_1 : d_1, ..., a_n : d_n \rangle$ et $C(c) = \langle r_1 : c_1, ..., r_m : c_m \rangle$,
  alors : $[\![c]\!] = (\langle [\![d_1]\!], ..., [\![d_n]\!] \rangle, \langle [\![c_1]\!], ..., [\![c_m]\!] \rangle)$
- Catégories ensembles :
  $[\![c^*]\!] = [\![c]\!]^*$

Exemple :

Si $Employé : (\langle insee : String, nom : String, prénom : String, âgeEmp : AgeEmp,$
$salaire : Salaire, adresses : Adresses \rangle, \langle\rangle)$, avec :
$AgeEmp : I/[18, 60]$ et $Salaire : R/]6000, 30000]$, alors :

$$[\![Employé]\!] = (\langle [\![String]\!], [\![String]\!], [\![String]\!], [\![AgeEmp]\!], [\![Salaire]\!], [\![Adresses]\!] \rangle, \langle\rangle)$$
$$= (\langle S, S, S, I, R, (I \times S \times I \times S)^* \rangle, \langle\rangle)$$

2.2.5.3 Lien d'héritage entre les catégories

Le lien d'héritage entre catégories est un ordre partiel, noté $\leq_c$, et tel que :

- il est défini *par l'utilisateur* sur les catégories simples, avec la contrainte que si $c_1 \leq_c c_2$, avec :
  - $A(c_1) = \langle a_{11} : d_{11}, ..., a_{1n_1} : d_{1n_1} \rangle$ et $C(c_1) = \langle r_{11} : c_{11}, ..., r_{1m_1} : c_{1m_1} \rangle$,
  - $A(c_2) = \langle a_{21} : d_{21}, ..., a_{2n_2} : d_{2n_2} \rangle$ et $C(c_2) = \langle r_{21} : c_{21}, ..., r_{2m_2} : c_{2m_2} \rangle$,

  alors :
  - $n_2 \leq n_1$ et $m_2 \leq m_1$,
  - si $n_2 \geq 1$, il existe une injection σ_a de $[1, n_2]$ vers $[1, n_1]$, appelée "renommage d'attributs", telle que $\forall i \in [1, n_2], d_{1\sigma_a(i)} \leq_d d_{2i}$,
  - si $m_2 \geq 1$, il existe une injection σ_c de $[1, m_2]$ vers $[1, m_1]$, appelée "renommage de composants", telle que $\forall j \in [1, m_2], c_{1\sigma_c(j)} \leq_c c_{2j}$,
- $c_1^* \leq_c c_2^*$ si et seulement si $c_1 \leq_c c_2$,
- si $c_1 \leq_c c_2$ et $c_1 \leq_c c_3$, alors on a soit $c_2 \leq_c c_3$, soit $c_3 \leq_c c_2$; autrement dit, l'héritage multiple n'est pas autorisé (le graphe d'héritage est un arbre).

Exemples :
$Employé \leq_c Personne$ est correct.
En effet, on est dans le premier cas ci-dessus, et l'on a $n_1 = 6$, $m_1 = 3$, $n_2 = 0$ et $m_2 = 0$.
Avec l'injection σ_a, définie par :

$$\begin{array}{rccc} \sigma_a : & [1,3] & \longrightarrow & [1,6] \\ & 1 & \longmapsto & 2 \\ & 2 & \longmapsto & 3 \\ & 3 & \longmapsto & 4 \end{array}$$

on a :
$d_{1\sigma_a(1)} = d_{12} = String \leq_d String = d_{21}$,
$d_{1\sigma_a(2)} = d_{13} = String \leq_d String = d_{22}$,
$d_{1\sigma_a(3)} = d_{14} = AgeEmp \leq_d Age = d_{23}$.

2.2.6 Objets

2.2.6.1 Définition

Un *objet simple* $o = (i, s)$ est défini par la donnée d'un identifiant unique et exclusif $i(o) \in \mathbf{I}$ (où $\mathbf{I}$ est un ensemble d'identifiants d'objets) et d'une structure $s(o)$, formée de deux listes $A(o) = \langle a_1 : v_1, ..., a_n : v_n \rangle$ et $C(o) = \langle r_1 : o_1, ..., r_m : o_m \rangle$.

Les a_i sont des noms d'attributs, les v_i des valeurs, les r_j des rôles de composants, et les o_j des objets composants.

Afin d'autoriser les objets à possèder des composants individuels, qui ne jouent pas de rôle particulier dans leur composition, on notera *vide* dans r_j à chaque fois qu'aucune valeur sémantique ne peut être attribuée au rôle que joue le composant o_j dans la composition de l'objet principal o (par exemple, lorsqu'une image est composée d'une voiture ou d'une personne). Ces composants seront appelés *composants d'observation.*

Un *objet ensemble* $o = (i, s)$ est tel que $s(o) = \{o_1, ..., o_p\}$, où les o_i sont des objets simples.

Exemples :
Nous donnons ci-après des exemples de structures d'objets. L'objet o_1 représente une image ayant comme attributs un titre, une date et un lieu de prise, ainsi que des caractéristiques visuelles. Cet objet fait référence à un autre objet (o_2) qui représente l'auteur de l'image, qui est une personne.

Toutes les caractéristiques de o_1 énoncées jusqu'ici se retrouvent dans toutes les images. L'objet o_1 fait de plus référence à l'objet o_3 qui représente une personne qu'on voit sur l'image en question et à un objet o_4 qui représente un employé, également visible sur l'image que représente o_1. Ces deux derniers objets ne jouent pas de rôle précis dans o_1 comme c'était le cas pour l'objet o_2 (qui représente l'*auteur* de o_1). Ils sont donc précédés d'un rôle noté *vide* dans la liste des objets qui composent o_1.

$s(o_1) = (\langle titre : "Deux\ hommes\ devant\ le\ pont\ Marie", date : [jour : 12, mois : 3, année : 1900],$
$lieu : "Paris", caractéristiques : \{"paysage", "Noir\&Blanc"\}\rangle,$
$\langle photographe : o_2, vide : o_3, vide : o_4\rangle)$
$s(o_2) = (\langle nom : "Albert", prénom : "Eugène", âge : 40\rangle, \langle\rangle)$
$s(o_3) = (\langle nom : "Meunier", prénom : "Jean", âge : 38\rangle, \langle\rangle)$
$s(o_4) = (\langle insee : "168087501566973", nom : "Duchemin", prénom : "Emile",$
$âge : 45, salaire : 9045.32,$
$adresses : \{[numéro : 4, rue : "bd\ des\ fleurs", localité : [code : 75000, ville : "Paris"]]$
$\langle\rangle)$

Remarque : Si dans un objet simple o, $C(o)$ est vide, o est dit *objet terminal.* Si de plus, $A(o)$ est vide, o est dit *objet vide.*

L'ensemble de tous les objets est noté **O**.

2.2.6.2 Type d'un objet

A chaque objet o est associé un type d'objet noté $[\![o]\!]$, défini récursivement de la manière suivante :

- Objets simples :
  Si $A(o) = \langle a_1 : v_1, ..., a_n : v_n\rangle$ et $C(o) = \langle r_1 : o_1, ..., r_m : o_m\rangle$, alors :

$[\![o]\!] = (\langle [\![v_1]\!], ..., [\![v_n]\!] \rangle, \langle [\![o_1]\!], ..., [\![o_m]\!] \rangle)$

- Objets ensembles :
  Si $s(o) = \{o_1, ..., o_p\}$, alors :
  $$[\![o]\!] = (\sup_{i=1}^{p} [\![o_i]\!])^*$$
  où *sup* désigne la borne supérieure au sens de l'ordre partiel sur les types d'objets (type le plus générique).

Remarquons que la définition du type d'un objet ensemble est validée par le fait que tout ensemble de types d'objets admet une borne supérieure (au pire le type vide).

Exemples :
$[\![o_1]\!] = (\langle S, I \times I \times I, S, S^* \rangle, \langle [\![o_2]\!], [\![o_3]\!], [\![o_4]\!] \rangle)$
$[\![o_2]\!] = (\langle S, S, I \rangle, \langle \rangle)$
$[\![o_3]\!] = (\langle S, S, I \rangle, \langle \rangle)$
$[\![o_4]\!] = (\langle S, S, S, I, R, (I \times S \times I \times S)^* \rangle, \langle \rangle)$

2.2.6.3 Lien de réalisation

Le noyau du lien de réalisation [LM96] est une application définie *par l'utilisateur* de l'ensemble des *objets simples* vers l'ensemble des *catégories simples*, notée $\leftarrow$ et qui vérifie la contrainte que si $o \leftarrow c$ (dit "*o réalise c*"), avec $A(c) = \langle a_1 : d_1, ..., a_n : d_n \rangle$ et $C(c) = \langle r_1 : c_1, ..., r_m : c_m \rangle$, alors :

- $A(o) = \langle a_1 : v_1, ..., a_{n'} : v_{n'} \rangle$ et $C(o) = \langle r_1 : o_1, ..., r_{m'} : o_{m'} \rangle$, avec $n' \geq n$ et $m' \geq m$,
- $\forall i \in [1, n], v_i \in V(d_i)$,
- $\forall j \in [1, m], o_j \leftarrow c_j$,

Le lien de réalisation est alors l'extension de ce noyau défini par l'utilisateur, à une relation binaire définie sur l'ensemble $\mathbf{O} \times \mathbf{C}$, par les règles :

- si o est un objet ensemble tel que $s(o) = \{o_1, ..., o_p\}$, alors $o \leftarrow c^*$ si et seulement si $\forall i \in [1, p], o_i \leftarrow c$.
- si $o \leftarrow c$ et si $c \leq_c c'$, alors $o \leftarrow c'$.

Exemples : (reprenant les objets précédemment décrits)
$o_1 \leftarrow Image$
$o_2 \leftarrow Personne$
$o_3 \leftarrow Personne$
$o_4 \leftarrow Employé$
$o_4 \leftarrow Personne$

Avec cette définition, nous avons le résultat fondamental suivant, qui stipule qu'un objet réalisant une catégorie possède une structure plus riche que celle-ci :

$$\text{Si } o \leftarrow c, \text{ alors } [\![o]\!] \leq_{to} [\![c]\!]$$

Preuve :

Soient donc $o \in \mathbf{O}$ et $c \in \mathbf{C}$, tels que $o \leftarrow c$. Il s'agit de montrer que $[\![o]\!] \leq_{to} [\![c]\!]$.

- Si o est un objet simple, alors c est une catégorie simple et l'on a (aux renommages près) :
  - $A(o) = \langle a_1 : v_1, ..., a_{n'} : v_{n'} \rangle$, $C(o) = \langle r_1 : o_1, ..., r_{m'} : o_{m'} \rangle$
  - $A(c) = \langle a_1 : d_1, ..., a_n : d_n \rangle$, $C(c) = \langle r_1 : c_1, ..., r_m : c_m \rangle$

  avec :
  - $n' \geq n$ et $m' \geq m$,
  - $\forall i \in [1, n], v_i \in V(d'_i)$, avec $d'_i \leq_d d_i$,
  - $\forall j \in [1, m], o_j \leftarrow c_j$.

  La démonstration se fera par induction sur les objets composant $C(o)$.

  On a déjà $\forall i \in [1, n], [\![v_i]\!] \leq_{tv} [\![d_i]\!]$.

  On suppose, comme hypothèse d'induction, que $\forall j \in [1, m], [\![o_j]\!] \leq_{to} [\![c_j]\!]$.

  $[\![o]\!] = (\langle [\![v_1]\!], ..., [\![v_{n'}]\!] \rangle, \langle [\![o_1]\!], ..., [\![o_{m'}]\!] \rangle)$.

  D'après l'hypothèse d'induction et la définition de l'ordre partiel sur les types d'objets simples, on a : $[\![o]\!] \leq_{to} (\langle [\![d_1]\!], ..., [\![d_n]\!] \rangle, \langle [\![c_1]\!], ..., [\![c_m]\!] \rangle) = [\![c]\!]$.

- Si o est un objet ensemble, alors $s(o) = \{o_1, ..., o_p\}$ et $c = c'^*$. Or, d'après la définition du lien de réalisation, $\forall i \in [1, p], o_i \leftarrow c'$.

  La démonstration se fera aussi par induction sur les objets appartenant à $s(o)$. Si on suppose, comme hypothèse d'induction, que $\forall i \in [1, p], [\![o_i]\!] \leq_{to} [\![c']\!]$, alors $[\![c']\!]$ est un majorant des $[\![o_i]\!]$, et donc :

$$\sup_{i=1}^{p} [\![o_i]\!] \leq_{to} [\![c']\!]$$

  d'où :

$$[\![o]\!] = (\sup_{i=1}^{p} [\![o_i]\!])^* \leq_{to} ([\![c']\!])^* = [\![c'^*]\!] = [\![c]\!]$$

□

Remarque :
Avec des définitions analogues dans un modèle de BDOO classique, les deux types d'objets sont identiques ($[\![o]\!] = [\![c]\!]$).

Avantages du lien de réalisation

- Le lien de réalisation ne contraint pas un objet réalisant une catégorie à une structure fixe (imposée par la catégorie), comme le ferait le classique lien d'instanciation entre un objet et une classe dans un SGBDOO.

- Pour modéliser certaines applications (notamment en recherche d'information basée sur le contenu, ou encore en architecture), le concepteur ne peut donner la structure exacte de toutes les catégories de l'application, mais uniquement une partie de cette structure. Cette partie représente, en général, la structure minimale stable et standard. L'avantage du lien de réalisation est alors de permettre au concepteur de créer ses catégories avec des structures minimales, pour ensuite générer des objets dont les structures sont plus riches.

- Lorsque l'application comporte des objets exceptionnels, qui ont des propriétés supplémentaires par rapport à l'objet commun de la catégorie, le lien de réalisation permet de ne pas avoir recours à la création de sous-catégories (catégories spécifiques) à chaque fois qu'une nouvelle exception se présente.

- Le lien de réalisation peut avoir un intérêt d'un point de vue "méthode de conception". En effet, pour des applications où les objets changent de structure dans le temps, d'une manière non maîtrisée à l'avance (cycle de vie non stable), le lien de réalisation permet aux objets d'évoluer assez librement par rapport à leurs catégories (en ayant des structures individuelles). Ensuite, à partir d'une base stabilisée (où la structure des objets n'évolue plus), il permettrait de générer les structures des catégories en analysant l'état final des objets qui les réalisent, pour revenir à un modèle de classes habituel. Ce type de conception peut être qualifié de "conception incrémentale".

- Par rapport aux langages de prototypes, démunis de la notion de classe, le lien de réalisation conserve, à travers la notion de catégorie, une primitive d'accès groupé aux objets.

2.3 Le modèle EMIR avec topologie

Nous allons enrichir le modèle de données précédemment cité, de manière à ce qu'il permette d'exprimer facilement les liens qui existent entre les composants d'un objet. Le but est de pouvoir exprimer dans un même objet, d'une part ses caractéristiques et ses objets composants, et d'autre part, la manière dont les composants sont "agencés". Cet agencement des composants prendra la forme d'un ensemble de liens sémantiques définis par l'utilisateur, et liant chacun, deux ou plusieurs composants.

Un lien entre les composants d'un objet est la concrétisation d'une entité conceptuelle : une relation. Une relation définit une association, en intension, entre deux ou plusieurs catégories. Elle exprime un lien sémantique potentiel entre des objets réalisant ces catégories ; par exemple, une relation de proximité entre deux objets dans une image, une relation d'appartenance d'un véhicule à une personne, ...

Le fait de relier des objets d'une application est une primitive de modélisation indispensable, exprimée différemment selon les modèles de données.

Dans le modèle relationnel [DA82] où tous les objets sont représentés par des tables, une relation sémantique peut être représentée elle-même par une table contenant les identifiants des objets qu'elle relie. Si le lien est orienté, dans le sens où l'un des objets participant à la relation détermine exactement l'autre, la relation est représentée implicitement par l'ajout d'un identifiant de l'objet dépendant, dans la structure de l'objet principal. Ainsi, on ajoutera un identifiant du propriétaire d'une voiture (son nom par exemple, s'il n'y a pas d'ambiguité), à la structure de celle-ci, puisque toute voiture admet un propriétaire et un seul. Dans certains cas, il est utile de représenter les liens inverses, en ajoutant un identifiant de chaque objet participant à la relation dans la structure des autres. Ainsi, dans une relation liant deux conjoints, on ajoutera à la structure de chaque objet protagoniste de la relation (de type Personne), un identifiant de son conjoint.

La manière de représenter les relations sémantiques dans le modèle relationnel n'est donc pas très naturelle, dans la mesure où le modèle ne fournit pas réellement le concept de relation sémantique comme primitive de modélisation. Les informations concernant un lien sémantique se retrouvent souvent disséminées à travers plusieurs tables.

Le modèle Entité / Association (E/A) [CHE76] confère à une relation sémantique (appelée association) la même importance qu'une entité du monde réel. Une association a un statut propre (et éventuellement des attributs), au même titre qu'un objet. La traduction d'un schéma E/A dans un modèle relationnel conduit d'ailleurs à la traduction de chaque association, en fonction de la nature de ses protagonistes, selon l'une ou l'autre des techniques discutées ci-dessus. Le modèle E/A permet donc à l'utilisateur une définition plus naturelle des liens entre objets. Ces liens peuvent éventuellement être traduits dans un modèle relationnel, qui est moins adapté à modéliser ce genre d'informations.

Dans un modèle à objets, les relations n'ont pas de statut propre, mais peuvent aussi être modélisées de diverses façons [RUM87]. La plus courante est de les représenter à travers deux attributs inverses, chacun dans une classe [CAT94].

Nous nous proposons de définir dans notre modèle le concept de relation sémantique comme primitive de modélisation, à la manière du modèle E/A. Cependant, le but est d'enrichir la sémantique de la composition des objets ; les relations ne seront utilisées que pour définir l'agencement des composants d'un même objet. Lorsqu'il s'agit de relations qui traduisent des propriétés des catégories reliées (voiture et son propriétaire par exemple), nous les modéliserons, comme c'est préconisé dans [CAT94], à travers des composants dans chacune des catégories mises en jeu.

Dans la suite de cette section, nous présentons les deux notions de relation et de relation concrète, respectivement dans 2.3.1 et 2.3.2. En particulier, dans 2.3.1.2, nous définissons un ordre partiel d'héritage sur les relations, au même titre que celui sur les catégories (cf 2.2.5.3). Enfin, dans 2.3.3, certaines définitions des concepts du modèle présentées dans 2.2, sont remaniées pour tenir compte des notions de relation et de relation concrète.

2.3.1 Relations

2.3.1.1 Définition

Dans EMIR, une relation sémantique $r : (C, A)$ est définie par la donnée d'un nom r et de deux listes $C = \langle ro_1 : c_1, ..., ro_n : c_n \rangle$ (ayant au moins deux éléments : $n \geq 2$) et $A = \langle a_1 : d_1, ..., a_m : d_m \rangle$ (éventuellement vide), notées respectivement $C(r)$ et $A(r)$.

Ces listes représentent respectivement les composants de la relation (i.e. les catégories qu'elle relie, avec leurs rôles) et ses attributs, avec leurs domaines. Les ro_i sont les rôles des composants, les c_i leurs catégories, les a_j des noms d'attributs et les d_j leurs domaines.

Remarque : Afin d'éviter toute confusion, on notera les relations par des r et les rôles par des ro.

Exemples :
La relation *A_gauche_de* est définie sur des bâtiments, par les listes :
$C(A\_gauche\_de) = \langle gauche : Bâtiment, droite : Bâtiment \rangle$ et
$A(A\_gauche\_de) = \langle \rangle$.
Les relations *Parallèle_à* et *Perpendiculaire_à* sont définies sur des murs, par les listes :
$C(Parallèle\_à) = \langle m1 : Mur, m2 : Mur \rangle$ et
$A(Parallèle\_à) = \langle distance : Réel \rangle$.
$C(Perpendiculaire\_à) = \langle m1 : Mur, m2 : Mur \rangle$ et
$A(Perpendiculaire\_à) = \langle \rangle$.

2.3.1.2 Lien d'héritage entre relations

Comme pour les catégories simples (cf 2.2.5.3), le lien d'héritage entre les relations est un ordre partiel, noté $\leq_r$, défini *par l'utilisateur* et respecte la contrainte que si $r_1 \leq_r r_2$, avec :

- $C(r_1) = \langle ro_{11} : c_{11}, ..., ro_{1m_1} : c_{1m_1} \rangle$ et $A(r_1) = \langle a_{11} : d_{11}, ..., a_{1n_1} : d_{1n_1} \rangle$,
- $C(r_2) = \langle ro_{21} : c_{21}, ..., ro_{2m_2} : c_{2m_2} \rangle$ et $A(r_2) = \langle a_{21} : d_{21}, ..., a_{2n_2} : d_{2n_2} \rangle$,

alors :

- $n_2 \leq n_1$ et $m_2 \leq m_1$,

- il existe une injection σ_c de $[1, m_2]$ vers $[1, m_1]$, appelée "renommage de composants", telle que $\forall j \in [1, m_2], c_{1\sigma_c(j)} \leq_c c_{2j}$,
- il existe une injection σ_a de $[1, n_2]$ vers $[1, n_1]$, appelée "renommage d'attributs", telle que $\forall i \in [1, n_2], d_{1\sigma_a(i)} \leq_d d_{2i}$.

Exemple :
Soit la relation *A_gauche_de_avec_distance*, définie sur les pavillons d'un lotissement, par les listes :
$C(A\_gauche\_de\_avec\_distance) = \langle gauche : Pavillon, droite : Pavillon \rangle$ et
$A(A\_gauche\_de\_avec\_distance) = \langle distance : Réel \rangle$.
Si on suppose que $Pavillon \leq_c Bâtiment$, alors $A\_gauche\_de\_avec\_distance \leq_r A\_gauche\_de$ est correct.

2.3.2 Relations concrètes

Une relation concrète rc est la matérialisation d'une relation sémantique (r), au niveau concret (celui des objets). Elle relie deux ou plusieurs objets par la relation sémantique r, en valuant ses attributs.

Elle est représentée par les deux listes $C(rc) = \langle ro_1 : o_1, ..., ro_n : o_n \rangle$ et $A(rc) = \langle a_1 : v_1, ..., a_m : v_m \rangle$, où les ro_i sont des rôles de composants, les o_i les objets correspondants, les a_j des noms d'attributs et les v_j leurs valeurs.

Les ro_i et les a_j sont *exactement* les mêmes que ceux de r et l'on a : $\forall i \in [1, n], o_i \leftarrow c_i$ et $\forall j \in [1, m], v_j \in V(d_j)$.

On dit que rc instancie la relation r, et on note $rc : r$.

Exemple :
Soient deux objets o_1 et o_2, réalisant la catégorie *Pavillon*.
rc : *A_gauche_de_avec_distance* est définie par les listes :
$C(rc) = \langle gauche : o_1, droite : o_2 \rangle$ et
$A(rc) = \langle distance : 1500.00 \rangle$.

2.3.3 Redéfinition de quelques concepts du modèle

La structure des objets et des catégories est enrichie par des liens entre leurs composants. Le but est de saisir dans la structure d'un objet, d'une part les objets qui le composent, d'autre part des liens qui les unissent entre eux, afin de mieux représenter la manière dont il est composé. Ces liens peuvent mettre en jeu l'objet lui-même.

Par exemple, on enrichit la modélisation d'une pièce dans un bâtiment, en ajoutant à la liste des objets qui la composent (quatre murs, un plancher, un plafond, ...), le fait que le premier mur et le second se calent l'un l'autre, ainsi que le second et le troisième, ..., que les murs sont supportés par le plancher et le plafond, ...

Cet enrichissement se traduira au niveau des structures d'objets simples (et de catégories simples) par l'ajout d'une troisième liste (en plus des attributs et des composants), notée L et composée de liens entre les composants. Ces liens peuvent aussi mettre en jeu l'objet lui-même.

Désormais, on appellera les listes A, C et L respectivement, *description, composition* et *topologie.*

Nous exposons ci-après les modifications qu'occasionne la prise en compte de la topologie, sur certaines des définitions présentées dans la partie 2.2.

Les autres définitions (en particulier celles concernant les types d'objets) restent inchangées, puisque la notion de type d'objets est basée dans notre modèle sur la structure. La topologie n'intervient pas dans la définition d'un type d'objets ; elle est considérée comme un complément d'information sur la structure (liens entre composants), et non comme une partie de la structure.

2.3.3.1 Redéfinition des catégories

Soit c une catégorie simple, telle que $C(c) = \langle ro_1 : c_1, ..., ro_m : c_m \rangle$. La topologie de c est une liste $L(c) = \langle l_1 : r_1, ..., l_p : r_p \rangle$, formée de liens en intension : l_k. Chaque lien l_k fait référence à une relation sémantique : r_k et met en jeu des composants de c (issus de la liste $C(c)$).

Les l_k sont définis de la manière suivante : $l_k = \langle ro_{k1} : ro_{\sigma(1)}, ..., ro_{kn_k} : ro_{\sigma(n_k)} \rangle$, où σ est une injection de $[1, n_k]$ dans $[0, m]$.

Le lien l_k fait référence à la relation r_k, telle que $C(r_k) = \langle ro_{k1} : c'_1, ..., ro_{kn_k} : c'_{n_k} \rangle$; il lie chaque composant de r_k, désigné par son rôle ro_{kl}, à un composant de la catégorie c, désigné par son rôle $ro_{\sigma(l)}$.

Dans le cas où $\sigma(l)$ vaut 0, le composant ro_0 n'existe pas dans la liste $C(c)$. En fait, il est utilisé pour l'uniformité de la notation. En réalité, il sera noté *Self* et dénotera l'objet principal lui-même. Cela permet de traduire des liens sémantiques entre un objet et ses composants ; par exemple, une voiture est à l'arrière plan d'une image.

Pour que cette définition soit cohérente, il faut que la catégorie $c_{\sigma(l)}$, reliée au composant de rôle $ro_{\sigma(l)}$ dans la composition de la catégorie c, soit plus spécifique que la catégorie c'_l, reliée au composant de rôle ro_{kl} dans la composition de la relation r_k : $c_{\sigma(l)} \leq_c c'_l$, pour $1 \leq l \leq n_k$.

Exemple :
La figure 2.4 représente une catégorie *Pièce* dans un formalisme graphique défini pour EMIR, et qui est explicité en détail dans l'annexe A.

Les domaines et valeurs sont représentés par des rectangles, les catégories et objets par des ellipses et les différentes relations par des losanges.

On définit la catégorie *Pièce* : (A, C, L) par les listes :

$A(Pièce) = \langle hauteur : Réel, surface : Réel \rangle$
$C(Pièce) = \langle mur_1 : Mur, mur_2 : Mur, mur_3 : Mur, mur_4 : Mur,$
$plancher : Plan\_horizontal, plafond : Plan\_horizontal \rangle$
$L(Pièce) = \langle calage_1 : Calage, calage_2 : Calage,$
$calage_3 : Calage, calage_4 : Calage,$
$support_1 : Support\_de\_mur, support_2 : Support\_de\_mur,$
$support_3 : Support\_de\_mur, support_4 : Support\_de\_mur \rangle.$

La dernière liste traduit des liens entre les composants de la catégorie. Ces liens seront réalisés, au niveau des objets, par des relations concrètes entre leurs objets composants.

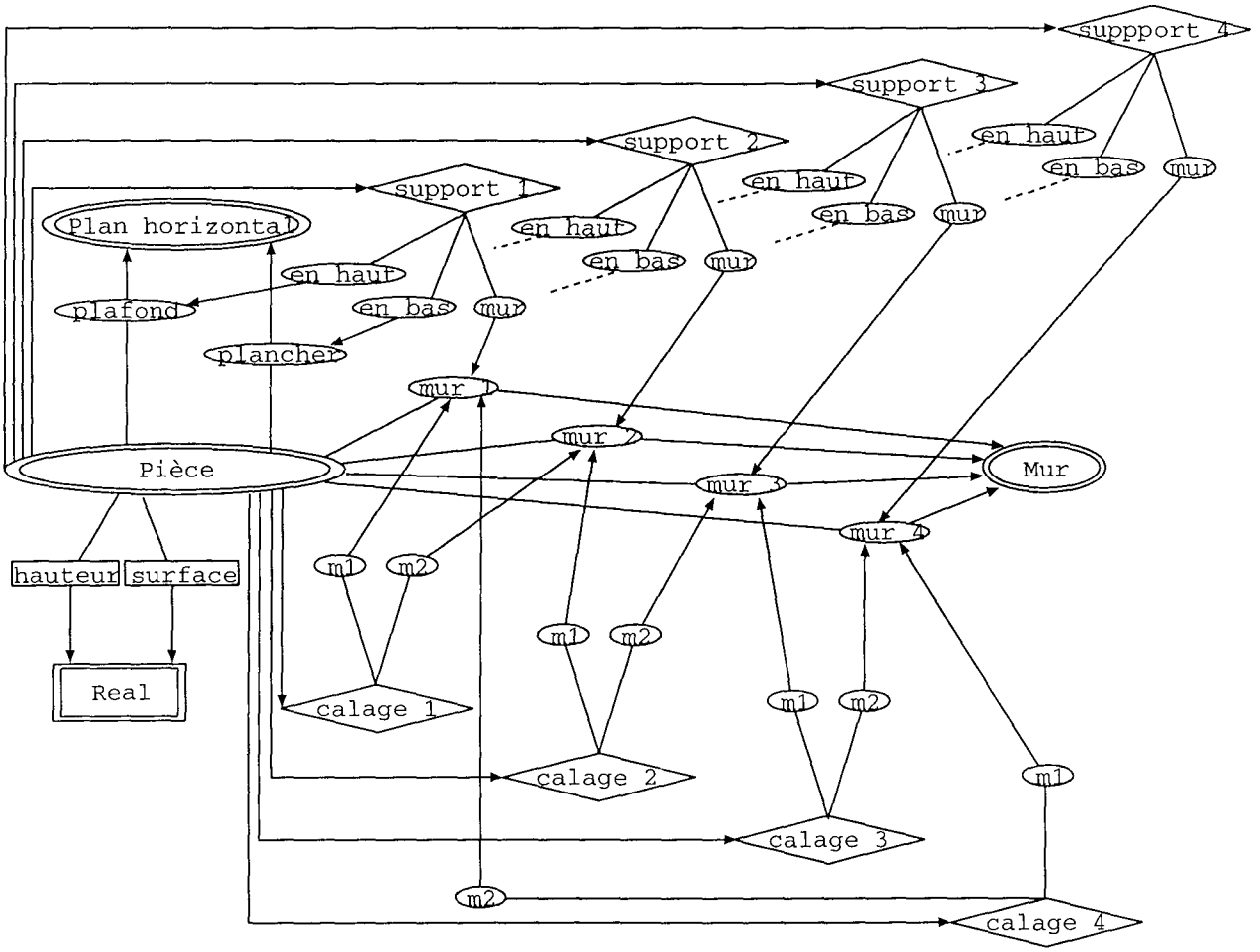


FIG. 2.4 – *La catégorie Pièce, issue d'une application du modèle en architecture*

Les relations *Calage* et *Support_de_mur* traduisent respectivement que deux murs se calent, et qu'un mur est supporté par un plancher et un plafond. Elles sont définies (cf. figure 2.5) par les listes :
$C(Calage) = \langle m_1 : Mur, m_2 : Mur \rangle$
$A(Calage) = \langle\rangle$.
$C(Support\_de\_mur) = \langle mur : Mur, en\_haut : Plan\_horizontal, en\_bas : Plan\_horizontal \rangle$
$A(Support\_de\_mur) = \langle\rangle$.

Ainsi, par exemple, les liens $calage_i$ et $support_j$ sont définis par :
$C(calage_1) = \langle m_1 : mur_1, m_2 : mur_2 \rangle$
$C(calage_2) = \langle m_1 : mur_2, m_2 : mur_3 \rangle$
$C(calage_3) = \langle m_1 : mur_3, m_2 : mur_4 \rangle$
$C(calage_4) = \langle m_1 : mur_4, m_2 : mur_1 \rangle$
$C(support_1) = \langle mur : mur_1, en\_haut : plafond, en\_bas : plancher \rangle$
$C(support_2) = \langle mur : mur_2, en\_haut : plafond, en\_bas : plancher \rangle$

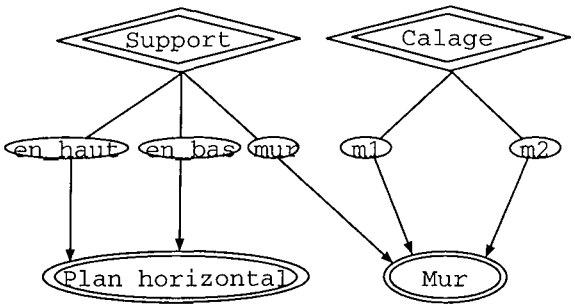


FIG. 2.5 – *Deux relations issues de l'application en architecture*

$C(support_3) = \langle mur : mur_3, en\_haut : plafond, en\_bas : plancher \rangle$
$C(support_4) = \langle mur : mur_4, en\_haut : plafond, en\_bas : plancher \rangle$.

2.3.3.2 Redéfinition du lien d'héritage entre catégories simples

La définition du lien d'héritage, telle qu'elle a été donnée en 2.2.5.3, entre deux catégories simples c_1 et c_2, avec $L(c_1) = \langle l_{11} : r_{11}, ..., l_{1p_1} : r_{1p_1} \rangle$ et $L(c_2) = \langle l_{21} : r_{21}, ..., l_{2p_2} : r_{2p_2} \rangle$, doit en plus imposer les contraintes :

- $p_2 \leq p_1$,
- il existe une injection σ_l de $[1, p_2]$ vers $[1, p_1]$, appelée "renommage de liens", telle que $\forall k \in [1, p_2], r_{1\sigma_l(k)} \leq_r r_{2k}$.

Exemple :
Soit la catégorie $Pièce\_rectangulaire : (A, C, L)$, définie par les listes :

$A(Pièce\_rectangulaire) = \langle hauteur : Réel, surface : Réel \rangle$
$C(Pièce\_rectangulaire) = \langle mur_1 : Mur, mur_2 : Mur, mur_3 : Mur, mur_4 : Mur,$
$plancher : Plan\_horizontal, plafond : Plan\_horizontal \rangle$
$L(Pièce\_rectangulaire) = \langle perp_1 : Perpendiculaire\_à, perp_2 : Perpendiculaire\_à,$
$perp_3 : Perpendiculaire\_à, perp_4 : Perpendiculaire\_à,$
$support_1 : Support\_de\_mur, support_2 : Support\_de\_mur,$
$support_3 : Support\_de\_mur, support_4 : Support\_de\_mur,$
$para_1 : Parallèle\_à, para_2 : Parallèle\_à \rangle$

où les liens $perp_i$ remplacent les $calage_i$ de la catégorie $Pièce$, et où, par exemple, le lien supplémentaire $para_1$ est défini par : $C(para_1) = \langle m_1 : mur_1, m_2 : mur_3 \rangle$.

$Pièce\_rectangulaire \leq_c Pièce$ est correct, avec les fonctions σ_a, σ_c et σ_l égales à l'identité sur les intervalles concernés et la supposition $Perpendiculaire\_à \leq_r Calage$.

2.3.3.3 Redéfinition des objets

Soit o un objet simple, tel que $C(o) = \langle ro_1 : o_1, ..., ro_m : o_m \rangle$. La liste $L(o)$ est formée de relations concrètes : rc_k, reliant des composants de o (issus de la liste $C(o)$).

$L(o) = \langle rc_1 : r_1, ..., rc_p : r_p \rangle$, avec: $\forall k \in [1,p], C(rc_k) = \langle ro_{k1} : o_{\sigma(1)}, ..., ro_{kn_k} : o_{\sigma(n_k)} \rangle$, où σ est une injection de $[1,n_k]$ dans $[0,m]$ (par convention encore, $o_0 = o$).

Exemple:
Reprenons l'objet o_1 défini dans 2.2.6, et qui représentait une image prise par un photographe (représenté par un objet o_2) et contenant deux personnes (représentées par les objets o_3 et o_4).
Nous complètons la définition de o_1 par la nouvelle liste:
$L(o_1) = \langle lien_1 : A\_gauche\_de \rangle$
avec $C(lien_1) = \langle gauche : o_3, droite : o_4 \rangle$, faisant référence à une relation *A_gauche_de*.
Le lien intitulé $lien_1$ représente le fait que l'objet o_3 est à gauche de l'objet o_4, dans l'image o_1.
Rappelons que o_3 et o_4 étaient des composants individuels de l'objet o_1 (de rôle *vide*).

2.3.3.4 Redéfinition du lien de réalisation sur les objets simples

La définition du lien de réalisation, telle qu'elle a été donnée dans 2.2.6.3, entre un objet simple o tel que $L(o) = \langle rc_1 : r'_1, ..., rc_{p'} : r'_{p'} \rangle$ et une catégorie simple c telle que $L(c) = \langle l_1 : r_1, ..., l_p : r_p \rangle$, doit être complétée par les conditions:

- $p' \geq p$,
- $\forall k \in [1,p]$:
  - $r'_k \leq_r r_k$
  - si $l_k = \langle ro_{k1} : ro_{\sigma(1)}, ..., ro_{kn_k} : ro_{\sigma(n_k)} \rangle$ et $C(rc_k) = \langle ro_{k1} : o_{\sigma'(1)}, ..., ro_{kn'_k} : o_{\sigma'(n'_k)} \rangle$ avec $n'_k \geq n_k$, alors pour tout composant de relation de rôle ro_{kl} ($l \in [1,n_k]$), l'objet $o_{\sigma'(l)}$ joue le rôle $ro_{\sigma(l)}$ dans la composition de o.

Exemples:
La nouvelle définition de o_1 vérifie encore: $o_1 \leftarrow Image$.

D'autre part, les figures 2.6 et 2.7 donnent des représentations graphiques d'une catégorie *Voiture* et d'un objet *Voiture 1* qui la réalise, en ajoutant un composant et une relation concrète individuels.
Le rectangle en pointillé de la figure 2.6 englobe une catégorie (*Any_ c*) et une relation (*Any_ r*), définies dans toute application EMIR, et respectivement génériques de toutes les catégories et de toutes les relations de l'application.

2.4 Discussion sur les liens inter-objets

Les liens entre objets revêtent, selon les travaux, différentes appellations. Cette différence de terminologie s'explique par la différence des domaines d'utilisation (méthodes de conception de systèmes d'information, systèmes à base de connaissances, sciences cognitives, ...), et par la finalité du système sous-jacent (stockage, interrogation des données,

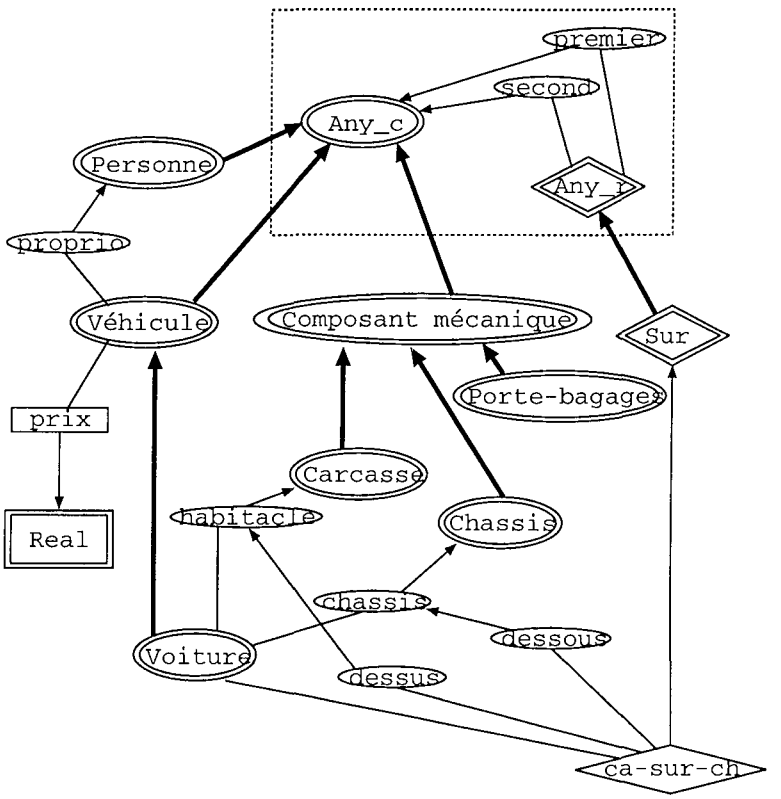


FIG. 2.6 – *Un schéma conceptuel EMIR*

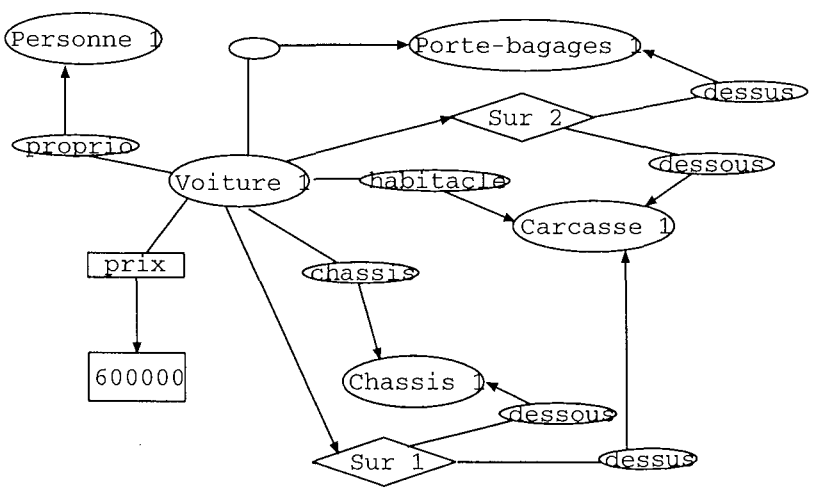


FIG. 2.7 – *Un schéma concret EMIR*

évolution dans le temps, inférence de propriétés, ...). On parle de lien d'agrégation, de composition, d'association, d'utilisation, ou encore de référence.

Différents travaux se sont intéressés à l'exploration des types de liens entre objets.

Certaines études dans le cadre des sciences cognitives [WCH87] ont fait émerger six différents types de liens entre objets dits "liens partie-tout", résumés dans [ODE94]. La différence entre les liens est basée sur des critères de configuration (si les parties de l'objet forment une certaine configuration), d'homéomérie (si les parties et le tout sont du même genre) et de séparation (si les parties sont séparables du tout).

Dans le cadre du SGBD ORION [BCG+87, KBC+87, KBG89], les liens entre des *objets composites* et leurs objets composants sont caractérisés par des règles impliquant les objets liées. Ces règles traduisent la dynamique sous-jacente des objets liés lors de la disparition de l'un d'entre eux. Cette approche a cependant le défaut de non exhaustivité puisqu'elle se limite aux liens de composition (les liens de simple référence sont cités mais ne sont pas explorés). Ces liens de composition sont aussi étudiés dans [MAG94], où sont explorés les problèmes liés aux exceptions dans les graphes de composition d'objets.

Dans le cadre du système à base de connaissances SHOOD [NR91, DNR93], un certain nombre de liens sont exposés, qui recouvrent des aspects de dépendance et de partage. La liste est assez riche ; elle souffre cependant de l'absence d'une spécification formelle de ces liens et d'un guide méthodologique qui facilite leur utilisation.

Dans notre modèle, nous nous intéressons particulièrement à la connaissance qu'a un objet des autres objets. Le but est d'explorer les différents types de liens possibles entre les objets, en précisant pour chaque type, ce qui le caractérise et l'effet induit sur l'évolution de la base.

Nous adoptons une approche pragmatique, reposant sur la dynamique que peut engendrer tel ou tel type de lien sur l'évolution conjointe des deux objets protagonistes. Nous sélectionnons pour cela des critères principaux, pondérés par des critères auxiliaires. Cette approche est largement présentée dans [BLM95], dans le cadre d'une base d'objets classique. Nous en présentons ici une synthèse adaptée à notre modèle, ébauchée dans [BLMN94].

Un lien entre deux objets est représenté dans l'un des objets, par l'intermédiaire d'un attribut ayant pour valeur l'autre objet. Ce lien est orienté, l'objet o référence o' si o connaît l'identifiant de o'. L'objet o est alors appelé *objet origine* (il est à l'origine du lien) et o' est appelé *objet destination.*

La sémantique du lien est fortement liée aux comportements des objets, et plus particulièrement à l'influence que peut avoir la disparition de l'un d'entre eux sur l'autre. Cette constatation nous a conduit à identifier deux critères principaux, qui caractérisent les liens entre objets :

- le critère de dépendance / indépendance :

  si la disparition de l'objet origine implique la disparition de l'objet destination, alors le lien est dit *dépendant*, sinon il est dit *indépendant*,

- le critère obligatoire / optionnel :

  si la disparition de l'objet destination implique la disparition de l'objet origine, alors le lien est dit *obligatoire*, sinon, il est dit *optionnel.*

Ces critères sont résumés dans le tableau ci-dessous, où l'événement de mort (disparition) d'un objet o est représenté par $Mort(o)$ et où une ligne du tableau exprime une règle d'identification de la nature du lien en se fondant sur l'incidence de la mort d'un des objets sur l'autre.

$Mort(o) \Rightarrow Mort(o')$	$Mort(o') \Rightarrow Mort(o)$	*Nature du lien*
Faux	Faux	Indépendant et optionnel
Faux	Vrai	Indépendant et Obligatoire
Vrai	Faux	Dépendant et optionnel
Vrai	Vrai	Dépendant et Obligatoire

Cependant, ces règles ne sont valables que dans le cas où les objets sont isolés du reste de la base ; les autres liens les faisant intervenir n'étant pas pris en compte. Nous allons donc utiliser deux critères auxiliaires, en plus des deux critères principaux précédents :

- le critère de monovaluation / multivaluation : si le même lien part d'un objet origine vers plusieurs objets destination (un objet ensemble), alors le lien est dit *multivalué*, sinon il est dit *monovalué*,

- le critère d'exclusivité / partage : si plusieurs objets origines peuvent avoir le même objet destination, alors le lien est dit *partagé*, sinon il est dit *exclusif.*

Ces critères s'inspirent des travaux cités précédemment portant sur les liens de composition. Nous adoptons une approche pragmatique, caractérisant les liens par la dynamique qu'ils infèrent au niveau des objets. De plus, nous ne plaçons pas tous les critères au même niveau, en pondérant l'effet des critères principaux par celui des critères auxiliaires. Enfin, nous ne limitons pas notre approche qu'aux seuls liens de composition ; nous parlerons plutôt de liens de connaissance, ou simplement de liens inter-objets.

Dans la suite de cette section, nous donnons les définitions des différents liens inter-objets dans une application, tout en tenant compte des critères énoncés précédemment. Ensuite, nous étudions les dépendances sous-jacentes aux différents types de liens, en soulignant l'aspect déterminant de certains critères et l'influence des autres.

2.4.1 Définitions préliminaires

Avant de présenter les différents types de liens inter-objets, nous introduisons quelques définitions préliminaires.

Soit $\mathbf{A}$ l'ensemble des noms de composants définis sur les catégories et les objets. Soient encore les ensembles de types de liens, $\mathbf{T_1} = \{dépendant, indépendant\}$ et $\mathbf{T_2} = \{obligatoire, optionnel\}$. Enfin, soit l'ensemble de leurs combinaisons possibles :

$\mathbf{T} = \{\{dépendant, obligatoire\}, \{dépendant, optionnel\},$
$\{indépendant, obligatoire\}, \{indépendant, optionnel\}\}.$

L'ensemble $\mathbf{L_o} \subseteq \mathbf{O} \times \mathbf{O} \times \mathbf{A} \times \mathbf{T}$ est alors défini comme étant l'ensemble des liens reliant les objets de l'application. Un lien est identifié par l'objet origine, l'objet destination, le nom du composant permettant de représenter le lien, et le type de lien.

Pour une facilité de notation, nous supposons que tous les objets sont simples. Dans le cas d'un composant multivalué (objet destination de type ensemble), nous mettrons un lien entre l'objet origine et *chacun* des objets appartenant à l'objet destination. Par exemple, pour les portières d'une voiture, si l'objet représentant les portières de la voiture v_1 est défini par l'ensemble $\{p_1, p_2, p_3, p_4\}$, on aura $(v_1, p_1, portières, \{dépendant, obligatoire\}) \in \mathbf{L_o}$; idem pour p_2, p_3 et p_4.

Nous définissons les fonctions suivantes :

$Composants : \mathbf{O} \rightarrow \mathcal{P}(\mathbf{A})$	fonction qui associe à un objet l'ensemble de ses composants, $Composants(o) = \{a \in \mathbf{A} \mid \exists o' \in \mathbf{O}, \exists t \in \mathbf{T}, < o, o', a, t > \in \mathbf{L_o}\}$
$Types : \mathbf{O} \rightarrow \mathcal{P}(\mathbf{T})$	fonction qui associe à un objet les types des liens l'ayant pour origine, $Types(o) = \{t \in \mathbf{T} \mid \exists o' \in \mathbf{O}, \exists a \in \mathbf{A}, < o, o', a, t > \in \mathbf{L_o}\}$
$Destinations : \mathbf{O} \rightarrow \mathcal{P}(\mathbf{O})$	fonction qui associe à un objet origine l'ensemble des objets destinations avec lesquels il est lié, $Destinations(o) = \{o' \in \mathbf{O} \mid \exists a \in \mathbf{A}, \exists t \in \mathbf{T}, < o, o', a, t > \in \mathbf{L_o}\}$
$Origines : \mathbf{O} \rightarrow \mathcal{P}(\mathbf{O})$	fonction qui associe à un objet destination l'ensemble des objets origine avec lesquels il est lié, $Origines(o') = \{o \in \mathbf{O} \mid \exists a \in \mathbf{A}, \exists t \in \mathbf{T}, < o, o', a, t > \in \mathbf{L_o}\}$

$Destinations_{lien} : \mathbf{O} \times \mathbf{A} \rightarrow \mathcal{P}(\mathbf{O})$	fonction qui associe à un objet, l'ensemble des objets destinations avec lesquels il est lié pour un lien donné (composant donné), $Destinations_{lien}(o,a) = \{o' \in \mathbf{O}, \mid \exists t \in \mathbf{T}, < o, o', a, t > \in \mathbf{L_o}\}$
$Origines_{lien} : \mathbf{O} \times (\mathbf{T_1} \cup \mathbf{T_2}) \rightarrow \mathcal{P}(\mathbf{O})$	fonction qui associe à un objet, l'ensemble des objets origines avec lesquels il est lié pour un type de lien donné, $Origines_{lien}(o',t) = \{o \in \mathbf{O} \mid \exists t' \in \mathbf{T}, t \in t', < o, o', a, t' > \in \mathbf{L_o}\}$

Invariants :
$Destinations(o) = \cup \{Destinations_{lien}(o,a), a \in Composants(o)\}$
$Origines(o) = \cup \{Origines_{lien}(o,t), t \in (\mathbf{T_1} \cup \mathbf{T_2})\}$

2.4.2 Caractérisation des différents types de liens

Avant de définir les différents types des liens inter-objets, nous introduisons, tout d'abord, le concept d'événement de mort. Cette notion va nous permettre de caractériser certaines propriétés dynamiques du lien. Un événement de mort est tout simplement un événement qui provoque la disparition logique ou physique d'un objet.

Plus formellement, soit $o \in \mathbf{O}$:

$$Mort(o) \Rightarrow \mathbf{O} = \mathbf{O} - \{o\},$$
$$\mathbf{L_o} = \mathbf{L_o} - \{< o, o', a, t >, o' \in Destinations(o), a \in Composants(o), t \in Types(o)\}$$
$$-\{< o', o, a, t >, o' \in Origines(o), a \in Composants(o'), t \in Types(o')\}.$$

Donnons maintenant la définition des différents types de liens inter-objets en fonction de leur nature :

- **dépendant / indépendant**
  Lorsque l'objet destination n'a d'existence qu'à travers l'objet origine, le lien est dit dépendant. Au contraire, s'il n'exerce aucune influence sur l'objet destination, il est dit indépendant.

  Soit $l =< o, o', a, t > \in \mathbf{L_o}$ avec $o, o' \in \mathbf{O}, a \in \mathbf{A}, t \in \mathbf{T}$,

  Si $Mort(o) \Rightarrow Mort(o')$,
  alors l est dépendant,
  sinon l est indépendant.

  Exemples :
  L'explosion d'une voiture provoque la destruction de ses roues, de son moteur ...
  L'annulation d'une image n'a pas d'impact sur la vie des objets auxquels elle fait référence.

- **obligatoire / optionnel**
  Lorsque l'objet destination est nécessaire pour l'existence de l'objet origine, le lien est dit obligatoire et, au contraire, lorsqu'il n'est pas indispensable, le lien est dit optionnel.

  Un composant peut être obligatoire pour au moins deux raisons : il provient d'une catégorie que l'objet réalise, ou encore il est propre à l'objet, mais traduit un lien obligatoire individuel entre l'objet et son composant.

  Soit $l =< o, o', a, t > \in \mathbf{L_o}$ avec $o, o' \in \mathbf{O}, a \in \mathbf{A}, t \in \mathbf{T}$,

  Si $Mort(o') \Rightarrow Mort(o)$,
  alors l est obligatoire,
  sinon l est optionnel.

  Exemples :
  Une commande est valide uniquement si elle est associée à un client (lien obligatoire provenant de la catégorie *Commande*).
  Le lien qui unit une voiture à son toit ouvrant, n'est pas nécessaire pour le fonctionnement de la voiture.

Comme nous l'avons mentionné précédemment, ces règles ne sont valides que si l'on considère que les objets origine et destination ne sont liés à aucun autre objet de la base. Dans le cas général, il faut tenir compte des propriétés suivantes :

- **monovalué / multivalué**
  Lorsque l'objet origine est lié à un objet simple pour un lien donné (représenté par un rôle), le lien est dit monovalué et, au contraire, lorsqu'il est relié à un objet ensemble par l'intermédiaire de ce rôle, il est dit multivalué.

  Soit $l =< o, o', a, t > \in \mathbf{L_o}$ avec $o, o' \in \mathbf{O}, a \in \mathbf{A}, t \in \mathbf{T}$,

  Si o' est un objet ensemble,
  alors l est multivalué,
  sinon l est monovalué.

  Exemples :
  Un client peut avoir plusieurs commandes.
  Une voiture a un seul propriétaire.

- **exclusif / partagé**
  Soit $t \in \mathbf{T_1} \cup \mathbf{T_2}$ un type de lien élémentaire. Lorsque l'objet destination n'a qu'un seul objet origine pour t, le lien est dit exclusif pour t, dans le cas contraire, il est dit partagé pour t.

Soit $l =< o_1, o', a_1, t_1 > \in \mathbf{L_o}$ avec $o_1, o' \in \mathbf{O}, a_1 \in \mathbf{A}, t_1 \in \mathbf{T}, t \in t_1$,

Si $\exists o_2 \in \mathbf{O}, \exists a_2 \in \mathbf{A}, \exists t_2 \in \mathbf{T} \mid < o_2, o', a_2, t_2 > \in \mathbf{L_o}, t \in t_2$,
alors l est partagé pour t,
sinon l est exclusif pour t.

Exemples :
Les roues d'une voiture ne peuvent simultanément appartenir qu'à une seule voiture.
Un mur peut être partagé par deux pièces.

La combinaison de chacun de ces critères conduit à identifier 2^4 liens possibles. Nous détaillons, dans la suite, pour chacun des liens obtenus, la réaction des objets liés lorsque l'un d'eux disparaît de la base, en illustrant notre propos par des exemples.

2.4.3 Dynamique sous-jacente aux différents types de liens

Considérons le lien $< o, o', a, t > \in \mathbf{L_o}$, défini entre les objets o et o', o est l'objet origine et o' l'objet destination, $a \in \mathbf{A}, t \in \mathbf{T}$.

Le tableau de la page suivante recense les différents types de liens possibles entre deux objets ; il souligne également l'influence respective des caractères monovalué / multivalué et partagé / exclusif sur les caractères obligatoire et dépendant.

Pour chaque ligne, si la valeur de l'expression contenue dans la colonne intitulée "$Mort(o) \Rightarrow Mort(o')$" est "Vrai" alors, cette dernière implication est vraie ; idem pour la colonne intitulée "$Mort(o') \Rightarrow Mort(o)$".
La dernière colonne donne des exemples des différents types des liens en précisant la catégorie des objets origine et destination et le rôle du composant.

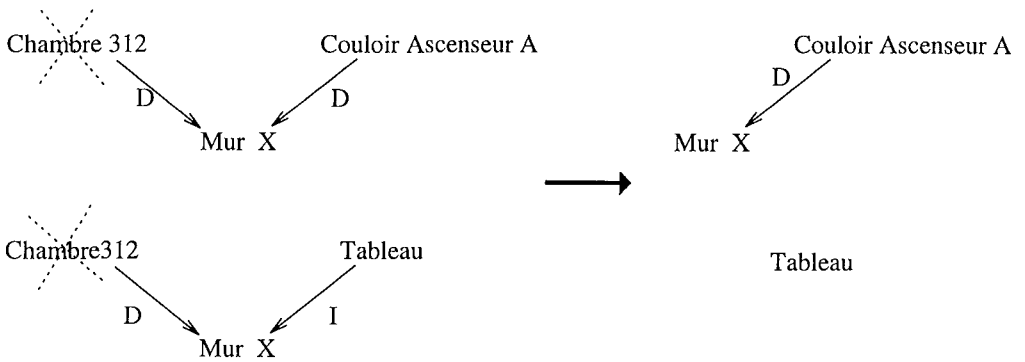
Abréviations :

D	lien dépendant
I	lien indépendant
O	lien obligatoire
o	lien optionnel
P	lien partagé
E	lien exclusif
M	lien multivalué
m	lien monovalué

D/I	O/o	P/E	M/m	Mort(o) ⇒ Mort(o')	Mort(o') ⇒ Mort(o)	Exemple (c,c',a)
I	o	E	m	Faux	Faux	(Personne, Personne, conjoint)
I	o	E	M	Faux	Faux	(Personne, Voiture*, voitures)
I	o	P	m	Faux	Faux	(Personne, Personne, enfant-de)
I	o	P	M	Faux	Faux	(Chapitre, Figure*, figures)
I	O	E	m	Faux	Vrai	(Voiture, Volant, volant)
I	O	E	M	Faux	$Destinations_{lien}(o, a) = \{o'\}$	(Voiture, Roue*, roues)
I	O	P	m	Faux	Vrai	(Commande, Personne, client)
I	O	P	M	Faux	$Destinations_{lien}(o, a) = \{o'\}$	(Projet, Employé*, équipe)
D	o	E	m	Vrai	Faux	(Voiture, Ouverture, toit-ouvr.)
D	o	E	M	Vrai	Faux	(Voiture, Rétroviseur*, rétros)
D	o	P	m	$Origines_{lien}(o', dépendant) = \{o\}$	Faux	(Pièce, Cheminée, cheminée)
D	o	P	M	$Origines_{lien}(o', dépendant) = \{o\}$	Faux	(Pièce, Porte*, portes)
D	O	E	m	Vrai	Vrai	(Voiture, Moteur, moteur)
D	O	E	M	Vrai	$Destinations_{lien}(o, a) = \{o'\}$	(Voiture, Ouverture*, portières)
D	O	P	m	$Origines_{lien}(o', dépendant) = \{o\}$	Vrai	(Pièce, Plancher, plancher)
D	O	P	M	$Origines_{lien}(o', dépendant) = \{o\}$	$Destinations_{lien}(o, a) = \{o'\}$	(Pièce, Mur*, murs)

Les exemples suivants illustrent la priorité de certains liens, sur la dynamique des objets lorsque les objets origine et / ou destination entrent en jeu dans d'autres liens.

En effet (cf figure 2.8), si l'objet destination est partagé par un autre lien de type dépendant, celui-ci invalide les actions qui sont implicites dans le lien dépendant initial.

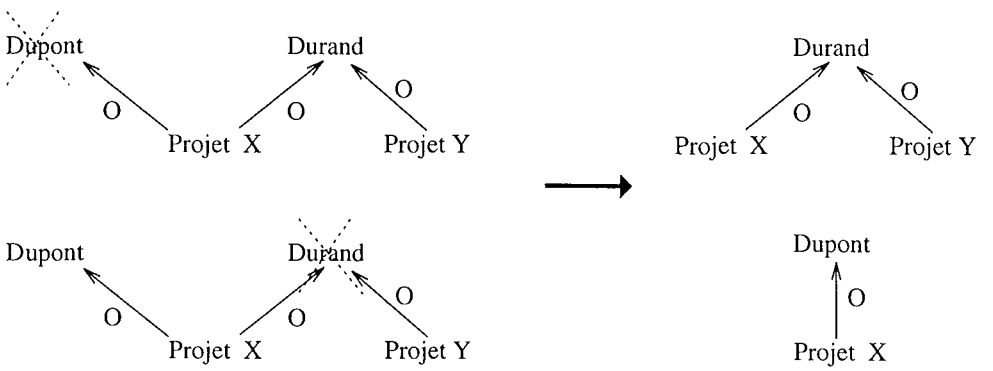


Légende :

D : lien dépendant
I : lien indépendant
: mort de l'objet

FIG. 2.8 – *Influence du caractère partagé sur le caractère dépendant*

De façon similaire (cf figure 2.9), l'aspect multivalué modifie la règle induite par le caractère obligatoire du lien. Par exemple, dans une société de services informatiques, au moins un employé est affecté à un projet et seule la suppression du dernier entraine la disparition du projet.

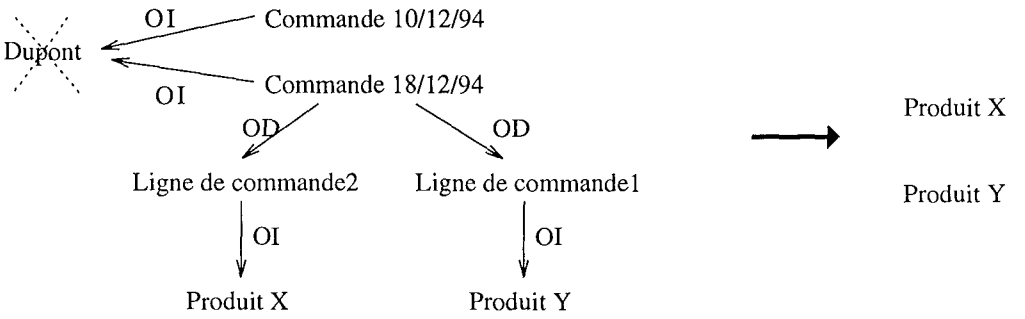


Légende :

O : lien obligatoire
: mort de l'objet

FIG. 2.9 – *Influence du caractère multivalué sur le caractère obligatoire*

Nous montrons dans l'exemple suivant (figure 2.10), que le type de lien partagé (respectivement multivalué) n'a pas d'influence sur le type obligatoire (respectivement dépendant).



Légende :

OI : lien obligatoire et indépendant
OD : lien obligatoire et dépendant
: mort de l'objet

FIG. 2.10 – *Combinaison de liens*

Comme nous pouvons le remarquer sur cet exemple, la suppression d'un objet peut avoir des conséquences importantes sur la base d'objets, car sa disparition est susceptible de provoquer celles d'autres objets, entraînant ainsi une vague de suppressions en cascade.

Le tableau recensant les différents types de liens, nous permet donc de spécifier entièrement les aspects dynamiques induits par les caractères dépendant/indépendant, obligatoire/optionnel, pondérés par les critères de partage et de multivaluation.

Soit $l =< o, o', a, t > \in \mathbf{L_o}, o, o' \in \mathbf{O}, a \in \mathbf{A}, t \in \mathbf{T}$

- cas d'un lien dépendant
  $Mort(o) \Rightarrow Mort(o')$ si et seulement si $Origines_{lien}(o', dépendant) = \{o\}$
- cas d'un lien indépendant
  $Mort(o) \not\Rightarrow Mort(o')$
- cas d'un lien obligatoire
  $Mort(o') \Rightarrow Mort(o)$ si et seulement si $Destinations_{lien}(o, a) = \{o'\}$
- cas d'un lien optionnel
  $Mort(o') \not\Rightarrow Mort(o)$

2.4.4 Redéfinition de quelques concepts du modèle

Après avoir étudié la sémantique des différents types de liens qui peuvent exister entre les objets, nous avons proposé une approche dynamique pour définir et identifier les liens inter-objets.

Notre démarche s'est appuyée sur des critères principaux (la dépendance et l'indépendance des objets liés, le caractère obligatoire ou optionnel du lien). Ces critères sont caractérisés par des règles qui régissent le comportement des objets liés lorsque l'un d'eux disparait. Nous avons identifié d'autres critères (partage, multivaluation), qui influencent la réaction des objets, par rapport aux premiers critères.

Afin de pouvoir utiliser ces caractéristiques dynamiques induites par les différents types de liens, il est nécessaire d'enrichir la structure des objets du modèle, redéfinie dans 2.3.3.3, en tenant compte des types de liens.

D'autre part, il est aussi intéressant d'intégrer ce genre d'informations au niveau des catégories, parce qu'elles représentent une abstraction des caractéristiques minimales communes à tous les objets qui les réalisent.

2.4.4.1 Enrichissement de la structure des objets

On enrichit la définition de la liste $C(o)$ d'un objet simple o, en attachant à chaque élément $ro_j : o_j$ un drapeau statuant si le lien est dépendant-obligatoire (DO), dépendant-optionnel (Do), indépendant-obligatoire (IO) ou indépendant-optionnel (Io). Les critères secondaires de partage et de monovaluation sont implicites (calculés) dans la base ; ils n'ont pas à être spécifiés au niveau de la structure des objets.

Exemple :
En reprenant l'objet image o_1, défini dans 2.2.6.1, sa liste de composition peut être enrichie comme suit :
$C(o_1) = \langle photographe : o_2(IO), vide : o_3(Io), vide : o_4(Io) \rangle)$.
Il n'y a pas de lien de dépendance, car les composants d'une image ne sont que des références de cette image à d'autres objets, dont l'existence n'est nullement liée à celle de l'image.

2.4.4.2 Enrichissement de la structure des catégories

La définition de la liste $C(c)$ d'une catégorie simple c est enrichie, en attachant à chaque élément $ro_j : c_j$ un drapeau statuant si le lien est dépendant (D), ou indépendant (I). Le critère obligatoire au niveau des objets est implicite dans le cadre d'un composant de catégorie.

Exemple :
Dans la catégorie $Image$, définie dans 2.2.5.1, la liste de composition peut être enrichie comme suit :
$C(Image) = \langle photographe : Personne(I) \rangle)$.

Dans [BLM95], sont décrites des heuristiques pour l'inférence de liens entre classes dans un modèle objet, à partir de liens entre objets. Elles statuent en particulier que si un lien est dépendant dans une classe, il le sera dans toutes ses instances.

Dans le cadre du modèle EMIR, étant donné une catégorie c et un de ses composants de catégorie c', un objet o réalisant c peut avoir, pour ce composant, un objet dont la catégorie est plus spécifique que c' (cf. définition du lien de réalisation). Dans le souci d'assurer une grande liberté des objets par rapport à leurs catégories, il nous parait naturel d'autoriser un objet à "transgresser" aussi le type du lien, en proposant un type de lien "plus spécifique". Cette approche de surcharge a déjà été adoptée auparavant, notamment dans le modèle SHOOD [DNR93, DJE94].

Pour cela, nous proposons une définition intuitive de cette notion, qui repose sur une constatation simple: au niveau du comportement des objets de la base par rapport aux types de lien (cf. 2.4.3), un lien dépendant est plus restrictif (donc plus spécifique) qu'un lien indépendant, et un lien obligatoire est plus restrictif (donc plus spécifique) qu'un lien optionnel.

La spécificité des types de liens est définie en extension par la relation $\leq_{tl}$ allant du plus "spécifique" au plus "générique":

$$DO \leq_{tl} Do \leq_{tl} D \leq_{tl} IO \leq_{tl} Io \leq_{tl} I.$$

Cet ordre servira notamment à l'affinement des définitions de l'héritage et de la réalisation.

2.4.4.3 Redéfinition du lien d'héritage

Dans la définition donnée de l'héritage dans 2.2.5.3 entre une catégorie spécifique c_1 et une catégorie générique c_2, il faut spécifier, pour chaque composant de c_2, non seulement que la catégorie qui lui est liée est plus générique que celle qui se trouve au niveau de c_1, mais que le type de lien l'est aussi.

2.4.4.4 Redéfinition du lien de réalisation

Dans la définition donnée de la réalisation dans 2.2.6.3, il faut spécifier, pour chaque composant de la catégorie, qu'au niveau de l'objet, le type de lien est plus spécifique.

2.5 Bases de données EMIR

Une base de données, selon notre modèle, est un quadruplet $(\mathbf{D}, \mathbf{R}, \mathbf{C}, \mathbf{O})$, muni des relations binaires $\leq_d$, $\leq_r$, $\leq_c$ et $\leftarrow$, où:

- **D** est un ensemble de domaines, muni d'un ordre partiel $\leq_d$ (calculé automatiquement à partir des règles données dans 2.2.3.3),

- **R** est un ensemble de relations muni d'un ordre partiel $\leq_r$ (entièrement défini par l'utilisateur),

- **C** est un ensemble de catégories muni d'un ordre partiel $\leq_c$ (défini par l'utilisateur sur les catégories simples, et complété automatiquement à partir des règles données dans 2.2.5.3),

- **O** est un ensemble d'objets avec une relation binaire $\leftarrow$, définie sur $\mathbf{O} \times \mathbf{C}$ (conformément à la partie 2.2.6.3), représentant le lien de réalisation.

La définition ci-dessus regroupe la notion de schéma et d'instance de la base de données. La première, appelée *schéma conceptuel*, comprend des domaines, des catégories, des relations et les ordres partiels associés. La seconde, appelée *schéma concret*, regroupe des objets, des valeurs, des relations concrètes.

Conclusion

Dans ce chapitre, nous avons proposé un modèle de données (EMIR), dédié à la modélisation d'objets complexes à forte structure individuelle, et dont le but est de rapprocher les domaines de la recherche d'information et des bases de données. Ce modèle permet de définir au sein d'un même environnement, des objets dont la structure est bien définie et d'autres pour lesquels il n'est pas possible d'abstraire *a priori* toutes les caractéristiques au sein de classes d'objets.

Nous avons illustré la définition des concepts par des exemples issus des deux domaines.

Le modèle se base sur les notions de domaines de valeurs, de catégories d'objets, de relations, de valeurs, d'objets et de relations concrètes. Un objet du modèle EMIR est modélisé selon trois aspects : la description, qui comprend des attributs avec leurs valeurs, la composition, qui comprend des objets avec un rôle (éventuellement vide) et la topologie, qui comprend des liens sémantiques entre les composants.

Le modèle s'appuie également sur des liens de généralisation entre catégories et entre relations, et surtout sur le lien de réalisation entre un objet et une catégorie. Ce lien permet d'abstraire dans une catégorie, toutes les caractéristiques communes aux objets qui lui sont reliés. Chacun de ces objets a, en plus, la possiblité d'avoir des caractéristiques individuelles, sous la forme de composants et de liens supplémentaires.

Nous avons par ailleurs, étudié les différentes natures du lien entre un objet et un de ses objets composants. Nous avons identifié seize types de liens (quatre principaux) et nous avons étudié leur influence sur la dynamique des objets.

Le modèle EMIR trouve son inspiration dans différents travaux issus à la fois du domaine des bases de donnés, des langages de programmation orientés objet (classes, prototypes) ou encore de la recherche d'informations dans les bases documentaires.

Or, définir une base de données ne se borne pas à structurer les données au sein d'un schéma conceptuel et alimenter la base. Encore faut-il assurer la cohérence des données et pouvoir interroger la base pour rechercher de l'information.

Dans le chapitre 3, nous présentons les mécanismes de gestion de l'intégrité d'une base de données EMIR, ainsi que la manière d'interroger celle-ci.

Chapitre 3

Maintien de l'intégrité et interrogation

Le modèle de données étant défini, nous nous intéressons dans le ce chapitre à deux fonctionnalités indispensables et complémentaires au modèle de données : le maintien de l'intégrité et l'interrogation des données.

Dans la partie 3.1, deux types d'intégrité sont étudiés : une intégrité inhérente aux constructeurs du modèle, indépendante des applications, et une intégrité sémantique, définie par l'utilisateur au sein d'une application donnée. Pour cette dernière, nous avons défini un langage adapté au modèle, permettant d'attacher des contraintes d'intégrité à des catégories, des objets particuliers ou encore des relations.

Par ailleurs, nous définissons (dans la partie 3.2) un langage de requêtes dédié au modèle, qui, tout comme le langage de contraintes, tire profit de toutes les spécificités du modèle de données. En particulier, des requêtes basées sur le contenu (du type de celles des systèmes de recherche d'information) peuvent être exprimées au même titre que des requêtes navigationnelles (basées sur la structure), plus habituelles dans les bases de données classiques.

3.1 Maintien de l'intégrité

La spécification et la gestion des contraintes d'intégrité est nécessaire pour la cohérence des données dans une base de données, au même titre que la synchronisation des accès concurrents ou les mécanismes de reprise sur panne.

Depuis le modèle relationnel [DA82, BDT91], en passant par les modèles sémantiques de données [CRZNM88, PMB+89, CP92, CQ92], et actuellement les modèles orientés objet [NQZ91, SA91, KLS92, KHO93, AF94], chaque modèle s'est doté de moyens de maintenir la cohérence de ses données, d'une manière adéquate, tenant compte de ses spécificités.

On peut énumérer deux types de contraintes d'intégrité [DA82, NQZ91, FTS96] :

- les contraintes statiques spécifient des conditions que doit vérifier une base de données, à tout moment ; elles expriment des combinaisons valides des données de la base, par rapport aux constructeurs d'entités (par exemple, les voitures officielles sont de couleur sombre) ;

- les contraintes dynamiques définissent les transitions correctes de la base, d'un état à un autre, autrement dit, les changements d'état valides (par exemple, les salaires ne peuvent diminuer).

D'autres typologies [MWW89] distinguent parmi les contraintes statiques ou dynamiques, des contraintes appelées "déontiques" des contraintes "nécessaires". Les contraintes déontiques expriment des contraintes "habituellement" respectées par les données, et qui peuvent être temporairement violées. Les contraintes nécessaires expriment des contraintes inviolables.

La différence réside surtout dans la réaction du système lors de la violation d'une contrainte. S'il s'agit d'une contrainte nécessaire (par exemple, $\hat{a}ge \leq 150$), elle doit être

corrigée sur le champ. S'il s'agit d'une contrainte déontique (par exemple, un adhérent d'une bibliothèque doit rendre un livre, au plus tard 3 semaines après l'emprunt), la réaction du système peut se limiter à un traitement particulier de nature à rétablir ultérieurement la contrainte (envoi d'une notification à l'intéressé).

Nous nous intéresserons plus particulièrement aux contraintes statiques, parmi elles [CQ92] :

- les contraintes *globales*, qui expriment la capacité du modèle de données de tenir compte de certains types de données particuliers (par exemple, alphanumérique, sonore, multimédia, ...) ;
- les contraintes *sur le modèle de données*, qui précisent les règles d'utilisation et de combinaison correctes des constructeurs du modèle (par exemple, chaque objet instancie une classe et une seule) ;
- les contraintes *sur le schéma de la base*, qui concernent un schéma particulier de base de données, défini par l'utilisateur ; elles précisent des conditions particulières inhérentes à la sémantique de l'application (par exemple, un étudiant ne dépasse pas trente ans d'âge) ;
- les contraintes *sur les données*, qui concernent des objets particuliers de la base (par exemple, le client X ne doit pas dépasser un débit de 4000 Frs).

Parmi ces différents types de contraintes statiques, les plus étudiées sont celles, liées au modèle de données, et celles, liées aux schémas d'applications.

Le modèle relationnel [DA82] distingue parmi les contraintes liées au schéma, les types suivants :

- unicité de la clé : c'est une contrainte spécifiant que les valeurs de la projection sur un groupe d'attributs donné d'une relation (appelé clé de la relation) ne peuvent être identiques dans deux tuples différents ;
- contraintes individuelles : elles portent sur un seul tuple à la fois et peuvent être définies par des plages de valeurs, des listes de valeurs possibles, des formats de valeurs, ou des contraintes inter-attributs (par exemple, la quantité en stock est supérieure à la quantité commandée) ;
- contraintes intra-relation : elles spécifient des conditions globales sur l'ensemble des tuples d'une relation ; elles sont définies soit sur un tuple par rapport à l'ensemble (un employé ne peut gagner plus du double de la moyenne des salaires de son équipe) ; soit sur tous les tuples, au moyen d'un opérateur d'agrégation (la moyenne des salaires d'une équipe est au plus de 10000 Frs) ;
- contraintes inter-relations : généralement appelées *dépendances entre données* : dépendances fonctionnelles, multivaluées, hiérarchiques, produit, ... [DA82] ; en particulier l'intégrité référentielle statue que la valeur d'un attribut dans une relation donnée (référençante) est une clé dans une autre relation (référencée).

Les modèles sémantiques et les modèles de bases de données orientées objet distinguent d'autres types de contraintes [CQ92, KHO93, AF94] :

- contraintes existentielles : équivalentes aux contraintes référentielles en relationnel, elle font intervenir la notion d'identité d'objet et statuent que la valeur d'un attribut de la classe référençante doit appartenir à un ensemble donné, précisé par la classe référencée ;
- contraintes de spécialisation : spécialisent une classe en restreignant ses instances à celles répondant à certains critères ;
- contraintes de disjonction : l'intersection des ensembles d'instances des sous-classes d'une même classe est vide ;
- contraintes de recouvrement : la réunion des ensembles d'instances des sous-classes d'une classe constitue exactement l'ensemble des instances de celle-ci.

La spécification des contraintes d'intégrité peut être effectuée selon différentes terminologies [NQZ91]. Les *équations* sont des contraintes atomiques, consistant en deux expressions définies par des chemins dans le schéma de la base, séparées par un opérateur. Les *assertions* sont des combinaisons d'équations par des opérateurs booléens, mais n'introduisant pas de quantificateur. Des contraintes plus complexes peuvent être définies en utilisant des sous-ensembles plus ou moins restrictifs du calcul des prédicats [CQ92].

Une fois les contraintes spécifiées, des approches d'analyse de contraintes visent à étudier leur inter-dépendance d'un point de vue logique, afin d'en dériver des protocoles de transactions élémentaires sur les objets [UD90, CP92], et aussi afin de fournir des explications sur la sémantique des objets et des traitements utilisés pour assurer le maintien de la cohérence de la base [UD88].

La gestion des contraintes d'intégrité donne lieu à différentes stratégies, pour la restauration de la cohérence de la base quand une contrainte est violée. La stratégie la plus triviale consiste à interdire toute action sur la base, de nature à violer une contrainte (particulièrement les mises à jour). Des stratégies moins restrictives utilisent la notion de *transaction*. Une transaction est constituée d'une suite d'actions sur la base séparant deux états cohérents de celle-ci. Au cours de l'exécution de ces actions, des contraintes peuvent être temporairement violées [DA82, CQ92].

Une autre approche vise à maîtriser complètement le comportement de la base de données en assurant sa cohérence à l'issue de chaque action élémentaire (ajout, modification, suppression) [AV89, WOR91]. La spécification d'une contrainte d'intégrité n'est pas effectuée d'une manière prédicative, mais d'une manière dynamique, à travers les opérations susceptibles de la violer, en y intégrant des mécanismes qui permettent d'éviter cette violation. Cette approche est particulièrement intéressante dans un environnement orienté objet, où le mécanisme d'encapsulation facilite cette spécification dynamique.

Deux problèmes majeurs se posent :

- assurer la cohérence des transactions élémentaires (correction),
- assurer une expressivité équivalente à la spécification prédicative (complétude). Dans [AV89], la complétude est assurée pour certains types de contraintes (principalement les dépendances fonctionnelles).

Dans la suite, nous présentons deux aspects du contrôle de l'intégrité statique dans le modèle EMIR.

L'intégrité référentielle (en l'occurrence existentielle) est décrite dans la partie 3.1.1 ; elle assure la cohérence de la base vis-à-vis de la sémantique des constructeurs du modèle ; elle est donc indépendante des applications. Nous adoptons une approche transactionnelle pour assurer le maintien des contraintes liées au modèle. Nous définissons d'abord les contraintes, ensuite, nous étudions pour chacune d'elles les opérations sur le schéma qui sont susceptibles de la violer, en proposant des préconditions et des post-conditions qui doivent être respectées par l'opération. Il s'agit donc d'une approche algébrique opérationnelle.

L'intégrité sémantique des applications est explorée dans la partie 3.1.2, sous une autre forme : celle d'un langage de spécification de contraintes dépendantes des applications. Nous fournissons une manière de spécifier des contraintes d'intégrité, et une méthode pour les évaluer sur un état donné de la base. Il s'agit donc d'une approche déclarative, dans la mesure où nous ne nous intéressons pas (dans le cadre de cette thèse) au maintien réel de ce type de contraintes d'intégrité, mais uniquement à leur spécification et leur évaluation.

Les deux parties tiennent bien sûr compte des spécificités du modèle, notamment en ce qui concerne la différenciation entre catégories et relations.

3.1.1 Intégrité référentielle liée au modèle

Etant donné les définitions du chapitre 2, nous présentons dans cette section une manière d'assurer l'intégrité d'une base de données EMIR, vis-à-vis des concepts du modèle, et indépendamment des applications.

Pour cela, nous définissons dans 3.1.1.1 un certain nombre de propositions, appelées "invariants du modèle", qui doivent rester vraies à n'importe quel moment de la vie d'une base de données EMIR. Ensuite, dans 3.1.1.2, nous explorons les différentes opérations qui peuvent survenir dans la vie d'une base, la faisant passer d'un état à un autre. Nous définissons, pour chacune de ces opérations, les conditions d'application ainsi que les actions qui doivent être menées en même temps que l'opération, afin d'assurer le respect des invariants.

Une première ébauche de ce travail a été menée dans [NML93].

3.1.1.1 Invariants du modèle

Nous classons les invariants du modèle en huit classes :

3.1.1.1.1 Invariants de nommage (IN)

IND : Tout domaine a un nom unique sur l'ensemble des noms de domaines.

INR : Toute relation a un nom unique sur l'ensemble des noms de relations.

INC : Toute catégorie a un nom unique sur l'ensemble des noms de catégories.

INO : Tout objet a un nom unique sur l'ensemble des noms d'objets.

INAD : Tout attribut de domaine agrégé a un nom unique sur l'ensemble des noms d'attributs du domaine agrégé.

INAR : Tout attribut de relation a un nom unique sur l'ensemble des noms d'attributs de la relation.

INCR : Tout composant de relation a un rôle unique sur l'ensemble des rôles de composants de la relation.

INAC : Tout attribut de catégorie a un nom unique sur l'ensemble des noms d'attributs de la catégorie.

INCC : Tout composant de catégorie a un rôle unique sur l'ensemble des rôles de composants de la catégorie.

INLC : Tout lien de catégorie a un nom unique sur l'ensemble des noms de liens de la catégorie.

INCO : Tout composant obligatoire d'objet a un rôle unique sur l'ensemble des rôles de composants de l'objet.

INLO : Toute relation concrète d'objet a un nom unique sur l'ensemble des noms de relations concrètes de l'objet.

3.1.1.1.2 Invariants de domaines (ID)

IDD : Tout domaine discret a au moins une valeur.

IDC : La borne inférieure d'un domaine continu est inférieure à sa borne supérieure.

IDA : Tout domaine agrégé est composé d'au moins deux domaines.

3.1.1.1.3 Invariants d'acyclicité (IA)

IAD : Il n'y a pas de cycle dans la définition récursive des domaines (agrégés et multivalués).

IAHR : Il n'y a pas de cycle dans l'ordre partiel d'héritage entre relations.

IAHC : Il n'y a pas de cycle dans l'ordre partiel d'héritage entre catégories.

3.1.1.1.4 Invariants d'héritage de relations (IHR)

IHRG : Si une relation hérite d'une autre, les attributs et les composants de la relation générique se retrouvent, éventuellement renommés, dans la relation spécifique.

IHRA : Si un attribut d'une relation provient d'une relation générique, le domaine qui lui est associé dans la relation spécifique est égal ou plus spécifique que celui qui lui est associé dans la relation générique.

IHRC : Si un composant d'une relation provient d'une relation générique, la catégorie qui lui est associée dans la relation spécifique est égale ou plus spécifique que celle qui lui est associée dans la relation générique.

3.1.1.1.5 Invariants d'héritage de catégories (IHC)

IHCG : Si une catégorie simple hérite d'une autre, les attributs, les composants et les liens de la catégorie générique se retrouvent, éventuellement renommés, dans la catégorie spécifique.

IHCA : Si un attribut d'une catégorie simple provient d'une catégorie générique, le domaine qui lui est associé dans la catégorie spécifique est égal ou plus spécifique que celui qui lui est associé dans la catégorie générique.

IHCC : Si un composant d'une catégorie simple provient d'une catégorie générique, la catégorie qui lui est associée dans la catégorie spécifique est égale ou plus spécifique que celle qui lui est associée dans la catégorie générique.

IHCTC : Si un composant d'une catégorie simple provient d'une catégorie générique, le type de lien qui lui est associé dans la catégorie spécifique est égal ou plus spécifique que celui qui lui est associé dans la catégorie générique.

IHCL : Si un lien d'une catégorie simple provient d'une catégorie générique, la relation qui lui est associée dans la catégorie spécifique est égale ou plus spécifique que celle qui lui est associée dans la catégorie générique.

3.1.1.1.6 Invariants d'instanciation de relations (IIR)

IIRE : Toute relation concrète instancie une relation et une seule.

IIRG : Si une relation concrète instancie une relation, les attributs, les composants de la relation et ceux de toutes ses relations génériques sont "valués" au niveau de la relation concrète.

IIRA : Si un attribut de relation est valué au niveau d'une relation concrète, la valeur associée appartient au domaine de l'attribut.

IIRC : Si un composant de relation est "valué" dans une relation concrète, l'objet associé réalise la catégorie du composant.

3.1.1.1.7 Invariants de réalisation de catégories (IRC)

IRCE : Tout objet réalise une catégorie.

IRCG : Si un objet réalise une catégorie, les attributs, les composants et les liens de la catégorie sont "valués", au niveau de l'objet, avec les mêmes noms ou rôles.

IRCA : Si un attribut de catégorie est valué, au niveau d'un objet, la valeur associée appartient au domaine de l'attribut.

IRCC : Si un composant de catégorie est "valué" dans un objet, l'objet associé réalise la catégorie du composant.

IRCTC : Si un composant de catégorie est "valué" dans un objet, le type de lien associé est plus spécifique que celui défini dans la catégorie.

IRCL : Si un lien de catégorie est "valué" dans un objet, la relation concrète associée instancie la relation du lien, les rôles sont respectés et les valeurs des attributs correspondent aux domaines.

3.1.1.1.8 Invariants de composition d'objets (ICO)

ICOD : Si un objet o référence un objet o' avec un type de lien dépendant (DO ou Do), et si o' est uniquement référencé par o avec le type de lien en question, alors la disparition de l'objet o (référençant) entraîne celle de l'objet o' (référencé).

ICOO : Si un objet o référence un objet o' avec un type de lien obligatoire (DO ou IO), et si l'objet o' est le seul objet à être référencé, par l'attribut en question, dans l'objet o, alors la disparition de l'objet o' (référencé) entraîne celle de l'objet o (référençant).

3.1.1.2 Opérations possibles sur une base EMIR

Nous distinguons parmi les opérations possibles, des opérations de création, de renommage, de suppression et de modification de structure.

La présentation exhaustive des opérations se trouve dans l'annexe B. Nous nous limiterons ici à prendre quelques exemples jugés représentatifs, afin d'illustrer notre méthode de maintien des contraintes du modèle.

Comme dans l'annexe B, chaque opération est désignée par son nom, son intitulé, ses paramètres, ses préconditions, ainsi que les actions à mener en parallèle de l'opération, afin de maintenir l'intégrité de la base, vis-à-vis des invariants.

Contrairement aux règles ECA (Evénement-Condition-Action) traditionnellement utilisées dans le cadre des bases de données actives, l'opération elle-même (qui est aussi l'événement déclenchant) n'est effectuée que si les préconditions sont vérifiées. A ce moment, les actions auxiliaires sont aussi effectuées, en même temps que l'opération.

Il est cependant possible de représenter le maintien des invariants, dans un formalisme ECA :

- E : appel d'une opération,
- C : conditions vérifiées,
- A : exécution de l'opération, ainsi que ses actions auxiliaires.

Avant de présenter quelques opérations en détail, nous allons définir quelques conventions de notation ainsi que quelques ensembles dont nous nous servirons :

- tout d'abord, nous désignons par les ensembles $\mathbf{D}.nom$, $\mathbf{R}.nom$, $\mathbf{C}.nom$ et $\mathbf{O}.nom$, respectivement les ensembles de noms de domaines, de relations, de catégories et d'objets,
- les symboles $\cup=$ et $--$ utilisés dans $E \cup= F$ et $E -- F$, désignent respectivement les opérations d'adjonction et de retrait de l'ensemble F à l'ensemble E,
- pour un domaine discret d, $d.valeurs$ représente l'ensemble des valeurs de d,
- pour un domaine continu d, $d.bornes$ représente l'intervalle des valeurs de d,
- pour un domaine multivalué $d : \{d'\}$, $d.sous\_domaine$ représente le domaine d',
- pour un domaine agrégé d, $d.description$ représente l'ensemble des sous-domaines de d,
- pour une catégorie, un objet, une relation ou une relation concrète x, $x.description$ représente l'ensemble des attributs de x,
- pour une catégorie, un objet, une relation ou une relation concrète x, $x.composition$ représente l'ensemble des composants de x,
- pour une catégorie (resp. un objet) x, $x.topologie$ représente l'ensemble des liens (resp. des relations concrètes) de x,
- le symbole $*$, utilisé dans une précondition signifie qu'elle doit être vérifiée pour n'importe quelle valeur possible de $*$; s'il est utilisé dans une opération, cela signifie que l'opération doit être effectuée pour toutes les valeurs possibles de $*$.

L'initialisation d'une base de données EMIR se fait avec les ensembles suivants :

- $\mathbf{D} := \{Integer : I, String : S, Real : R, Boolean : B\}$, ces domaines prédéfinis ne peuvent être supprimés,
- $\mathbf{C} := \{Any_c\}$, catégorie vide généralisant toutes les catégories, elle ne peut être supprimée,
- $\mathbf{R} := \{Any_r\}$, relation générique de toutes les relations, elle n'a pas d'attribut, mais possède deux composants dont les rôles sont respectivement *premier* et *second*, tous les deux reliés à la catégorie Any_c ; cette relation ne peut être supprimée,

- $\mathbf{O} := \{\}$.

Nous définissons en outre les ensembles suivants :

- $RgenR \subseteq \mathbf{R} \times \mathbf{R}$, qui représente le graphe d'héritage de relations, initialisé à $\{(Any_r, Any_r)\}$,
- $CgenC \subseteq \mathbf{C} \times \mathbf{C}$, qui représente le graphe d'héritage de catégories, initialisé à $\{(Any_c, Any_c)\}$,
- $OrealC \subseteq \mathbf{O} \times \mathbf{C}$, qui représente le lien de réalisation, initialisé à $\{\}$,
- $DuseD \subseteq \mathbf{D} \times \mathbf{D} \times \mathbb{N}$, qui représente le lien de dépendance de domaines, défini dans 2.2.3.1, initialisé à $\{\}$ (l'entier représente le degré de dépendance, détaillé plus loin),
- $RuseD \subseteq \mathbf{R} \times \mathbf{D}$, qui répertorie les descriptions de relations, initialisé à $\{\}$,
- $RuseC \subseteq \mathbf{R} \times \mathbf{C}$, qui répertorie les compositions de relations, initialisé à $\{(Any_r, Any_c)\}$,
- $CuseD \subseteq \mathbf{C} \times \mathbf{D}$, qui répertorie les descriptions de catégories, initialisé à $\{\}$,
- $CuseC \subseteq \mathbf{C} \times \mathbf{C}$, qui répertorie les compositions de catégories, initialisé à $\{\}$,
- $CuseR \subseteq \mathbf{C} \times \mathbf{R}$, qui répertorie les topologies de catégories, initialisé à $\{\}$,
- $OuseO \subseteq \mathbf{O} \times \mathbf{O}$, qui répertorie les compositions d'objets, initialisé à $\{\}$,
- $OuseR \subseteq \mathbf{O} \times \mathbf{R}$, qui répertorie les topologies d'objets, initialisé à $\{\}$.

Ces ensembles sont définis pour rendre compte, à chaque instant, de la structure de la base représentée par ses quatre ensembles **D**, **R**, **C** et **O**. Ils ont été définis par l'analyse des dépendances entre ces quatre ensembles, selon la structure du modèle (cf. figure 3.1).

L'approche que nous adoptons pour assurer le maintien des contraintes d'intégrité liées au modèle consiste à alimenter ces ensembles à chaque opération sur la base, de manière à pouvoir les utiliser pour vérifier les préconditions d'autres opérations, intervenant ultérieurement.

Prenons l'exemple de l'invariant **IAC** qui statue qu'il n'y a pas de cycle dans la hiérarchie d'héritage entre catégories. Afin de maintenir cet invariant, l'ensemble $CgenC$ est mis à jour à chaque création ou suppression d'une catégorie, et à chaque ajout ou suppression d'une catégorie générique pour une catégorie. Il est, en permanence, le reflet du graphe d'héritage entre catégories. Toute modification de ce graphe doit être consistante vis à vis de l'état courant du graphe.

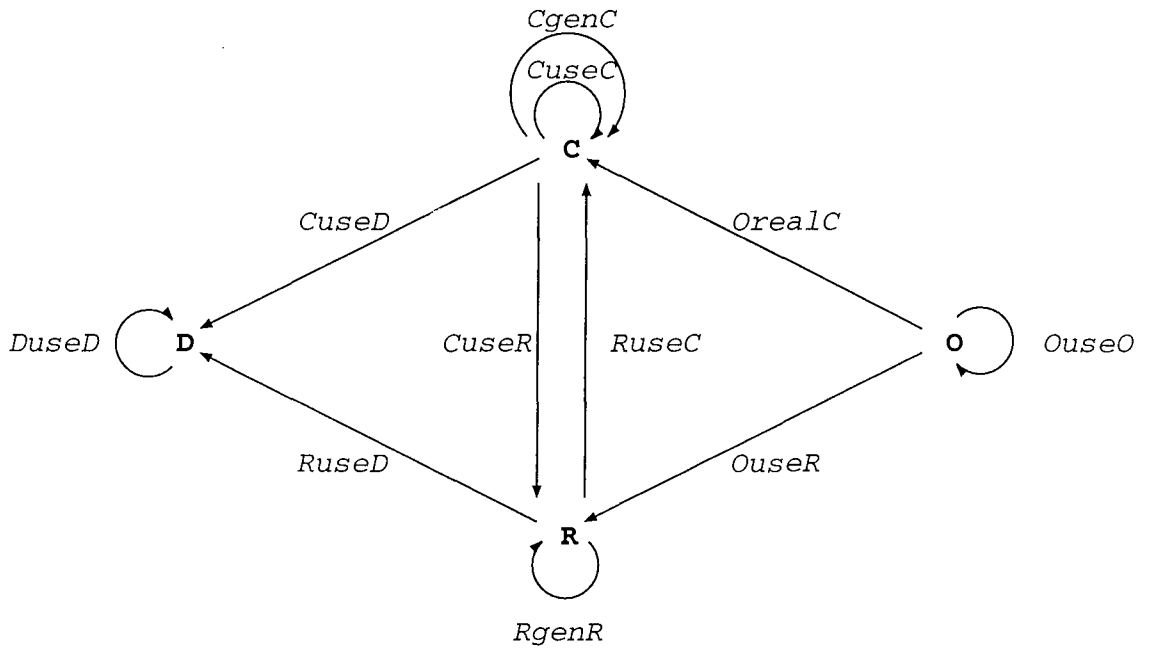


FIG. 3.1 – *Dépendances entre les quatre ensembles fondamentaux d'une base EMIR*

Afin de mieux illustrer le rôle de ces ensembles, nous allons détailler certaines opérations de l'annexe B ; cela nous permettra aussi de préciser certaines notations, qui y sont utilisées.

Exemple 1 : Opération de Création d'une Catégorie (OCC).

Paramètres :
c : string
$c' \in \mathbf{C}$
Préconditions :
$c \notin \mathbf{C}.nom$ ⟵ INC
Actions :
$\mathbf{C} \cup = \{c\}$
$c.description := c'.description$ ⟵ IHCG
pour $a : d \in c'.description$
    $CuseD \cup = \{(c, d)\}$
$c.composition := c'.composition$ ⟵ IHCG
pour $ro : c''(t) \in c'.composition$
    $CuseC \cup = \{(c, c'')\}$
$c.topologie := c'.topologie$ ⟵ IHCG
pour $l : r \in c'.topologie$
    $CuseR \cup = \{(c, r)\}$
$CgenC \cup = \{(c, c)\}$ (initialisation des fonctions de renommage σ_a, σ_c et σ_l)
pour $(c'', c') \in CgenC$
    $CgenC \cup = \{(c'', c)\}$ (initialisation des fonctions de renommage σ_a, σ_c et σ_l)

Les paramètres de l'opération OCC sont une chaîne de caractères c représentant le nom de la catégorie à créer et une catégorie dont la nouvelle catégorie héritera. Toute catégorie est donc reliée à d'autres déjà existantes par le lien de spécialisation.

La seule précondition d'exécution de cette opération concerne l'invariant d'unicité de noms de catégories (INC).

Lorsque cette précondition est vérifiée, une nouvelle catégorie est ajoutée à l'ensemble des catégories (on confond ici la catégorie c avec son nom).

On ajoute la liste d'attributs de la catégorie générique c' à celle de c. Le même traitement est effectué pour la composition et la topologie. Ces trois étapes assurent le respect de l'invariant d'héritage de structure entre catégories (IHCG).

A chacune de ces trois étapes, un ensemble particulier est mis à jour, afin de tenir compte des "utilisations" de domaines induites par la description (ensemble $CuseD$), des utilisations d'autres catégories induites par la composition (ensemble $CuseC$) et des utilisations de relations induites par la topologie (ensemble $CuseR$).

Enfin, on met à jour l'ensemble $CgenC$ représentant le graphe d'héritage de catégories, en reliant c à elle-même dans un premier temps et ensuite à toutes les catégories génériques de c'. A chaque mise à jour, les fonctions de renommages sont spécifiées (cf. 2.2.5.3 et 2.3.3.2).

On peut tout de suite remarquer que les opérations sont spécifiées dans un pseudo-code suffisamment simple à déchiffrer pour nous éviter d'en spécifier la syntaxe ou la sémantique. A chaque fois qu'un invariant entre en jeu, il est mentionné à droite de la ligne en question avec une flèche.

Exemple 2 : Opération de Suppression d'un Objet (OSO).

Paramètres :
$o \in \mathbf{O}$
Préconditions :
Actions :
pour $ro : o'(t) \in o.composition$
    $OuseO -= \{(o, o')\}$
    si $(t = DO) \vee (t = Do)$
        $OSO\,(o')$ ⟵ ICOD
pour $(o', o) \in OuseO$
    pour $ro : o(t) \in o'.composition$
        si $(t = DO) \vee (t = IO)$
            $OSO\,(o')$ ⟵ ICOO
        sinon
            $OuseO -= \{(o', o)\}$
pour $rc : r \in o.topologie$
    $OuseR -= \{(o, r)\}$

$OrealC -= \{(o, *)\}$
$\mathbf{O} -= \{o\}$

La suppression d'un objet o de la base ne nécessite aucune précondition. En revanche, elle entraîne plusieurs actions qui visent à maintenir l'intégrité.

Pour tous les objets composant l'objet à supprimer, on met à jour l'ensemble $OuseO$ d'utilisation d'objets. De plus, si le lien est de type dépendant, l'objet composant est également supprimé, afin de maintenir l'invariant de composition dépendante (ICOD).

Ensuite, pour tous les objets dont l'objet à supprimer fait partie de la composition, si le type de lien est de type obligatoire, on supprime l'objet composé (invariant de composition obligatoire ICOO), sinon, on met simplement à jour l'ensemble $OuseO$.

Les ensembles $OuseR$ et $OrealC$ sont diminués de tout ce qui concerne l'objet à supprimer, avant de détruire réellement l'objet de la base d'objets.

Exemple 3 : Opération de Modification d'un Domaine Agrégé: Ajout d'un Attribut (OMDAAA).

Paramètres :
$d \in \mathbf{D}$
a : string
$d' \in \mathbf{D}$
Préconditions :
$a : * \notin d.description$ ⟵ INAD
$d' \neq d$ ⟵ IAD
$(d', d, *) \notin DuseD$ ⟵ IAD
Actions :
$d.description \cup = \{a : d'\}$
pour $d_h \in \{d\} \cup \{d'' \in \mathbf{D}, (d'', d, *) \in DuseD\}$
  pour $d_b \in \{d'\} \cup \{d'' \in \mathbf{D}, (d', d'', *) \in DuseD\}$
    si $(d_h, d_b, n) \in DuseD$
      $DuseD -= \{(d_h, d_b, n)\} \cup = \{(d_h, d_b, n+1)\}$
    sinon
      $DuseD \cup = \{(d_h, d_b, 1)\}$
pour $(c, d) \in CuseD$
  pour $(o, c) \in OrealC$
    pour $a : d \in c.description$
      $o.description \cup = \{a : defaut(d)\}$ ⟵ IRCG
pour $(r, d) \in RuseD$
  pour $(o, r) \in OuseR$
    pour $rc : r \in o.topologie$
      pour $a : d \in r.description$
        $rc.description \cup = \{a : defaut(d)\}$ ⟵ IRCG

L'opération d'ajout d'un attribut à un domaine agrégé nécessite comme paramètres, le domaine à mettre à jour (d), un nom d'attribut (a) et le nouveau domaine qui lui sera affecté (d').

Une précondition assurant l'invariant d'unicité de noms d'attributs dans un domaine agrégé (INAD) nécessite que le nom du nouvel attribut ne fasse pas déjà partie des noms d'attributs du domaine. Deux autres préconditions assurent l'invariant d'acyclicité dans la définition des domaine agrégés (IAD). Elles stipulent que le domaine associé à l'attribut ne doit pas être le domaine agrégé ou l'un des domaines qui en dépendent, selon la définition donnée en 2.2.3.1.

Cela nous amène à parler de l'ensemble $DuseD$, qui, comme mentionné auparavant, représente le lien de dépendance de domaines défini dans 2.2.3.1.

$(d_h, d_b, n) \in DuseD$ signifie que le domaine d_h dépend n fois du domaine d_b ; n est appelé degré de dépendance entre les deux domaines. Un lien de dépendance existe entre chaque domaine multivalué et son sous-domaine et entre un domaine agrégé et chacun de ses domaines subordonnés. Dans l'exemple de la figure 3.2, le domaine agrégé "Caractéristiques de champignon" dépend deux fois du domaine discret "Couleur".

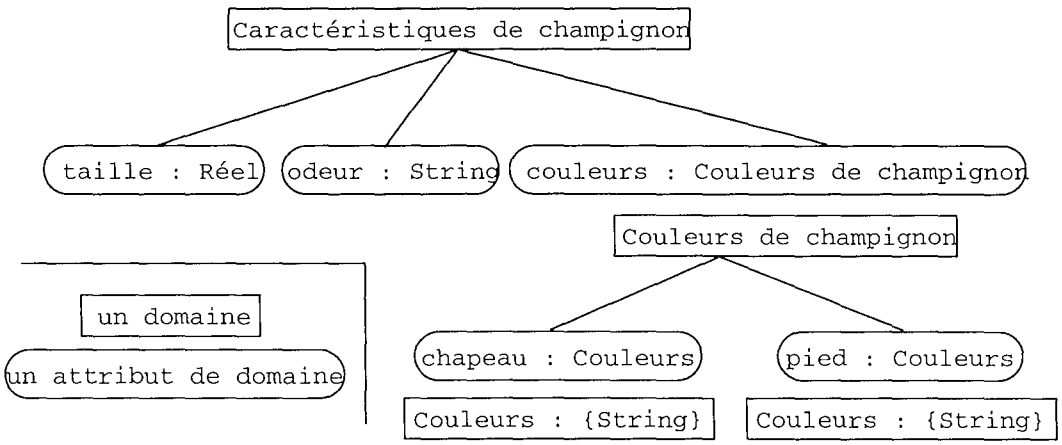


FIG. 3.2 – *Exemple de domaine agrégé*

Ainsi, $(Caractéristiques\ de\ champignon, Couleur, 2) \in DuseD$. Si on supprime l'attribut "pied" du domaine agrégé "Couleurs de champignon", le degré de dépendance ne sera plus que de 1, mais les domaines seront quand même dépendants. Cela permet de gérer l'acyclicité à travers les opérations de mise à jour de domaines.

Revenons aux actions de l'opération OMDAAA. Tout d'abord, la description du domaine agrégé est augmentée du nouvel attribut, avec son domaine associé.

Ensuite, pour chaque domaine d_h qui dépend du domaine agrégé et pour chaque domaine d_b dont dépend le sous domaine ajouté, on augmente de 1 le degré de dépendance.

Enfin, on met à jour la description des objets et des relations concrètes qui utilisent le domaine agrégé afin de maintenir l'invariant général de réalisation de catégories (IRCG). Ici, nous avons choisi de réinitialiser la valeur des attributs correspondants avec la nouvelle valeur par défaut du domaine agrégé, mais on peut aussi garder les anciennes valeurs pour les anciens sous-domaines et initialiser uniquement celle du nouveau.

Exemple 4 : Opération de Modification d'une Relation : Suppression d'un Attribut (OMRSA).

Paramètres :
$r \in \mathbf{R}$
a : string
Préconditions :
$a : d \in r.description$
$pour\ (r', r) \in RgenR$
$si\ r' \neq r$
$a : * \notin r'.description$ (en respectant les renommages) ⟵ IHRG
Actions :
$pour\ (r, r_s) \in RgenR$ (en respectant les renommages)
$r_s.description - = \{a : d\}$ ⟵ IHRG
$si * : d \notin r_s.description$
$RuseD - = \{(r_s, d)\}$
$pour\ (o, r_s) \in OuseR$
$pour\ rc : r_s \in o.topologie$
$rc.description - = \{a : *\}$ ⟵ IRCG

Lorsqu'on veut supprimer un attribut a d'une relation r, il ne faut pas qu'il provienne d'une relation générique, car cela violerait l'invariant général d'héritage de relations (IHRG).

Si cette précondition est vérifiée, on supprime l'attribut de la description de toutes les relations spécifiques de la relation r, afin de respecter ce même invariant.

Si ces relations n'utilisent plus le domaine de l'attribut supprimé, on met à jour l'ensemble $RuseD$ qui répertorie les utilisations de domaines dans des relations.

Enfin, on supprime l'attribut dans les descriptions de toutes les relations concrètes qui instancient les relations spécifiques de r. Ces relations concrètes se trouvent dans des objets utilisant ces relations, repérés dans l'ensemble $OuseR$.

Exemple 5 : Opération de Modification d'un Objet : Modification de l'Objet d'un Composant (OMOMOC).

Paramètres :
$o \in \mathbf{O}$
ro : string
$o' \in \mathbf{O}$
Préconditions :

$\exists c \in \mathbf{C}$, $((o,c) \in OrealC) \wedge (ro : c'(t) \in c.composition) \wedge ((o',c') \in OrealC)$ ⟵ IRCC
$ro' : o'(t') \in o.composition$
Actions :
$o.composition -= \{ro : o''(t'')\} \cup= \{ro : o'(tO), "vide" : o''(t'')\}$ ⟵ IRCG
(le nouveau type de lien tO est obtenu en ajoutant "O" (obligatoire) au type de lien t)
pour $(o,c) \in OrealC$
  pour $l : r \in c.topologie$
    si $ro'' : ro \in l$
      $OMOMCL(o, l, ro'', o')$ ⟵ IRCG

Cette opération s'applique à un objet o et permet de remplacer l'objet relié à un composant de rôle ro (provenant d'une catégorie que réalise o) par l'objet o'.

Une première précondition précise la provenance du composant et vérifie que l'objet de remplacement est admissible pour jouer le rôle ro dans la composition de l'objet o, en ce sens qu'il réalise la catégorie requise pour ce composant. Cela permet d'assurer l'invariant de réalisation de catégories qui concerne la partie composition (IRCC).

Une seconde précondition stipule que l'objet de remplacement doit déjà appartenir à la composition de l'objet o.

Une fois ces préconditions vérifiées, on met à jour la composition de l'objet o en faisant jouer au composant de remplacement le rôle ro et en associant le rôle *vide* (cf. 2.2.6.1) à l'ancien composant de rôle ro. Le type de lien associé au nouveau composant de rôle ro est obtenu par adjonction du type obligatoire au type existant dans la catégorie.

L'invariant général de réalisation de catégories (IRCG) est assuré aussi par la mise à jour de toutes les relations concrètes de l'objet o, qui proviennent de liens topologiques de catégories, faisant intervenir le composant de rôle ro. Ces mises à jours sont effectuées par l'opération de modification de composant de relation concrète, dans un objet (OMOMCL).

Nous avons montré, à la lumière de quelques exemples (dont la version exhaustive se trouve dans l'annexe B), la manière d'assurer les invariants du modèle énoncés dans 3.1.1.1.

Cette méthode s'appuie sur la définition de quelques ensembles, faisant office de dictionnaires de données, initialisés au début de chaque application et alimentés à chaque opération élémentaire d'accès à la base correspondante.

Cette approche peut paraître gourmande en temps, puisqu'elle nécessite des traitements à la moindre opération sur la base. Cependant, nous sommes partis du principe : "*Qui peut le plus, peut le moins !*". Une approche plus relâchée peut être définie plus simplement ; par exemple, avec la notion de transaction, l'utilisateur peut définir des points

d'ancrage pour la vérification de la cohérence.

Nous devons aussi signaler que nous avons fixé un certain nombre de règles empiriques nous permettant d'assurer le respect des invariants dans certains cas particuliers. Ces règles représentent des choix arbitraires que nous nous sommes fixés et peuvent être transgressées dans une nouvelle approche de maintien d'intégrité dans notre modèle :

- réinitialisation des valeurs d'attributs agrégés dont la description a changé (ajout, modification de domaine ou suppression de sous-attribut), dans les objets et les relations concrètes ; nous l'avons vu plus haut dans l'exemple de l'opération OMDAAA, mais cela concerne aussi les opérations OMDAMA et OMDASA (cf. annexe B) ;

- lors de l'ajout ou de la modification d'un composant de relation, tous les liens topologiques qui utilisent cette relation sont supprimés des topologies de catégories et d'objets. Ceci est dû au fait qu'on ne peut assurer l'existence d'un composant (de la catégorie ou de l'objet) susceptible de jouer le nouveau rôle dans la relation. Cela concerne les opérations OMRAC et OMRMC (cf. annexe B).

3.1.2 Intégrité sémantique liée aux applications

Dans cette section, nous présentons dans 3.1.2.1, un langage de spécification de contraintes d'intégrité liées à une application particulière. La sémantique de ce langage est décrite dans 3.1.2.2, sous forme d'une technique d'évaluation de contraintes sur la base d'objets de l'application concernée.

Certains exemples servant à illustrer la définition et l'évaluation de ces contraintes sont tirés d'une application architecturale du modèle, détaillée dans la section 4.3. Les autres exemples proviennent du chapitre 2.

Nous utiliserons en particulier la catégorie *Pièce_rectangulaire* ainsi que la relation *Perpendiculaire_à*, définies dans 2.3, ainsi qu'une catégorie *Mur*, décrite dans la figure 3.3.

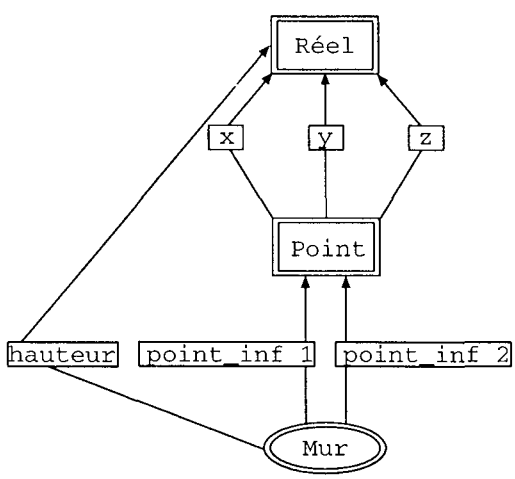


FIG. 3.3 – *La catégorie Mur, issue de l'application architecturale*

Nous utiliserons également les objets pr_1 (décrit ci-après), m_1, m_2, m_3 et m_4 (décrits dans la figure 3.4).

$A(pr_1) = \langle hauteur : 3.0,\ surface : 24 \rangle$
$C(pr_1) = \langle mur_1 : m_1,\ mur_2 : m_2,\ mur_3 : m_3,\ mur_4 : m_4,$
$plancher : ph_1,\ plafond : ph_2 \rangle$
$L(pr_1) = \langle perp_1 : Perpendiculaire\_à,\ perp_2 : Perpendiculaire\_à,$
$perp_3 : Perpendiculaire\_à,\ perp_4 : Perpendiculaire\_à,$
$support_1 : Support\_de\_mur,\ support_2 : Support\_de\_mur,$
$support_3 : Support\_de\_mur,\ support_4 : Support\_de\_mur,$
$para_1 : Parallèle\_à,\ para_2 : Parallèle\_à \rangle$

où le lien $para_1$, par exemple, est défini par : $C(para_1) = \langle m_1 : m_1,\ m_2 : m_3 \rangle$.

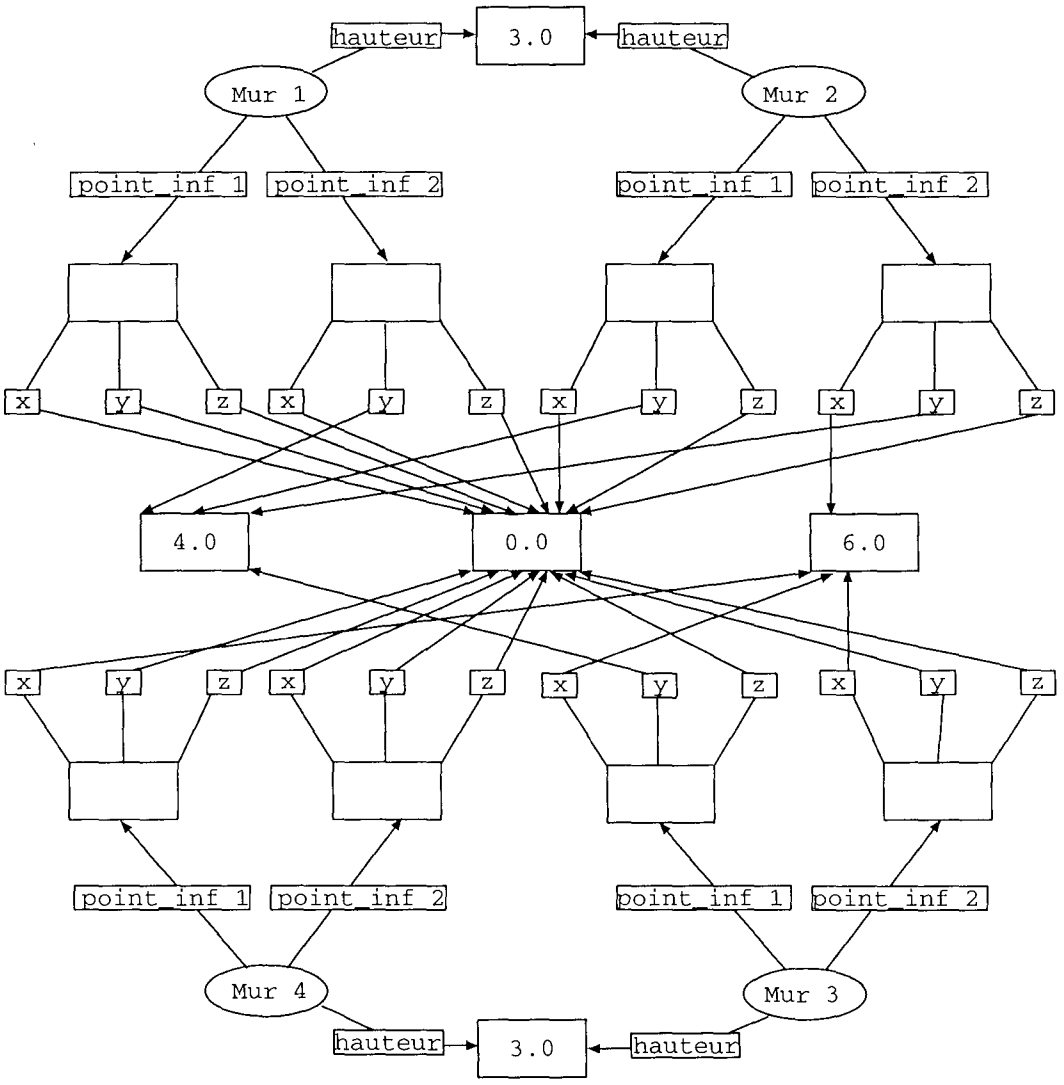


FIG. 3.4 – *Quelques objets (murs) issus de l'application architecturale*

3.1.2.1 Spécification de contraintes

La spécification des contraintes d'intégrité liées aux applications se fait par l'intermédiaire d'assertions. Il s'agit de prédicats spécifiées selon un formalisme que nous présentons dans les paragraphes suivants.

Ces assertions peuvent être liées à des catégories, à des relations ou à des objets particuliers de l'application. Lorsqu'une assertion est liée à une catégorie, elle doit être vérifiée par tous les objets réalisant cette catégorie ou l'une de ses catégories spécifiques. Lorsqu'une assertion est liée à un objet particulier, elle doit être vérifiée par cet objet. Enfin, lorsqu'une assertion est liée à une relation, elle doit être vérifiée par tous les objets qui possèdent dans leur topologie une relation concrète instanciant cette relation ou l'une de ses relations spécifiques.

Pour spécifier les contraintes d'intégrité, nous avons besoin de définir auparavant les notions de chemin, de validité et de destination (fonction *Dest*) de chemin pour un domaine complexe (multivalué ou agrégé), une catégorie, une relation, une valeur d'un domaine complexe, un objet et une relation concrète.

Un *chemin* est une liste de caractéristiques d'un domaine agrégé, d'une catégorie, d'un objet ou d'une relation (attributs, composants, liens) séparés par des points ; par exemple : *adresse.ville* ou *photographe.âge*. Comme nous considérons qu'un composant d'objet avec un rôle *vide* est aussi une caractéristique de cet objet, nous autoriserons un chemin à contenir le nom d'un objet composant d'un autre (cf. 3.1.2.1.4).

Un chemin n'étant pas forcément cohérent, nous définissons la notion de validité d'un chemin pour un domaine complexe, une valeur de ce domaine, une catégorie, un objet, une relation et une relation concrète.

3.1.2.1.1 Validité et destination d'un chemin pour un domaine Ces notions sont définies uniquement pour les domaines multivalués et agrégés.

Pour un domaine agrégé, deux cas se présentent :

- un chemin de longueur 1 : $p = a$, est valide pour un domaine agrégé $d : [a_1 : d_1, ..., a_n : d_n]$ si et seulement si : $\exists 1 \leq i \leq n, a_i = a$. On a alors : $Dest(d, p) = d_i$,
- un chemin $p = a_1...a_q\ (q > 1)$, est valide pour un domaine agrégé d, si et seulement si $p' = a_1...a_{q-1}$ est valide pour d et $p'' = a_q$ est valide pour $Dest(d, p')$. On a alors : $Dest(d, p) = Dest(Dest(d, p'), p'')$.

Pour un domaine multivalué $d : \{d'\}$, un chemin p est valide si et seulement si il l'est pour d' et on a un nouveau domaine multivalué $Dest(d, p) : \{Dest(d', p)\}$.

Exemple :
(on reprend le domaine agrégé *Adresse* défini dans 2.2.3.1)

$Dest(Adresse, localité.ville) = Dest(Dest(Adresse, localité), ville)$

$= Dest(Localité, ville)$
$= String.$

3.1.2.1.2 Validité et destination d'un chemin pour une valeur Un chemin p est valide pour une valeur $v \in V(d)$ où d est un domaine agrégé ou multivalué, si et seulement si p est valide pour d.

Pour une valeur agrégée, on a :

- chemins de longueur 1 : $Dest(v, a) = v.a$: valeur correspondant à l'attribut a dans v.
- chemins de longueur $q > 1$: $Dest(v, a_1...a_q) = Dest(Dest(v, a_1...a_{q-1}), a_q)$.

Pour une valeur ensemble $Dest(\{v_1, ..., v_n\}, p) = \{Dest(v_1, p), ..., Dest(v_n, p)\}$.

3.1.2.1.3 Validité et destination d'un chemin pour une catégorie

- Catégories simples :
  - un chemin de longueur 1 : $p = a$, est valide pour une catégorie $c = (\langle a_1 : d_1, ..., a_n : d_n\rangle, \langle ro_1 : c_1, ..., ro_m : c_m\rangle, \langle l_1 : r_1, ..., l_p : r_p\rangle)$ si et seulement si l'une des conditions suivantes est vérifiée :
    - $\exists 1 \leq i \leq n, a_i = a$; on a alors : $Dest(c, p) = d_i$,
    - $\exists 1 \leq j \leq m, ro_j = a$; on a alors : $Dest(c, p) = c_j$,
    - $\exists 1 \leq k \leq p, l_k = a$; on a alors : $Dest(c, p) = r_k$,
  - un chemin $p = a_1...a_q$ $(q > 1)$, est valide pour une catégorie c, si et seulement si $p' = a_1...a_{q-1}$ est valide pour c et $p'' = a_q$ est valide pour $Dest(c, p')$. On a alors : $Dest(c, p) = Dest(Dest(c, p'), p'')$.

  Remarque : Pour rendre la définition de la destination d'un chemin uniforme pour les catégories ensembles également, nous imposons à un chemin de ne pas aboutir à une relation, même si le "passage" par une relation est autorisé (cf. 3.1.2.1.5). La destination d'un chemin à partir d'une catégorie est donc soit un domaine, soit une autre catégorie.

- Catégories ensembles :
  un chemin est valide pour c^* si et seulement si il est valide pour c : on a alors :
  - $Dest(c^*, p) = (Dest(c, p))^*$, si $Dest(c, p)$ est une catégorie,
  - $Dest(c^*, p) = \{Dest(c, p)\}$, si $Dest(c, p)$ est un domaine.

Exemple :

$$\begin{aligned} Dest(Image, date.mois) &= Dest(Dest(Image, date), mois) \\ &= Dest(Date, mois) \\ &= Mois : I/[1, 12]. \end{aligned}$$

3.1.2.1.4 Validité et destination d'un chemin pour un objet D'abord, un chemin p ne contenant pas de nom d'objets, est valide pour un objet o, si et seulement si $\exists c \in \mathbf{C}$, $o \leftarrow c$ et p valide pour c.

Des chemins particuliers sont définis pour les objets. Ils commencent par des noms d'objets : $p = o_1...o_n.a_1...a_m$. Ils seront utilisés pour imposer des contraintes sur la composition d'un objet par un autre, quand cela ne provient pas d'une catégorie que l'objet composé réalise (ce qui explique l'utilisation du nom de l'objet composant dans le chemin, au lieu du rôle qui peut être *vide*).

Par exemple, si on désire imposer une contrainte sur un objet o', *en sa qualité de composant d'un objet o, et non dans l'absolu*, on attachera la contrainte à l'objet composé o (et non au composant o'), en faisant précéder le chemin utilisé dans la contrainte par (le nom de) o'. Le chemin sera valide pour o, si et seulement si le reste du chemin obtenu en enlevant o' est valide pour o' ; ce sous-chemin pouvant à son tour commencer par un composant de o'.

Ce type de chemin permet d'exploiter la capacité du modèle en ce qui concerne la liberté de structure des objets par rapport à leurs catégories. Cette propriété sera discutée plus longuement dans la partie 3.1.2.3.

A présent, nous définissons la notion de destination d'un chemin pour un objet.

- Objets simples :
  - chemins de longueur 1 : $Dest(o, a) = o.a$: valeur, objet ou relation concrète correspondant à la caractéristique a dans $s(o)$; si a est le nom d'un composant o', alors $Dest(o, o') = o'$.
  - chemins de longueur $q > 1$: $Dest(o, a_1...a_q) = Dest(Dest(o, a_1...a_{q-1}), a_q)$.

  Remarque : D'après la remarque précédente sur les destinations de chemins pour les catégories, la destination d'un chemin à partir d'un objet est soit une valeur, soit un autre objet, même si le chemin peut "passer" par une relation concrète (cf. 3.1.2.1.6).

- Objets ensembles :
  - $Dest(o, a) = o'$ tel que : $s(o') = \{Dest(o_i, a), o_i \in s(o)\}$, si le chemin mène à des objets.

    Remarque : Dans ce cas, l'évaluation de la destination conduit à la création d'un nouvel objet ensemble non existant auparavant dans la base.
  - $Dest(o, a) = \{Dest(o_i, a), o_i \in s(o)\}$, si le chemin mène à des valeurs.

Exemples :

$$\begin{aligned} Dest(o_1, photographe.nom) &= Dest(Dest(o_1, photographe), nom) \\ &= Dest(o_2, nom) \\ &= "Albert". \end{aligned}$$

$$\begin{aligned} Dest(o_1, o_4.\hat{a}ge) &= Dest(Dest(o_1, o_4), \hat{a}ge) \\ &= Dest(o_4, \hat{a}ge) \\ &= 45. \end{aligned}$$

3.1.2.1.5 Validité et destination d'un chemin pour une relation

- un chemin de longueur 1 : $p = a$, est valide pour une relation $r = (\langle ro_1 : c_1, ..., ro_n : c_n \rangle, \langle a_1 : d_1, ..., a_m : d_m \rangle)$ si et seulement si l'une des conditions suivantes est vérifiée :
  - $\exists 1 \leq i \leq n, ro_i = a$; on a alors : $Dest(r, p) = c_i$,
  - $\exists 1 \leq j \leq m, a_j = a$; on a alors : $Dest(r, p) = d_j$,
- un chemin $p = a_1...a_q (q > 1)$, est valide pour une relation r, si et seulement si $p' = a_1...a_{q-1}$ est valide pour r et $p'' = a_q$ est valide pour $Dest(r, p')$. On a alors : $Dest(r, p) = Dest(Dest(r, p'), p'')$.

Exemple :

$$\begin{aligned} Dest(Support\_de\_mur, en\_bas.hauteur) &= Dest(Dest(Support\_de\_mur, en\_bas), hauteur) \\ &= Dest(Mur, hauteur) \\ &= Réel. \end{aligned}$$

3.1.2.1.6 Validité et destination d'un chemin pour une relation concrète Un chemin p est valide pour une relation concrète $rc : r$, si et seulement si p est valide pour r ; on a alors :

- chemins de longueur 1 : $Dest(rc, a) = rc.a$: valeur ou objet correspondant à la caractéristique a dans $C(rc)$ ou $A(rc)$.
- chemins de longueur $q > 1$: $Dest(rc, a_1...a_q) = Dest(Dest(rc, a_1...a_{q-1}), a_q)$.

Exemple :

$$\begin{aligned} Dest(lien_1, gauche.prénom) &= Dest(Dest(lien_1, gauche), prénom) \\ &= Dest(o_3, prénom) \\ &= "Jean". \end{aligned}$$

Remarque :
Dans tous les cas de figure, la destination d'un chemin est :

- soit une catégorie, soit un domaine (si l'origine est une catégorie ou une relation)
- soit un objet, soit une valeur (si l'origine est un objet ou une relation concrète).

Dans la suite, nous allons définir successivement les notions d'*expression*, d'*équation* et d'*assertion*, qui constituent les bases de notre modèle de contraintes. Dans le cas général, une contrainte sera attachée à une catégorie (elle devra être respectée par tous les objets réalisant la catégorie), à une relation (elle devra être respectée par toutes les relations concrètes instanciant la relation) ou à un objet particulier (elle devra être respectée par cet objet).

3.1.2.1.7 Expressions Nous allons définir en parallèle la notion d'expression et de type d'une expression. Pour cela, nous utiliserons certaines notions définies dans le chapitre 2, notamment concernant les notions de types de valeurs et d'objets.

Etant donné une expression ε, nous lui associons un type $[\![\varepsilon]\!]$, élément de $\mathbf{Tv} \cup \mathbf{To}$.

- Les valeurs (éléments de **V**) sont des expressions *pour tous les objets, catégories et relations.* Leurs types sont définis comme dans 2.2.1.1.

  Exemples :
  5, "*toto*", [*jour* : 1, *mois* : 3, *année* : 1995], ...

- Les objets (éléments de **O**), désignés par leurs noms, sont des expressions *pour tous les objets, catégories et relations.* Leurs types sont définis comme dans 2.2.6.2.

  Exemples :
  m_1, o_2, ...

- Si $o \in \mathbf{O}$ est un objet, et si p est un chemin valide pour cet objet, alors $o.p$ est une expression *pour tous les objets, catégories et relations.* Son type est $[\![Dest(o,p)]\!]$, qui est bien un élément de $\mathbf{Tv} \cup \mathbf{To}$, d'après la remarque ci-dessus.

  Exemples :
  $o_1.photographe.nom, m_1.hauteur$, ...

- Tout chemin p valide pour une catégorie c est une expression *pour cette catégorie.* Son type est $[\![Dest(c,p)]\!]$, qui est bien un élément de $\mathbf{Tv} \cup \mathbf{To}$, d'après la remarque ci-dessus.

  Tout chemin p valide pour une relation r est une expression *pour cette relation.* Son type est $[\![Dest(r,p)]\!]$, qui est bien un élément de $\mathbf{Tv} \cup \mathbf{To}$, d'après la remarque ci-dessus.

  Tout chemin p valide pour un objet o est une expression *pour cet objet.* Son type est $[\![Dest(o,p)]\!]$, qui est bien un élément de $\mathbf{Tv} \cup \mathbf{To}$, d'après la remarque ci-dessus.

  Exemples :
  Le chemin *hauteur* pour la catégorie *Pièce_rectangulaire.*
  Le chemin *en_bas.hauteur* pour la relation *Parallèle_à.*
  Le chemin *hauteur* pour l'objet mur_2.

- Si ε_1, ..., ε_n sont des expressions (pour une catégorie, une relation ou un objet) dont les types sont des éléments de **Tv** (et non pas **To**), et si φ est une fonction définie de $[\![\varepsilon_1]\!] \times ... \times [\![\varepsilon_n]\!]$ vers un type de valeurs tv, alors $\varphi(\varepsilon_1, ..., \varepsilon_n)$ est aussi une expression pour l'entité en question, dont le type est tv.

  Cette notation regroupe toutes les fonctions arithmétiques (addition soustraction, ..., sinus, logarithme, puissance, ...), les fonctions sur les chaînes de caractères (concaténation, ...), ...

  Exemples :
  $sup(point\_inf_1.y, point\_inf_2.y)$ est une expression valide pour la catégorie *Mur*. Le

type de cette expression est R (nombres réels, cf. 2.2.1.1).
D'autre part, si une catégorie c a un attribut a dont le domaine est de type R, alors les expressions suivantes sont valables pour c: $a + log(a) - 1$, $sin(a)$ et $E(a)$, représentant la partie entière de a. Leurs types respectifs sont R, R et I (entiers naturels, cf. 2.2.1.1).

- Si ε est une expression (pour une catégorie, une relation ou un objet) telle que $[\![\varepsilon]\!] = tv^*$, $tv \in \mathbf{Tv}$ ou $[\![\varepsilon]\!] = to^*$, $to \in \mathbf{To}$, alors $card(\varepsilon)$ est aussi une expression pour l'entité en question, dont le type est I.

  Exemples :
  $card(caractéristiques)$ est une expression pour la catégorie *Image*, définie dans 2.2.5.1.

3.1.2.1.8 Equations Une équation est en général formée de deux expressions séparées par un opérateur dont l'évaluation donne une valeur Booléenne ; cet opérateur peut être l'égalité, l'inclusion, la supériorité ..., ou leur négation.

Dans ce qui suit, à chaque fois qu'une équation utilise des expressions, elle est valable pour une catégorie si toutes les expressions le sont ; idem pour une relation ou un objet.

- Si ε_1 et ε_2 sont des expressions telles que $[\![\varepsilon_1]\!] = [\![\varepsilon_2]\!]$ ou encore $[\![\varepsilon_1]\!]$, $[\![\varepsilon_2]\!] \in \mathbf{To}$, sont des types d'objets comparables, au sens de l'ordre partiel sur les types d'objets (cf. 2.2.4.2), alors $\varepsilon_1 = \varepsilon_2$ et $\varepsilon_1 \neq \varepsilon_2$ sont des équations.

  Exemples :
  $hauteur = 15.0$ pour l'objet m_3.
  $distance \neq 20.0$ pour la relation *Parallèle_à*.

- Si ε_1 et ε_2 sont des expressions telles que $[\![\varepsilon_1]\!]$ et $[\![\varepsilon_2]\!]$ sont des types de valeurs munis d'un ordre partiel (I, R ou S), alors $\varepsilon_1 \, op \, \varepsilon_2$ est une équation, avec $op \in \{<, >, \leq, \geq\}$.

  Exemples :
  $distance \leq 20.0$ pour la relation *Parallèle_à*.

- Si ε_1 et ε_2 sont des expressions telles que $[\![\varepsilon_1]\!]$, $[\![\varepsilon_2]\!] \in \mathbf{Tv}$, avec $[\![\varepsilon_1]\!] = [\![\varepsilon_2]\!]^*$, ou encore $[\![\varepsilon_1]\!]$, $[\![\varepsilon_2]\!] \in \mathbf{To}$, avec $[\![\varepsilon_1]\!] = [\![\varepsilon_2]\!]^*$, alors $\varepsilon_2 \in \varepsilon_1$ et $\varepsilon_2 \notin \varepsilon_1$ sont des équations.

  Exemples :
  "*paysage*" $\in$ *caractéristiques* pour la catégorie *Image*.

- Si ε_1 et ε_2 sont des expressions telles que $[\![\varepsilon_1]\!]$ et $[\![\varepsilon_2]\!]$ sont des types ensembles (de valeurs ou d'objets), alors $\varepsilon_1 \subset \varepsilon_2$ et $\varepsilon_1 \not\subset \varepsilon_2$ sont des équations.

  Exemples :
  {"*paysage*", "*apaisant*"} $\subset$ *caractéristiques* pour la catégorie *Image*.

3.1.2.1.9 Assertions Les assertions (le mot est emprunté à [CRZNM88]) sont les primitives principales de la modélisation des contraintes d'intégrité dans les applications EMIR. Elles sont basées sur des combinaisons logiques d'équations.

Dans ce qui suit, d'une manière analogue à ce qui précède, à chaque fois qu'une assertion utilise des équations, elle est valable pour une catégorie si toutes les équations le sont ; idem pour une relation ou un objet.

- Toute équation est une assertion.
- Si Ξ est une assertion, alors $\neg\Xi$ l'est aussi.

  Exemples :
  $\neg(\text{"}paysage\text{"} \in caractéristiques)$ est une assertion pour la catégorie *Image*.
- Si Ξ_1 et Ξ_2 sont des assertions, alors $\Xi_1 \vee \Xi_2$, $\Xi_1 \wedge \Xi_2$, $\Xi_1 \Rightarrow \Xi_2$ et $\Xi_1 \Leftrightarrow \Xi_2$ sont des assertions.

  Exemples :
  $(photographe.nom = \text{"}Albert\text{"}) \Rightarrow (\text{"}paysage\text{"} \in caractéristiques)$
  est une assertion pour la catégorie *Image*.

Nous avons défini un langage formel pour exprimer ces contraintes d'intégrité liées aux applications. Ce travail a fait l'objet, en partie, d'une thèse CNAM [ISS95].

3.1.2.2 Evaluation des contraintes

L'évaluation des contraintes d'intégrité se fait par rapport à un objet de la base, et éventuellement une des relations concrètes de sa topologie (pour les contraintes sur des relations).

Il s'agit à ce stade de donner une sémantique formelle au "langage" de contraintes défini plus haut. Le but est de définir une méthode calculatoire qui, pour un objet de la base (resp. pour une relation concrète) et une contrainte donnée valable pour cet objet ou pour une catégorie qu'il réalise (resp. valable pour une relation qu'elle instancie), indique si la contrainte est satisfaite (évaluée à $VRAI$) ou violée (évaluée à $FAUX$).

Afin d'avoir une terminologie uniforme, nous notons $eval(\rho, o, rc)$ l'évaluation d'une partie ρ d'une contrainte sur un objet o et une des relations concrètes de sa topologie rc. S'il n'y a pas besoin de la relation concrète, nous noterons $eval(\rho, o, -)$.

La fonction d'évaluation *eval* est une fonction à valeurs booléennes.

3.1.2.2.1 Evaluation d'expressions Soient ε une expression, $o \in \mathbf{O}$ un objet et $rc : r$ un élément de $L(o)$.

- Si $\varepsilon = v \in \mathbf{V}$, alors $eval(v, o, rc) = v$.

  Exemples :
  $eval(5, o, rc) = 5$.
  $eval(\text{"}toto\text{"}, o, rc) = \text{"}toto\text{"}$.

- Si $\varepsilon = o' \in \mathbf{O}$, alors $eval(o', o, rc) = o'$.

  Exemples :
  $eval(m_1, pr_1, -) = m_1$.

- Si $\varepsilon = o'.p$, avec $o' \in \mathbf{O}$ et p chemin valide pour o', alors $eval(o'.p, o, rc) = Dest(o', p)$.

  Exemples :
  $eval(m_1.hauteur, pr_1, -) = Dest(m_1, hauteur) = 3.0$.

- Si $\varepsilon = p$, avec p chemin valide pour l'objet o ou pour une catégorie que réalise o, alors $eval(p, o, rc) = Dest(o, p)$.

  Si $\varepsilon = p$, avec p chemin valide pour une relation qu'instancie rc, alors $eval(p, o, rc) = Dest(rc, p)$.

  Exemples :
  $eval(hauteur, pr_1, -) = Dest(pr_1, hauteur) = 3.0$.

$$\begin{aligned} eval(m_2.point\_inf_2.y, pr_1, para_1) &= Dest(para_1, m_2.point\_inf_2.y) \\ &= Dest(Dest(para_1, m_2), point\_inf_2.y) \\ &= Dest(m_3, point\_inf_2.y) \\ &= Dest(Dest(m_3, point\_inf_2), y) \\ &= Dest([x : 6.0,\ y : 0.0,\ z : 0.0], y) \\ &= 0.0. \end{aligned}$$

- Si $\varepsilon = \varphi(\varepsilon_1, ..., \varepsilon_n)$, alors $eval(\varepsilon, o, rc) = \varphi(eval(\varepsilon_1, o, rc), ..., eval(\varepsilon_n, o, rc))$.

  Exemples :
  $eval(sup(point\_inf_1.y, point\_inf_2.y), m_1, -)$
  $= sup(eval(point\_inf_1.y, m_1, -), eval(point\_inf_2.y, m_1, -))$
  $= sup(Dest(m_1, point\_inf_1.y), Dest(m_1, point\_inf_2.y))$
  $= sup(0.0, 4.0)$
  $= 4.0$

  Remarque :
  Il est possible que l'évaluation d'une telle fonction au niveau d'un objet ou d'une relation concrète, conduise à une erreur (division par zéro, logarithme d'une quantité négative, ...), même si l'expression est valable à un niveau conceptuel. Dans ce cas la contrainte en cours d'évaluation est automatiquement violée.

  Exemple :
  $m_2.point\_inf_1.x / m_4.point\_inf_2.x$

- Si $\varepsilon = card(\varepsilon')$ alors $eval(\varepsilon, o, rc) = card(eval(\varepsilon', o, rc))$

  Exemples :
  Si on reprend l'exemple de l'objet o_1 défini dans 2.2.6.1, et réalisant la catégorie

Image, on a :

$$\begin{aligned} eval(card(caractéristiques), o_1, -) &= card(eval(caractéristiques, o_1, -)) \\ &= card(\{"paysage", "Noir\&Blanc"\}) \\ &= 2. \end{aligned}$$

3.1.2.2.2 Evaluation d'équations Soient ε_1 et ε_2 deux expressions, $o \in \mathbf{O}$ un objet et $rc : r$ un élément de $L(o)$.

D'après 3.1.2.1.8, une équation est de la forme : $\varepsilon_1 \, op \, \varepsilon_2$, où op est un opérateur de l'ensemble $\{=, \neq, \in, \notin, \subset, \not\subset, <, >, \leq, \geq\}$.

$$eval(\varepsilon_1 \, op \, \varepsilon_2, o, rc) = eval(\varepsilon_1, o, rc) \; op \; eval(\varepsilon_2, o, rc).$$

Exemples :

$$\begin{aligned} eval("paysage" \in caractéristiques, o_1, -) &= "paysage" \in eval(caractéristiques, o_1, -) \\ &= "paysage" \in \{"paysage", "Noir\&Blanc"\} \\ &= VRAI \end{aligned}$$

3.1.2.2.3 Evaluation d'assertions

- $eval(\neg \Xi, o, rc) = \neg eval(\Xi, o, rc)$.

  Exemples :
  $eval(\neg("paysage" \in caractéristiques), o_1, -) = FAUX.$

- Si $lop \in \{\vee, \wedge, \Rightarrow, \Leftrightarrow\}$, alors $eval(\Xi_1 \, lop \, \Xi_2, o, rc) = eval(\Xi_1, o, rc) \; lop \; eval(\Xi_2, o, rc)$.

  Exemples :
  $eval((photographe.nom = "Albert") \Rightarrow ("paysage" \in caractéristiques), o_1, -) = VRAI.$
  En effet, on a les deux résultats suivants :
  $eval((photographe.nom = "Albert"), o_1, -) = VRAI$
  $eval("paysage" \in caractéristiques, o_1, -) = VRAI.$

3.1.2.3 Discussion

Nous avons présenté un langage de définition de contraintes d'intégrité sur une base de données EMIR, ainsi qu'une technique d'évaluation de ces contraintes.

Ce langage permet d'attacher des contraintes soit à des catégories, soit à des relations, ou encore à des objets. On peut donc définir la cohérence d'une base EMIR à différents niveaux, qui font intervenir divers éléments de la description de la base.

En particulier, des contraintes exploitant la flexibilité du lien de réalisation ont été définies par l'intermédiaire de chemins utilisant des noms d'objets les uns à la suite des

autres. Ce genre de contraintes est particulièrement utile pour définir des règles de cohérence ponctuelles sur certains objets.

Par exemple, dans le cadre de l'application architecturale décrite dans la partie 4.3, lors de la conception d'un projet architectural, faisant intervenir, dans une construction, des murs en tant que composants (non définis dans la catégorie représentant les constructions), le concepteur peut être amené à positionner ou déplacer un mur d'une manière erronée par rapport à certains critères de construction.

Considérons la figure 3.5, représentant une construction avec des murs de façades $(m_1, ..., m_{10})$, et des cloisons intérieures (c_1 et c_2), et supposons que cette construction est représentée par l'objet $cons_1$ réalisant une catégorie *Construction* et composé des objets $m_1, ..., m_{10}$ (matérialisant les murs externes de la construction) et des objets c_1 et c_2 (matérialisant les cloisons internes). Comme on ne sait pas *a priori* de quoi est composée une construction en général, tous ces composants sont individuels à l'objet $cons_1$, donc de rôle *vide*.

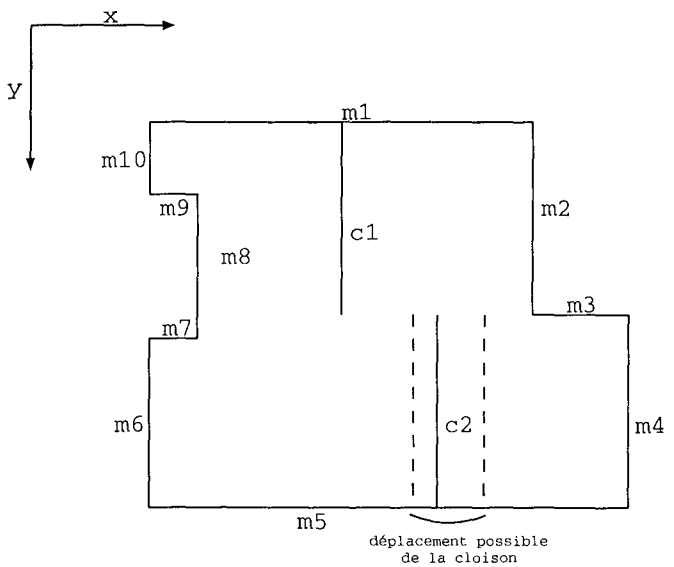


FIG. 3.5 – *Un projet de construction dans l'application architecturale*

Supposons qu'on désire qu'au cours de tout le processus de conception, la pièce délimitée par m_3, m_4 et c_2 ait une largeur suffisante (supérieure à un seuil). On ne peut définir ce critère au niveau de la catégorie *Construction*, puisqu'il fait intervenir des composants individuels de l'objet $cons_1$, non prévus dans la catégorie.

Notre langage de définition de contraintes permet de saisir cette contrainte par une assertion, attachée à l'objet $cons_1$:

$$m_4.point\_inf_1.x - c_2.point\_inf_1.x \geq 5.0$$

qui est bien une assertion valide pour l'objet $cons_1$, d'après les définitions données dans 3.1.2.1.

De cette manière, on oblige les murs m_4 et c_2 à rester à une distance raisonnable au cours de tout le processus de conception. S'ils sont trop rapprochés, la contrainte serait violée.

Cette facette du langage de contraintes permet d'attacher des critères de cohérence à un objet, en utilisant non seulement ses composants obligatoires (avec des chemins valides pour sa catégorie) mais aussi ses composants individuels.
Il ne s'agit pas là d'une originalité du langage de définition de contraintes, mais bien d'une exploitation du lien de réalisation dans ce langage.

Pour clore cette section, remarquons à nouveau que nous avons seulement présenté un langage de définition de contraintes d'intégrité, et une sémantique associée sous forme de technique d'évaluation. Une étude rigoureuse d'une technique de maintien de ces contraintes (à la manière des contraintes intrinsèques au modèle vues dans 3.1.1) est un complément nécessaire à ce travail.

3.2 Interrogation d'une base EMIR

Après avoir défini le modèle de données EMIR et étudié les aspects de maintien de l'intégrité des données, dans la section précédente, nous nous intéressons, dans celle-ci, à la manière d'interroger une base de données EMIR.

L'interrogation d'une base de données est l'action de rechercher des unités d'information dans un très gros volume d'informations. Il s'agit de sélectionner parmi des milliers, voire des millions d'unités informationnelles celles qui répondent au besoin de l'utilisateur en information.

Dans le cadre des bases de données structurées, le schéma conceptuel de la base décrit la manière dont sont structurées ses unités informationnelles.
Dans une base de données relationnelle [DA82], une requête sélectionne dans certaines tables, certains tuples qui répondent à des critères bien précis. Ces critères sont spécifiés en précisant quelles valeurs doivent être prises par quels attributs. Les requêtes sont donc définies en utilisant la structure conceptuelle de la base.
Dans une base de données orientée objet [DLR91, CAT94], une requête sélectionne dans certaines classes, certains objets qui répondent, encore une fois, à des critères spécifiés en utilisant la structure conceptuelle de la base (les variables d'instance et éventuellement certaines méthodes).
Dans la suite, ce genre de requêtes, dont la spécification repose sur la structure du schéma conceptuel seront appelées requêtes basées sur la structure ou *requêtes de structure.*

Dans le cadre des systèmes de recherche d'information [SM83, FBY92], il n'y a pas de schéma conceptuel, ou du moins il n'est pas défini par l'utilisateur. Les documents ne sont pas modélisés, mais plutôt *indexés.* A chaque document est associée, par exemple, une

liste de descripteurs qui traduisent son contenu réel. Ce contenu n'obéit pas à un schéma conceptuel préétabli, comme dans une base de données structurée.

Une requête, ne pouvant s'appuyer sur un schéma conceptuel inexistant, cherche des documents qui *contiennent* certaines informations ; c'est à dire des documents ayant été indexés avec ces informations. Autrement dit, la formulation de la requête ne s'adresse pas à une structure (conceptuelle) préétablie, mais plutôt au contenu effectif des documents.

Dans la suite, ce genre de requêtes, dont la spécification repose sur les éléments d'indexation du contenu réel des objets seront appelées requêtes basées sur le contenu ou *requêtes de contenu.*

Les langages de requêtes pour bases de données structurées sont de différents types : prédicatifs [BK86] (en particulier à la SQL [CDV88, LR89, KIM89, ASL89, CAT94]), algébriques [COL90, SZ90, SGL93, MO94] ou encore logiques [AK89].

Dans un langage prédicatif (aussi appelé *calcul*), une requête est exprimée selon une formule $\{x, \varphi(x)\}$, qui précise simplement le résultat attendu et non la manière de l'obtenir. Dans un langage à la SQL, par exemple, la syntaxe de base est la célèbre clause `SELECT...FROM...WHERE...`.

Dans un langage algébrique, comme son nom l'indique, une algèbre d'opérations est définie et la spécification d'une requête algébrique est la suite des opérations à exécuter pour obtenir le résultat. Un grand intérêt est d'ailleurs souvent attaché au fait de démontrer l'équivalence entre un calcul et une algèbre [AB86, AB88, RKS88, STR91].

Les requêtes dans un langage logique comme IQL [AK89], sont définies comme un programme logique à la Prolog, c'est à dire sous forme de règles de production, qui construisent la requête progressivement depuis des critères élémentaires.

Dans les systèmes de recherche d'information, une requête est formulée en fournissant un ensemble d'indices que doivent contenir les objets recherchés. Ces indices peuvent être des mots-clés pour les textes [SCH90] ou les images [HCK90], des indices visuels ou des croquis pour les images [TOU91, KAT92, HK92], voire des concepts spéciaux pour décrire les objets (approche grammaticale dans [HLMM92] ou théorie des ensembles flous et interrogation flexible dans [SUB95]).

L'exécution de telles requêtes repose sur la structure d'indexation de la base qui relie, en général, chaque primitive d'indexation (mot-clé, indice visuel, distribution de pixels de couleurs, ...) à l'ensemble des documents qui le contiennent.

La technique d'indexation dans une base de données structurées est différente. Des objets de la même classe sont groupés entre eux, par exemple. Mais l'indexation du contenu est demandée explicitement par l'utilisateur. Il peut ainsi faire correspondre à chaque valeur d'un attribut donné, l'ensemble des objets ayant cette valeur pour cet attribut. Cette primitive d'indexation sera utile seulement pour des requêtes portant sur cet attribut.

Si l'indexation est *a priori* plus complète dans les systèmes de recherche d'information, le pouvoir d'expression de ceux-ci est très limité, tant au niveau de la modélisation des documents qu'*a fortiori* de l'interrogation. Les bases de données structurées sont par ailleurs difficiles à indexer correctement pour n'importe quel type de requêtes.

Notre objectif est de définir un langage de requêtes adapté au modèle EMIR. Ce langage doit utiliser le schéma conceptuel, en permettant la spécification de requêtes navigationnelles, portant sur les trois facettes de modélisation d'une catégorie. Il doit en outre permettre de spécifier des requêtes basées sur le contenu permettant d'interroger la structure individuelle des objets.

Dans une base de données orientée objet par exemple, en recherchant les employés âgés de 50 ans, habitant à Nancy, on sait *a priori* que la classe *Employé* a un attribut donnant l'âge (*âge*) et un attribut donnant la ville (*adresse*). La requête peut donc être exprimée, "en navigant" à travers la structure des classes, par l'ensemble: $\{o \in Employé, o.âge = 50 \wedge o.adresse.ville = "Nancy"\}$.

Dans un système de recherche d'information, on peut rechercher des images *contenant* des personnes, par exemple.

Le langage de requêtes du modèle EMIR combine les deux types de requêtes. Il permet, par exemple, de rechercher des images contenant des employés nancéiens âgés de plus de 50 ans, ou encore des images contenant un cheval blanc à gauche d'une maison rouge!

La solution adoptée consiste à définir une requête à la manière d'un objet EMIR. Une partie description comprend les critères qui peuvent être exprimés à un niveau conceptuel (mettant en jeu des caractéristiques de la catégorie destination). Une partie composition est formée de sous-requêtes auxquelles doivent répondre des objets de la composition (obligatoire ou individuelle) de l'objet principal répondant à la requête. Les sous-requêtes mettent en jeu des composants qui ne peuvent être envisagés au niveau conceptuel. Enfin, une partie topologie permet de spécifier des relations sémantiques particulières entre les composants de l'objet, en général, et ceux parmi eux qui répondent aux sous-requêtes, en particulier.

Dans la suite, nous définissons, selon cette technique, un langage de requêtes approprié au modèle, exploitant notamment l'intérêt du lien de réalisation.

Nous présentons dans 3.2.1 la syntaxe générale d'une requête, et dans 3.2.2 sa sémantique formelle, le tout agrémenté d'exemples.

3.2.1 Syntaxe d'une requête

Une requête EMIR cherche des objets d'une catégorie donnée, répondant à un certain nombre de conditions. Ces objets sont ensuite projetés selon un certain chemin pour construire le résultat effectif de la requête.

A la manière de la structure d'un objet, les conditions d'une requête sont divisées en trois types: des conditions de structure, de contenu et de topologie.

- Les conditions de structure sont des assertions valables pour la catégorie cible de la requête (cf. 3.1.2.1.9), appelées *critères* (de la requête courante). Elles représentent des conditions utilisant la structure obligatoire des objets (leurs attributs, leurs composants *obligatoires* ou leur topologie *obligatoire*).

  Par exemple, on recherche les images en noir et blanc (attribut) dont l'auteur (composant obligatoire) habite Nancy.

- Les conditions de contenu forment un ensemble de requêtes EMIR, appelées *sous-requêtes* (de la requête courante). A chacune de ces sous-requêtes, au moins un composant de l'objet satisfaisant la requête courante doit répondre. Ce composant est *a priori* quelconque, mais il est possible de contraindre un composant obligatoire d'objet à répondre à une sous-requête, en faisant précéder celle-ci par le rôle de ce composant).

  Les conditions de contenu peuvent éventuellement se réduire à l'appartenance à une catégorie donnée. Les objets répondant à la requête courante doivent, dans ce cas, inclure dans leur composition (obligatoire ou individuelle) au moins un objet réalisant cette catégorie.

  Ces conditions représentent la partie récursive des requêtes EMIR et permettent de spécifier des requêtes de contenu, à la manière des systèmes de recherche d'informations.

  Par exemple, on recherche des images contenant des voitures rouges et dont le propriétaire habite Nancy.

  Il ne doit pas y avoir de cycle, dans la définition récursive des requêtes.

- Les conditions de topologie, appelées *liens* (de la requête courante) ajoutent des restrictions topologiques à la requête. Elles stipulent qu'une certaine relation sémantique doit exister entre des composants des objets répondant à la requête courante. Ces composants peuvent, en plus, être contraints de répondre à l'une des sous-requêtes de contenu.

  Un lien topologique est la définition en intension d'une relation, reliant des composants de l'objet recherché, désignés soit directement par leur rôle dans la composition soit par des sous-requêtes dont ils doivent être des résultats. La définition d'un lien topologique pour une requête ressemble à celle d'un lien de catégorie, donnée dans 2.3.3.1, excepté qu'en plus des composants obligatoires de l'objet, il peut également mettre en jeu des sous-requêtes de la requête.

  Par exemple, les voitures (sous-requête) des images précédentes doivent être devant (relation sémantique) un hôtel (autre sous-requête).

Nous définissons une requête comme étant un quintuplet $q = (c, Cl, Q, L, p)$ où :

- c est une catégorie, dite *cible* de q,

- $Cl = \{cl_1, ..., cl_n\}$ est un ensemble d'assertions valables pour la catégorie c (cf. 3.1.2.1.9), dites *critères* de q,

- $Q = \{ro_1 : q_1, ..., ro_m : q_m\}$ est un ensemble de requêtes, dites *sous-requêtes* de q, précédées par d'éventuels rôles de composants de c pour forcer leur évaluation au composant en question ; si aucun composant particulier n'est visé, nous utilisons le rôle *vide* comme pour la composition individuelle des objets (cf 2.2.6.1),

- $L = \{l_1, ..., l_p\}$ est un ensemble de liens topologiques, dits *liens* de q; chaque lien l_k, $k \in [1..p]$ est défini par: $l_k : r_k(cr_{k1} : x_{k1}, ..., cr_{kt} : x_{kt})$, où r_k est une relation sémantique ayant t composants: cr_{ku}, $u \in [1..t]$, et où les x_{ku} représentent les objets qui doivent jouer ces rôles dans la relation. Ces représentants d'objets sont soit des composants de c (désignés par leurs rôles), soit des sous-requêtes (élements de Q, désignés par leurs noms), soit encore l'objet recherché lui-même (désigné par *Self*),

- $p = p_1.p_2...p_q$ est un chemin valide pour c, dit *projection* de q.

Les critères, les sous-requêtes, les liens et/ou la projection peuvent éventuellement être vides. S'ils le sont tous, la requête donne comme résultat tous les objets réalisant la catégorie cible ou l'une de ses catégories spécifiques.

Avant de donner la sémantique formelle d'une requête, nous allons en présenter informellement le sens et en donner des exemples.

La requête q cherche des objets o:

- réalisant la catégorie c (ou l'une de ses catégories spécifiques),
- vérifiant toutes les assertions de l'ensemble Cl,
- contenant pour chaque requête de l'ensemble Q, un objet au moins, qui en soit un résultat; si la requête est précédée d'un composant de catégorie, cet objet doit obligatoirement être celui qui joue ce rôle dans o,
- ayant pour chaque lien de l'ensemble L, une relation concrète qui l'implémente. Cela veut dire qu'elle fait référence à la relation sémantique en question et, qu'à chaque composant cr de cette relation, elle lie un objet satisfaisant les conditions suivantes:
  - si le composant cr est lié à un composant de c de rôle cc, l'objet qui joue ce rôle dans la composition de o doit jouer le rôle cr dans la relation concrète,
  - si le composant cr est lié à une sous-requête, un composant de o qui résulte de cette sous-requête doit jouer le rôle cr dans la relation concrète,
  - si le composant cr est lié à *Self*, l'objet o doit jouer lui-même le rôle cr dans la relation concrète.

Le résultat de q est l'ensemble des projections selon le chemin p, des objets vérifiant toutes ces conditions.

Exemples de requêtes:

La requête $q_1 = (c_1, Cl_1, Q_1, L_1, p_1)$ cherche les employés âgés de 50 ans, habitant Nancy.
$c_1 = Employé$
$Cl_1 = \{âge = 50, "Nancy" \in adresses.ville\}$
$Q_1 = \{\}$
$L_1 = \{\}$
p_1 est vide

La requête $q_2 = (c_2, Cl_2, Q_2, L_2, p_2)$ cherche des voitures.
$c_2 = Voiture$
$Cl_2 = \{\}$
$Q_2 = \{\}$
$L_2 = \{\}$
p_2 est vide

La requête $q_3 = (c_3, Cl_3, Q_3, L_3, p_3)$ cherche les noms d'auteurs d'images contenant un employé nancéien et cinquantenaire, se trouvant devant une voiture.
$c_3 = Image$
$Cl_3 = \{\}$
$Q_3 = \{vide : q_1, vide : q_2\}$
$L_3 = \{l_1 : Au\_Devant\_de(devant : q_1, derrière : q_2)\}$
$p_3 = auteur.nom$

avec la relation *Au_Devant_de* ayant deux composants : $(devant : Any_c)$ et $(derrière : Any_c)$.

La requête $q_4 = (c_4, Cl_4, Q_4, L_4, p_4)$ cherche les titres d'images contenant une voiture en arrière plan, et dont l'auteur habite Nancy.
$c_4 = Image$
$Cl_4 = \{"Nancy" \in auteur.adresses.ville\}$
$Q_4 = \{vide : q_2\}$
$L_4 = \{l_1 : Arrière\_plan(image : Self, composant : q_2)\}$
$p_4 = titre$

avec la relation *Arrière_plan* ayant deux composants : $(image : Image)$ et $(composant : Any_c)$.

La requête $q_5 = (c_5, Cl_5, Q_5, L_5, p_5)$ cherche les images dont les auteurs répondent à la requête q_1 (employés cinquantenaires nancéiens).
$c_5 = Image$
$Cl_5 = \{\}$
$Q_5 = \{auteur : q_1\}$
$L_5 = \{\}$
p_5 est vide

Comme les requêtes sont définies de la même manière que les objets, elles peuvent être représentées avec le même formalisme graphique. Ainsi, la figure 3.6 représente la requête q_4.

L'intérêt du lien de réalisation au niveau des requêtes est contenu dans les ensembles Q et L, qui spécifient des critères de recherche portant sur toute la structure des objets (y compris les composants et les relations concrètes individuelles) et non pas uniquement

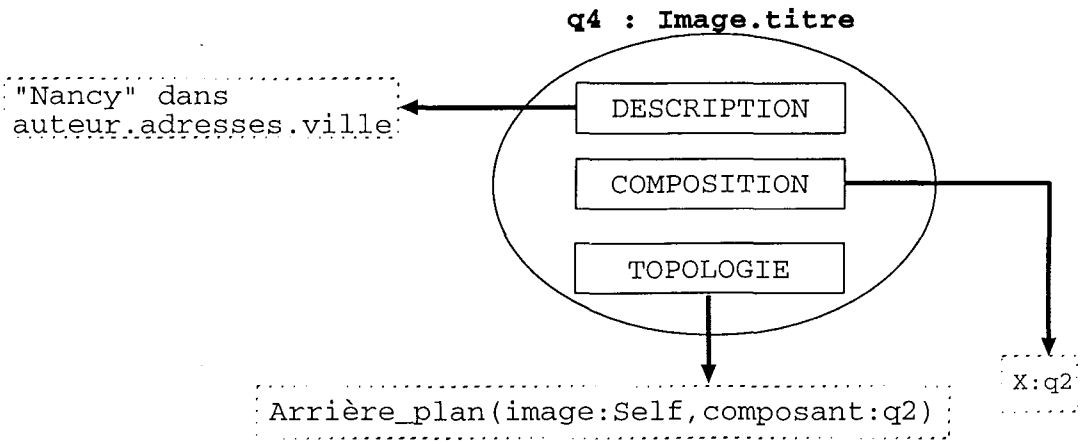


FIG. 3.6 – *Un exemple de requête EMIR*

sur la structure de la catégorie c, qui elle, est exploitée dans les critères Cl.

Une requête EMIR peut donc définir des critères de sélection portant sur des aspects de structure autres que ceux de la catégorie cible.

3.2.2 Sémantique formelle et exemples

Nous définissons le résultat d'une requête $q = (c, Cl, Q, L, p)$ par un ensemble $\mu_q \subseteq \mathcal{O}$ comme suit : $\mu_q = \{Dest(o, p), o \in \mathcal{O} \wedge o \vdash q\}$.

Cet ensemble est obtenu en projetant les objets *satisfaisant* la requête, selon le chemin de projection de celle-ci.

La satisfaction d'une requête q par un objet concret o, notée $o \vdash q$ est définie comme suit :

- pour un objet simple o tel que :
  - $A(o) = \langle a_1 : v_1, ..., a_{n'} : v_{n'} \rangle$,
  - $C(o) = \langle ro_1 : o_1, ..., ro_{m'} : o_{m'} \rangle$,
  - $L(o) = \langle rc_1 : r_1, ..., rc_{p'} : r_{p'} \rangle$, avec $C(rc_{k'}) = \langle cr_{k'1} : o_{k'1}, ..., cr_{k't} : o_{k't} \rangle$ pour tout $1 \leq k' \leq p'$,

  on a $o \vdash q = (c, Cl, Q, L, p)$ (requête définie avec les mêmes notations que dans 3.2.1), si et seulement si les conditions suivantes sont vérifiées.
  - $o \leftarrow c$: l'objet o réalise la catégorie c.
  - $\forall cl_i \in Cl$, $eval(cl_i, o, -) = VRAI$: l'évaluation de toutes les clauses de Cl dans o est vraie ; cette évaluation se fait selon la technique présentée dans 3.1.2.2 pour les contraintes d'intégrité.
  - $\forall ro_j : q_j \in Q$:
    - si $ro_j \neq$ "*vide*", alors $o_j \in \mu_{q_j}$: si la sous requête s'adresse à un composant obligatoire d'objet, l'objet composant en question doit appartenir à l'ensemble résultat de la sous-requête,

- si ro_j = "*vide*", alors $\exists ro_{j'} : o_{j'} \in C(o), o_{j'} \in \mu_{q_j}$: si la sous-requête s'adresse à un composant quelconque ; dans ce cas, elle doit inclure dans son ensemble résultat, au moins un objet composant de o.

- $\forall l_k \in L$ tel que $l_k : r_k(cr_{k1} : x_{k1}, ..., cr_{kt} : x_{kt})$, $\exists rc_{k'} : r_{k'} \in L(o)$, telle que $r_{k'} \leq_r r_k \wedge \forall 1 \leq u \leq t$,
  - si x_{ku} = "*Self*", alors $o_{k'u} = o$
  - si $x_{ku} = ro_{j'}$, avec $1 \leq j' \leq m'$, alors $o_{k'u} = o_{j'}$
  - si $x_{ku} = q_j$, avec $1 \leq j \leq m$, alors $o_{k'u} \in \mu_{q_j}$

  Cela signifie qu'à chaque lien de la requête, doit correspondre une relation concrète de l'objet qui instancie une relation plus spécifique que celle du lien. De plus, pour chaque composant de cette relation concrète, l'objet mis en jeu doit répondre au critère fixé dans le même composant, au niveau du lien. Ce critère peut impliquer l'objet principal ou un composant obligatoire particulier ou encore une sous-requête. Dans ce dernier cas, l'objet mis en jeu dans le composant de la relation concrète doit appartenir à l'ensemble résultat de cette sous-requête.

- pour un objet ensemble o, on a $o \vdash q$, si et seulement si $\forall o' \in s(o), o' \vdash q$.

Exemples

Reprenons les cinq exemples de requêtes données précédemment dans 3.2.1 et considérons les objets présentés dans la figure 3.7. Cette figure est complétée au niveau conceptuel par la figure 3.8.

L'objet *Employé* 1 satisfait la requête q_1. Donc il appartient à μ_{q_1} car p_1 est vide (pas de chemin de projection).

L'objet *Photographe* 1 ne satisfait pas la requête q_1.

L'objet *Voiture* 1 satisfait la requête q_2. Donc il appartient à μ_{q_2} car p_2 est vide (pas de chemin de projection).

L'objet *Image* 1 satisfait la requête q_3. Donc l'objet *Photographe* 1 appartient à μ_{q_3} car c'est l'auteur de *Image* 1.

L'objet *Image* 1 satisfait la requête q_4. Donc son titre "*Scène de rue No 1*" appartient à μ_{q_4}.

L'objet *Image* 1 ne satisfait pas la requête q_5 car son auteur (l'objet *Photographe* 1) ne satisfait pas la requête q_1.

Conclusion

Dans cette section, nous avons d'une part, étudié deux aspects du maintien de l'intégrité dans le modèle EMIR, d'autre part, nous avons présenté la technique retenue pour interroger une base de données EMIR.

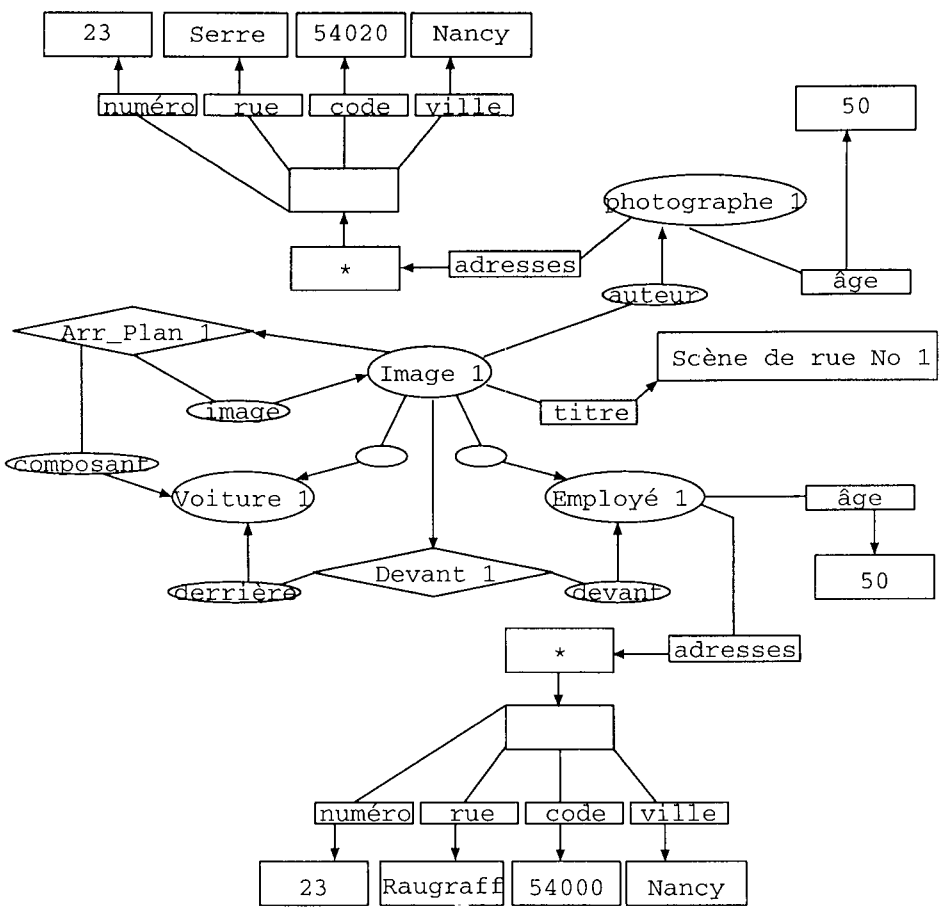


FIG. 3.7 – *Un schéma concret dans le formalisme d'EMIR*

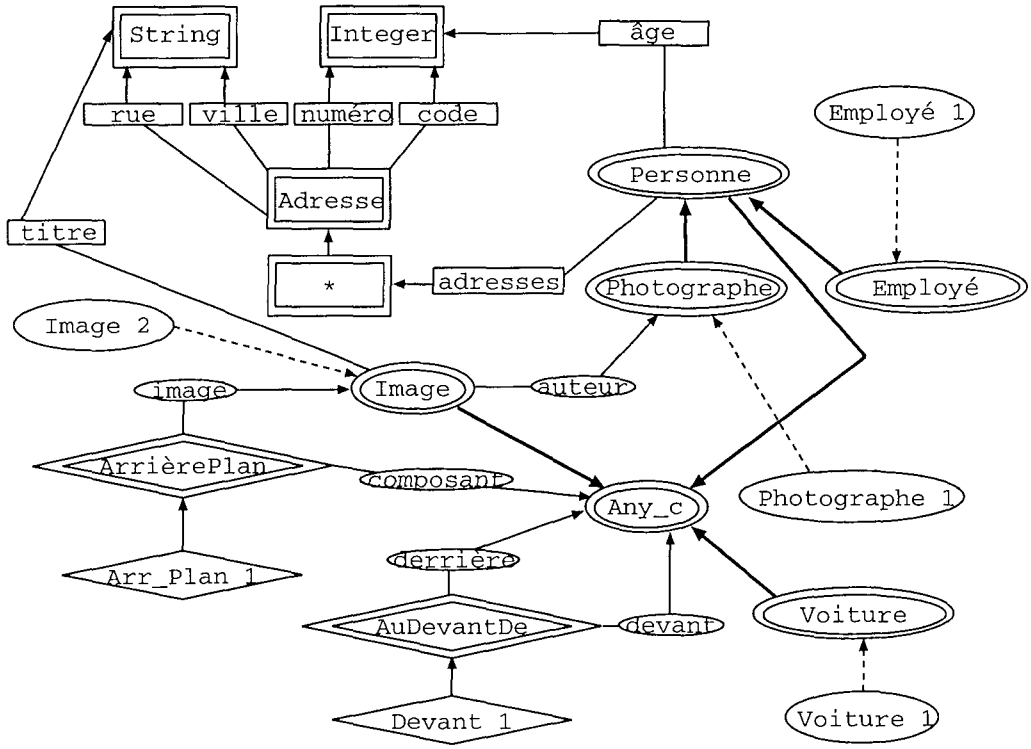


FIG. 3.8 – *Quelques objets, catégories et relations dans le formalisme d'EMIR*

Le premier aspect de l'intégrité est lié au modèle de données lui-même. Nous avons spécifié formellement les règles de cohérence d'une base de données EMIR, sous forme d'un ensemble d'invariants, devant être respectés à tout moment de la vie d'une base EMIR. Nous avons développé une méthode de maintien de ces invariants, en explorant de manière exhaustive toutes les opérations possibles sur une base EMIR. Pour chaque opération, nous avons dégagé les invariants qu'elle est susceptible de violer et défini, pour chacun, des traitements visant à le maintenir.

Le second aspect de la cohérence des bases de données EMIR est lié aux applications. Nous avons proposé un langage de spécification de contraintes d'intégrité, sous forme d'assertions, sur des bases EMIR. Ces assertions peuvent être liées à des catégories, à des relations ou encore à des objets concrets particuliers. La spécification des contraintes liées aux applications tire profit des spécificités du modèle de données, en particulier en ce qui concerne le lien de réalisation (noms d'objets dans les assertions) et la topologie (attachement de contraintes à des relations et utilisation de noms de liens topologiques dans les assertions).

Par ailleurs, les requêtes ont été définies, à l'instar des objets EMIR, selon trois facettes, et ont pour cible une catégorie donnée. La première facette représente l'aspect navigationnel des requêtes. Elle utilise la structure de la catégorie cible, sous forme de critères qui sont des assertions définies, sur la catégorie. La seconde facette représente l'aspect basé sur le contenu des requêtes. Elle est constituée d'une liste de sous-requêtes, devant être satisfaites par des composants des objets répondant à la requête. Enfin, une

dernière facette permet de spécifier des contraintes topologiques sur les objets résultats.

Cette technique permet de formuler des requêtes très diverses. Nous en avons donné des exemples illustratifs.

Dans le chapitre suivant, nous présentons le prototype EMIR, dans sa version actuelle, programmée en Smalltalk-80, ainsi que deux applications particulières du modèle, pour illustrer sa puissance d'expression.

Chapitre 4

Implantation et validation

Dans cette partie, nous présentons d'abord quelques caractéristiques du prototype réalisé pour implanter le modèle, dans 4.1. Ensuite, dans le but de valider notre approche par des exemples concrets, nous présentons succinctement deux applications de notre modèle : la première, détaillée dans 4.2, concerne la modélisation et la recherche d'informations dans une base d'images. La seconde, qui fait l'objet du paragraphe 4.3, concerne la modélisation de projets architecturaux, au cours du processus de conception.

4.1 Le prototype EMIR

Pour valider notre approche, nous avons développé un prototype avec le langage orienté-objet Smalltalk-80 [GR83], sur une station SPARC.

Chaque concept du modèle (objets, catégories, attributs, requêtes, ...) est représenté par une classe Smalltalk. Lorsqu'une instanciation de ce concept a lieu (création d'une catégorie, d'un objet concret, d'une requête ...), des objets Smalltalk sont créés et reliés entre eux selon la définition des classes Smalltalk correspondantes. Nous avons donc un schéma de classes sous Smalltalk, que nous appelons *méta-schéma*, pour ne pas le confondre avec les schémas conceptuels ou concrets des applications EMIR. En général, nous utiliserons le préfixe "méta" pour nous situer dans le contexte et la terminologie de Smalltalk, et non dans ceux d'EMIR. Ce méta-schéma renferme la structure générale du modèle EMIR, indépendamment des particularités de chaque application.

A chaque classe du méta-schéma, appelée *méta-classe EMIR*, pour ne pas la confondre avec une catégorie d'objets EMIR, sont associés un ensemble de variables d'instance et un ensemble de méthodes, qui en définissent la structure et le comportement. Ce niveau de description (appelé méta-niveau) est évidemment transparent à l'utilisateur final du prototype. Les méta-classes EMIR ne doivent pas être confondues non plus avec les méta-classes de Smalltalk, puisqu'il s'agit simplement de classes Smalltalk.

La figure 4.1, présente le méta-schéma d'EMIR dans le formalisme orienté objet d'ODMG-93, emprunté à [CAT94].

Le rectangle en pointillé montre comment est gérée la différence de structure entre un objet concret et sa catégorie. Deux méta-classes EMIR abstraites (classes sans instances) sont créées : *Object Component* et *Relationship*. Elles sont spécialisées chacune en deux sous-classes ; l'une pour les aspects obligatoires (composants et relations concrètes provenant de la catégorie, qui gardent d'ailleurs un lien vers elle) et l'autre pour les aspects individuels (composants et relations concrètes particulières à l'objet concret en question).

Le prototype EMIR permet de :

- créer, supprimer, charger, décharger (à partir de fichiers) une application donnée (cf. figure 4.2),
- créer, modifier ou supprimer des domaines particuliers (discrets ou continus) définis pour une application donnée, (cf. 4.1.1, figure 4.4),

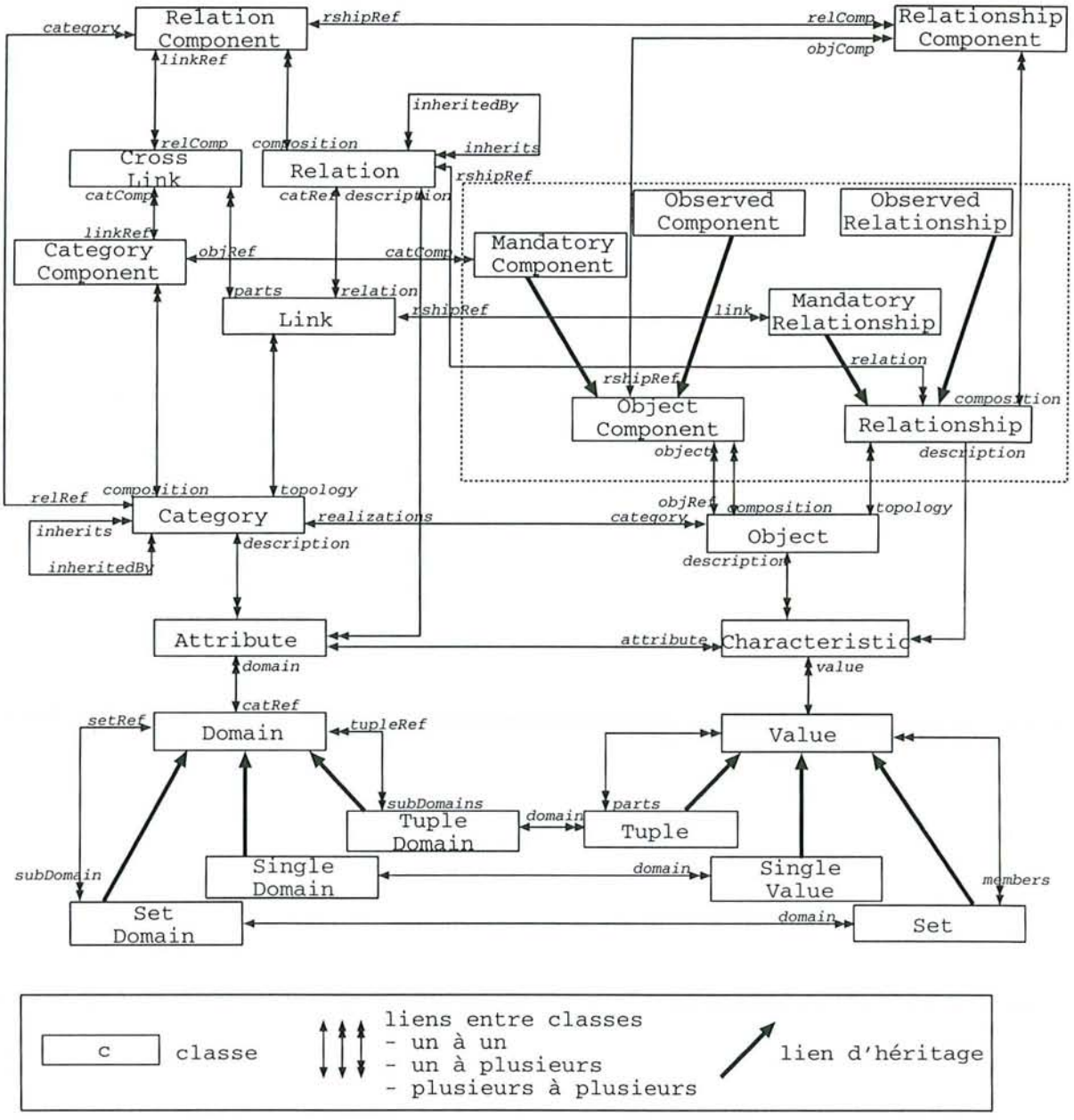


FIG. 4.1 – *Une partie du schéma ODMG-93 des méta-classes EMIR*

- créer, modifier ou supprimer des catégories ou des relations (schéma conceptuel), dans ce qu'on appelle un *browser conceptuel* (cf. 4.1.1, figure 4.3),
- créer, modifier ou supprimer des objets (schéma concret incluant les relations concrètes), dans ce qu'on appelle un *browser concret* (cf. 4.1.2, figure 4.7),
- créer, modifier ou supprimer des requêtes, dans ce qu'on appelle un *browser de requêtes* (cf. 4.1.3, figure 4.15),
- visualiser le résultat d'une requête sous forme d'un browser concret réduit aux objets qui répondent à la requête (permettant en particulier d'y naviguer), ou sous forme simplement d'une liste de valeurs, si le résultat de la requête est de cette forme (cf. 4.1.3, figure 4.16),
- spécifier des contraintes d'intégrité sur un objet, une catégorie ou une relation, à l'aide de ce qu'on appelle un *browser de contraintes* (cf. 4.1.1, figure 4.6),
- afficher la liste des contraintes d'intégrité violées par un objet concret particulier (cf. 4.1.2, figure 4.14).

Le browser de la figure 4.2 est la fenêtre principale du prototype. Elle renferme les noms des différentes applications créées. Une application est composée d'un schéma conceptuel, et d'un schéma concret, auxquels sont éventuellement attachées des contraintes d'intégrité (au niveau des catégories, des relations ou des objets), et enfin d'un ensemble de requêtes.

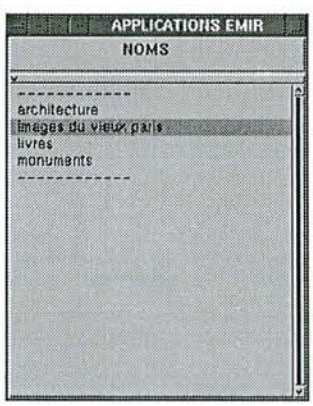


FIG. 4.2 – *Browser général des applications*

L'utilisateur sélectionne l'application qui l'intéresse et peut alors l'explorer ou la mettre à jour.

4.1.1 Browser conceptuel

Le browser conceptuel (figure 4.3) permet à l'utilisateur du modèle de définir le schéma conceptuel d'une application. Ce schéma est formé d'un ensemble de catégories, organisées en hiérarchie d'héritage, et d'un ensemble de relations, organisées également en hiérarchie d'héritage.

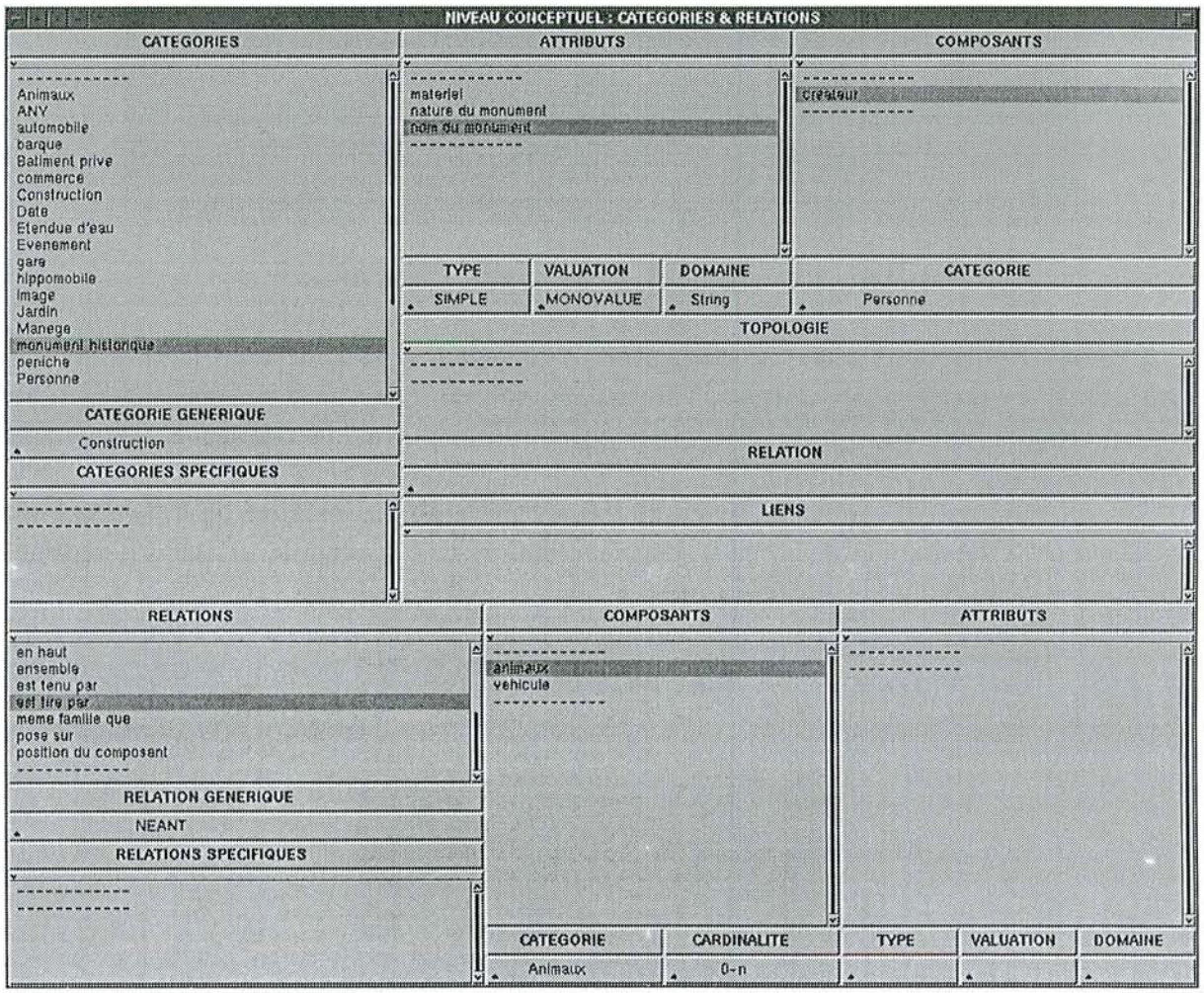


FIG. 4.3 – *Browser conceptuel*

Un browser annexe, appelé browser de domaines particuliers (figure 4.4) permet de saisir et de mettre à jour des domaines particuliers de l'application (continus ou discrets).

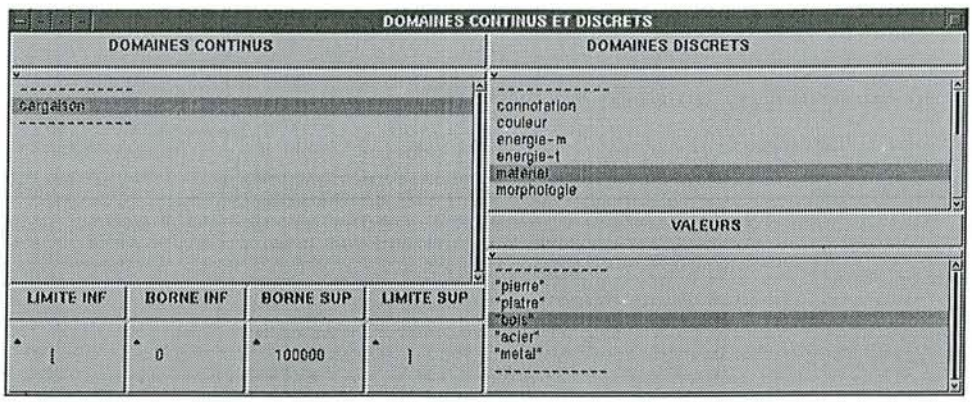


FIG. 4.4 – *Browser de domaines particuliers*

Lorsque l'utilisateur crée une catégorie c, un méta-objet EMIR (objet Smalltalk), instance de la méta-classe *Category* est créé pour c et relié aux éventuels méta-objets des catégories dont hérite c, à travers la variable d'instance *inherits* de la méta-classe *Category*.

Quand l'utilisateur ajoute un attribut à c, un méta-objet, instance de la méta-classe *Attribute* est créé et relié au méta-objet représentant c, à travers la variable d'instance *description* de la méta-classe *Category*.

Si l'attribut est agrégé, un browser d'attributs agrégés du type de celui présenté dans la figure 4.5 est ouvert pour la saisie des sous-attributs, qui peuvent à leur tour être simples ou agrégés.

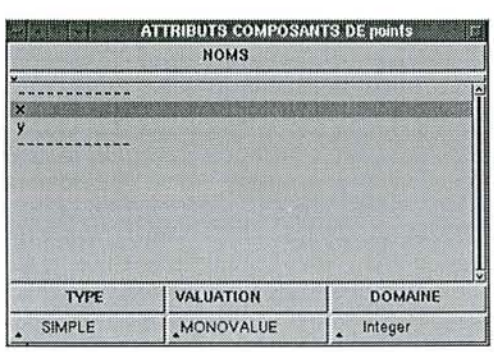


FIG. 4.5 – *Browser d'attributs agrégés*

Quand l'utilisateur ajoute un composant à c, un méta-objet, instance de la méta-classe *Category Component* est créé et relié au méta-objet représentant c, à travers la variable d'instance *composition* de la méta-classe *Category*.

Quand l'utilisateur ajoute un lien à c, un méta-objet, instance de la méta-classe *Link* est créé et relié au méta-objet représentant c, à travers la variable d'instance *topology* de

la méta-classe *Category*.

Une technique similaire est suivie lors de la création et de la mise à jour d'une relation.

Lorsque l'utilisateur décide d'attacher une contrainte d'intégrité à une catégorie c ou à une relation r, un browser de contraintes (figure 4.6) est ouvert. L'utilisateur y entre le nom de la contrainte, son expression selon un langage qui implante les spécifications présentées dans la partie 3.1 [ISS95], et un éventuel commentaire.

La saisie de l'expression de la contrainte est effectuée au clavier mais, à chaque instant, il y a possibilité de faire appel à un navigateur qui permet de définir de façon correcte les chemins valides issus de c ou de r.

Une fois la contrainte spécifiée, un méta-objet, instance de la méta-classe *Constraint* est créé et relié au méta-objet représentant c ou r à travers une variable d'instance *constraints* de la méta-classe *Category* ou *Relation*.

Les méta-classes ayant trait aux contraintes (ainsi que celles liées aux requêtes) n'ont pas été représentées dans le schéma de la figure 4.1 pour des soucis de clarté et de lisibilité de ce schéma.

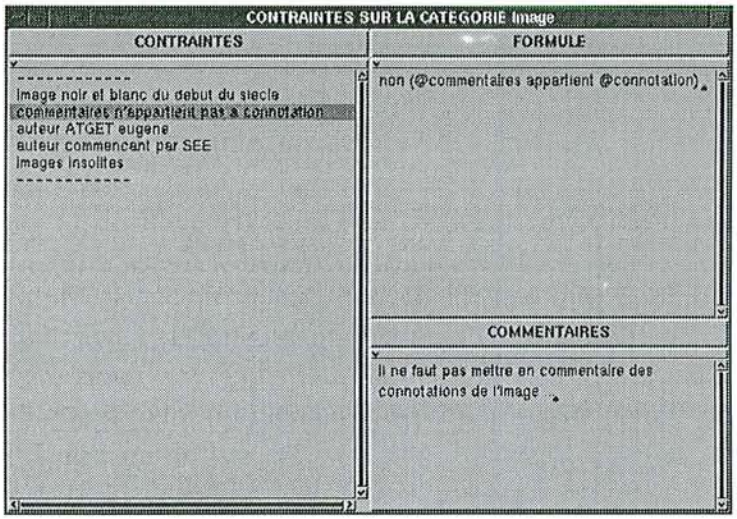


FIG. 4.6 – *Browser de contraintes*

4.1.2 Browser concret / Implantation du lien de réalisation

Lorsque l'utilisateur crée un objet concret o, comme étant une réalisation de la catégorie c, un méta-objet, instance de la méta-classe *Object* est créé pour o et relié au méta-objet représentant c à travers la variable d'instance *category* de la méta-classe *Object*.

Le mécanisme de réalisation consiste alors en les trois étapes suivantes.

- Implantation de la description :
  pour chaque attribut a de c, relié au domaine d, un méta-objet, instance de la méta-classe *Characteristic* est créé et relié au méta-objet représentant o, à travers la variable d'instance *description* de la méta-classe *Object* (cf. figure 4.8).

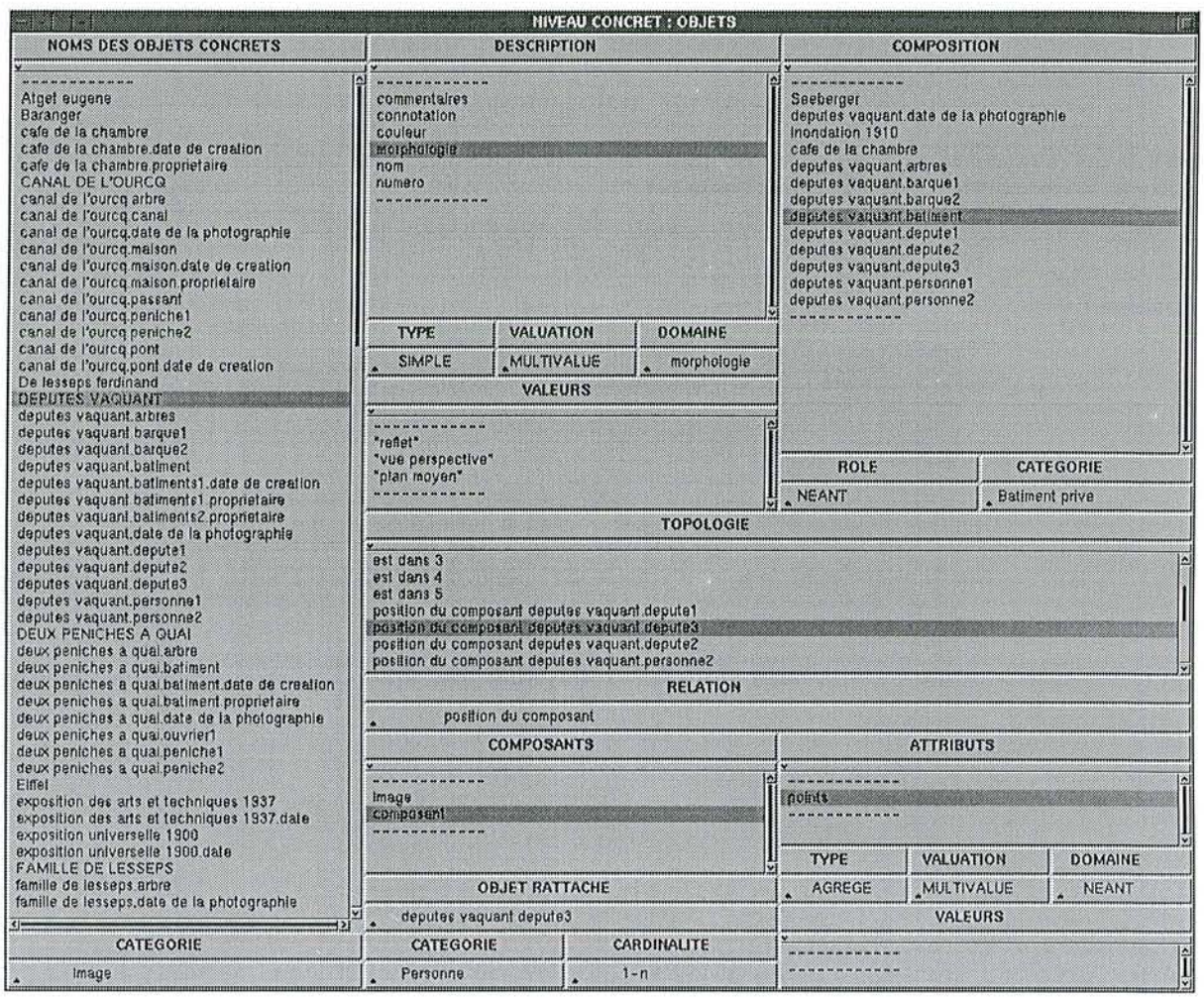


FIG. 4.7 – *Browser concret*

La variable d'instance *attribute* de la méta-classe *Characteristic* référence, pour chaque caractéristique de o, le méta-objet représentant l'attribut a de c. Quant à la variable d'instance *value*, elle référence un méta-objet, instance de la méta-classe *Value*, spécialement créé et renfermant la valeur par défaut du domaine d.

L'utilisateur a ensuite tout le loisir de mettre à jour cette valeur.

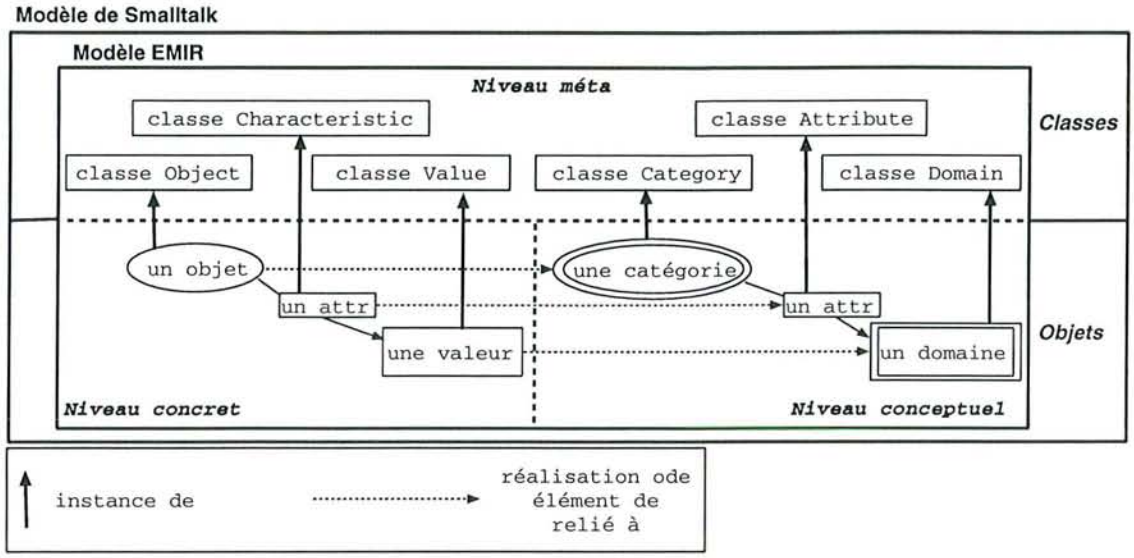


FIG. 4.8 – *Une vue partielle du méta-modèle (attributs et caractéristiques)*

- Implantation de la composition :
  pour chaque composant *co* de c, relié à une catégorie c', un méta-objet, instance de la méta-classe *Mandatory Component* est créé et relié au méta-objet représentant o, à travers la variable d'instance *composition* de la méta-classe *Object* (cf. figure 4.9).

  La variable d'instance *catComp* de la méta-classe *Mandatory Component* référence, pour chaque composant obligatoire de o, le méta-objet représentant le composant *co* de c. Quant à la variable d'instance *object* (héritée de la méta-classe *Object Component*), elle référence un méta-objet, instance de la méta-classe *Object*, qui représente un objet concret o' réalisant la catégorie c'. Cet objet concret est soit créé soit choisi parmi les objets existants.

- Implantation de la topologie :
  pour chaque lien l de c, relié à la relation r, un méta-objet, instance de la méta-classe *Mandatory Relationship* (relation concrète obligatoire) est créé et relié au méta-objet représentant o, à travers la variable d'instance *topology* de la méta-classe *Object* (cf. figure 4.10).

  La variable d'instance *relation* de la méta-classe *Mandatory Relationship* référence, pour chaque relation concrète obligatoire de o, le méta-objet représentant la relation r. Quant à la variable d'instance *parts*, elle référence des méta-objets, instances de la méta-classe *Relationship Component*.

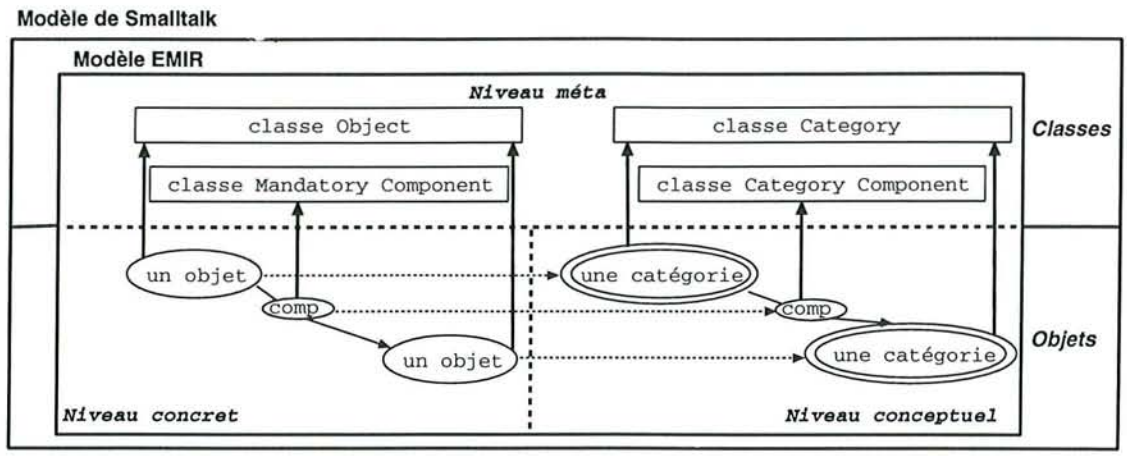


FIG. 4.9 – *Une vue partielle du méta-modèle (composants obligatoires d'objets)*

Chaque partie de la relation concrète est "calculée", en respectant la partie correspondante, dans le lien l. Ainsi, si $cr : cc$ est une partie de l statuant que le composant de catégorie cc joue le rôle cr (composant de la relation r) dans le lien l, le méta-objet représentant cette partie au niveau de la relation concrète sera relié d'une part au méta-objet représentant cr et d'autre part au méta-objet représentant le composant obligatoire relié à cc au niveau de l'objet o.

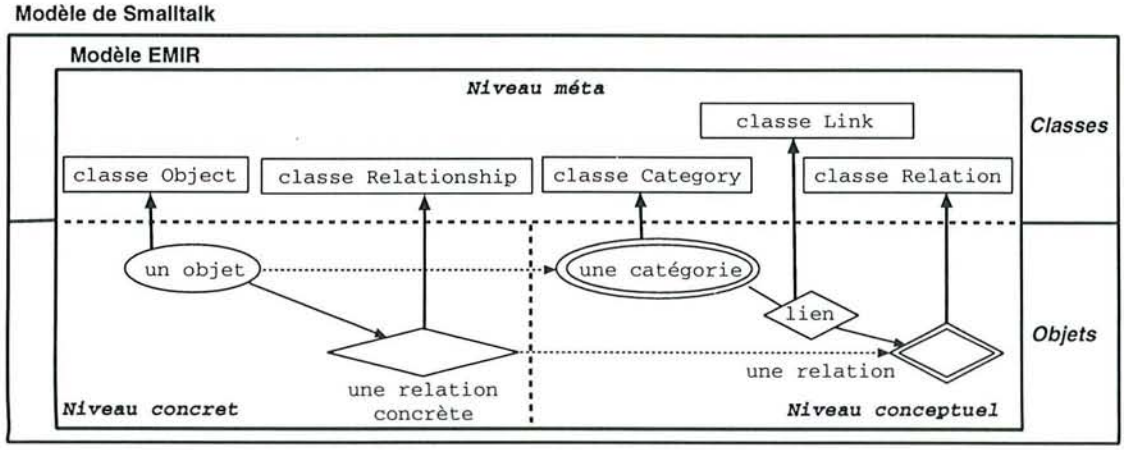


FIG. 4.10 – *Une vue partielle du méta-modèle (relations concrètes obligatoires d'objets)*

Remarquons que si l'utilisateur décide de mettre à jour la valeur d'un attribut agrégé au niveau d'un objet concret, un browser de caractéristiques agrégées du type de celui présenté dans la figure 4.11 est ouvert pour la saisie des nouvelles valeurs.

Ce browser affiche une valeur agrégée à la fois. Si l'attribut agrégé correspondant est en plus multivalué, l'utilisateur affiche les valeurs successives écran par écran, en passant d'une occurrence à l'autre. Le coin supérieur droit de la fenêtre indique le numéro de l'occurrence courante dans la liste.

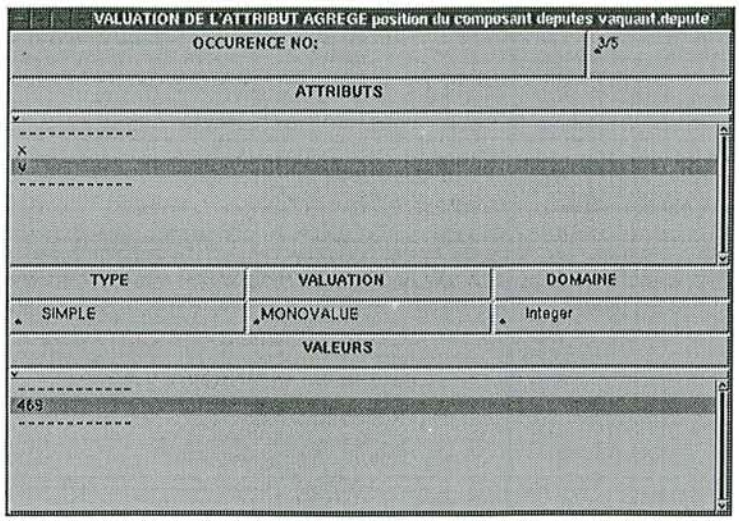


FIG. 4.11 – *Browser de caractéristiques agrégées*

A présent, après avoir présenté le mécanisme qui réalise un objet à partir d'une catégorie, nous nous intéressons à l'ajout de structures individuelles (composant ou relation concrète) dans un objet.

Quand l'utilisateur ajoute un composant individuel o' à un objet o, un méta-objet, instance de la classe *Observed Component* est créé et relié aux deux méta-objets représentant o et o' (cf. figure 4.12). Il est relié au premier à travers la variable d'instance *composition* de la méta-classe *Object* (dans o), et au second à travers la variable d'instance *object* de la méta-classe *Object Component*, dont hérite *Observed Component*.

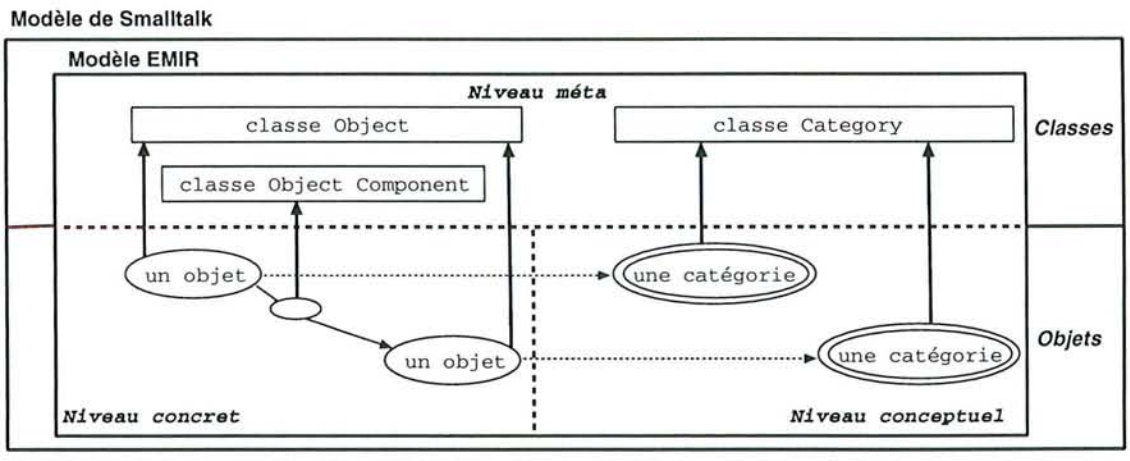


FIG. 4.12 – *Une vue partielle du méta-modèle (composants individuels d'objets)*

Quand l'utilisateur ajoute une relation concrète individuelle rc (reliée à une relation r) à un objet o, un méta-objet, instance de la classe *Observed Relationship* est créé et relié aux deux méta-objets représentant o et r (cf. figure 4.13). Il est relié au premier à

travers la variable d'instance *topology* de la méta-classe *Object* (dans o), et au second à travers la variable d'instance *relation* de la méta-classe *Relationship*, dont hérite *Observed Relationship*.

La variable d'instance *parts* du méta-objet représentant rc référence des méta-objets, instances de la méta-classe *Relationship Component*.

Chaque partie de la relation concrète est reliée à un objet concret, choisi par l'utilisateur parmi les composants de o. Ainsi, si o' (composant de o) est choisi pour jouer le rôle cr (composant de la relation r) dans rc, le méta-objet représentant cette partie sera relié d'une part au méta-objet représentant cr et au méta-objet représentant o'.

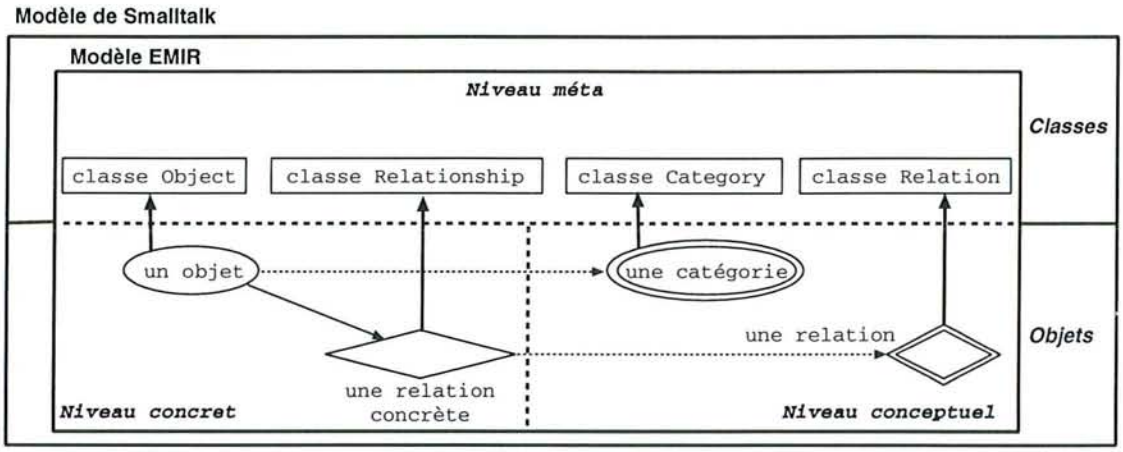


FIG. 4.13 – *Une vue partielle du méta-modèle (relations concrètes individuelles d'objets)*

Notons qu'il est possible d'ouvrir un browser concret uniquement à des fins de navigation dans la base d'objets concrets. Ce genre de browsers concrets est appelé *browser concret de navigation*. Toutes les opérations de création, mise à jour ou suppression d'objets y sont désactivées.

Pour ce qui concerne les contraintes d'intégrité, l'utilisateur peut en définir au niveau d'un objet concret particulier, en les saisissant au niveau du browser concret.

Un browser de contraintes (cf. figure 4.6) est ouvert et l'utilisateur spécifie sa contrainte de la même manière que pour une catégorie ou une relation (cf. 4.1.1), à l'exception que lorsqu'il fait appel au navigateur pour définir ses chemins valides, la source du chemin qu'il définit est l'objet concret lui-même et non sa catégorie.

Dans la version actuelle du prototype, l'utilisateur a la possibilité de demander au niveau du browser concret si les contraintes d'intégrité qu'il a définies sont vérifiées par un objet donné.

Normalement, il doit y avoir des mécanismes de maintien des contraintes d'intégrité [CRZNM88, BDT91, BM91, BS94] afin de garder la base dans un état cohérent en permanence. Cependant, cela ferait, à lui seul, l'objet de tout un sujet de recherche. Notre but, au niveau du prototype, était uniquement de montrer comment vérifier si les contraintes

d'intégrité définies par l'utilisateur sont satisfaites ou pas, *dans un état donné de la base.*

Le résultat d'une telle opération sur un objet donné est une liste de toutes les contraintes violées par cet objet (cf. figure 4.14). Ces contraintes peuvent avoir été définies :

- au niveau de l'objet lui-même,
- au niveau de la catégorie qu'il réalise, ou d'une de ses génériques,
- au niveau d'une relation utilisée par l'objet dans sa topologie, ou d'une de ses génériques.

FIG. 4.14 – *Contraintes violées par un objet concret*

4.1.3 Browser de requêtes

Lorsque l'utilisateur définit une requête q s'adressant à la catégorie c, un méta-objet, instanciant la méta-classe *Query* est créé.

En accord avec la définition des requêtes donnée dans le chapitre 3.2, ce méta-objet est relié :

- au méta-objet représentant c à travers une variable d'instance *category* de la méta-classe *Query*,
- aux méta-objets représentant les critères descriptifs de la requête à travers la variable d'instance *criteria* de la méta-classe *Query*,
- aux méta-objets représentant les sous-requêtes à travers la variable d'instance *sub-queries* de la méta-classe *Query*,
- aux méta-objets représentant les liens topologiques qui la définissent à travers la variable d'instance *links* de la méta-classe *Query*,
- au méta-objet représentant le chemin de projection à travers la variable d'instance *projection* de la méta-classe *Query*.

Encore une fois, les méta-classes représentant ce qui a trait aux requêtes n'ont pas été reportées dans le schéma de la figure 4.1, pour des soucis de simplification et de lisibilité.

Dans la partie de la fenêtre réservée à la saisie des critères descriptifs, le même navigateur utilisé pour la spécification des contraintes d'intégrité est utilisé. Il permet d'aider l'utilisateur à définir des chemins valides issus de la catégorie à laquelle s'adresse la requêtes. Ces chemins sont les briques de base des critères descriptifs, que doivent satisfaire en particulier les objets répondant à la requête.

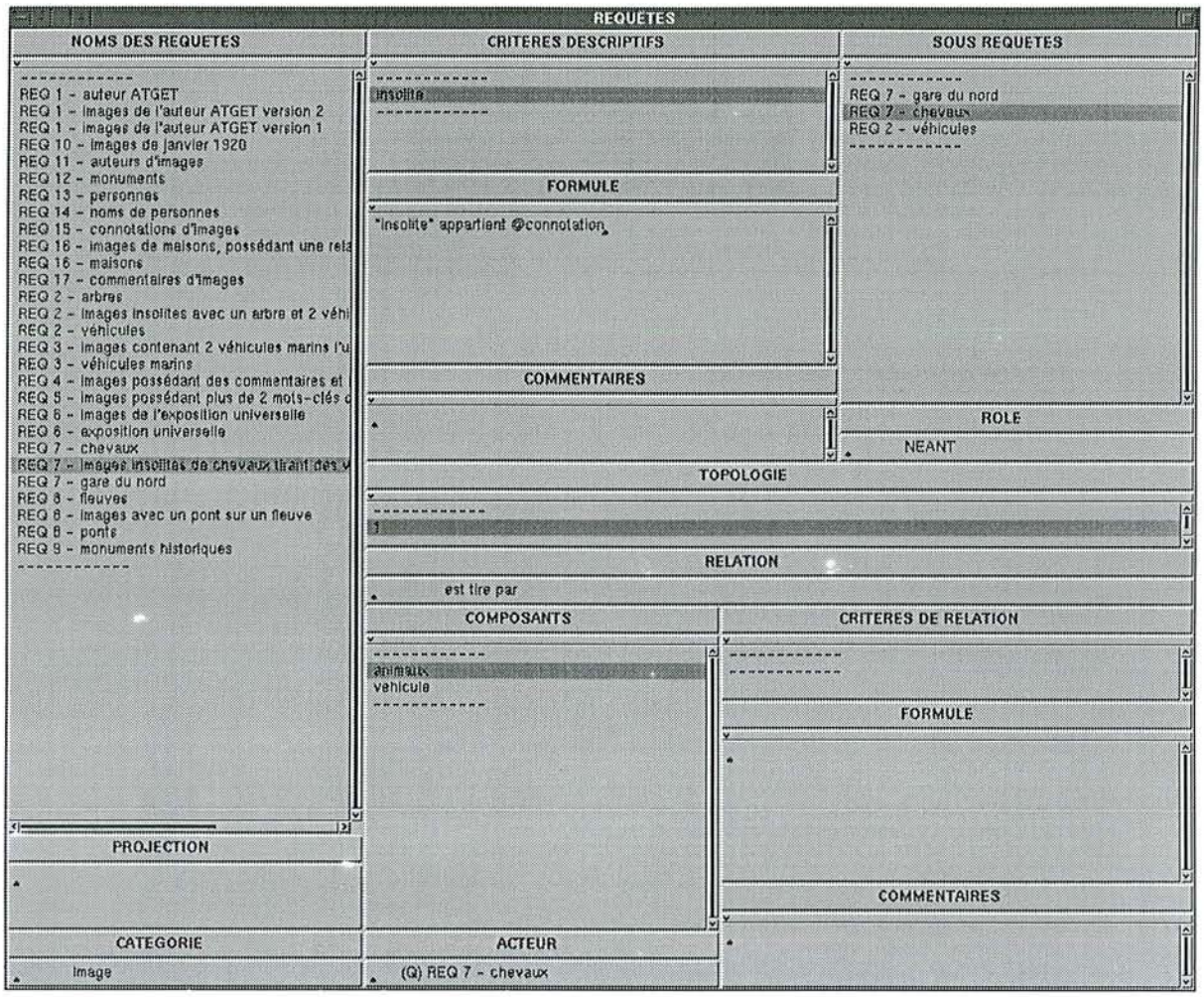


FIG. 4.15 – *Browser de requêtes*

Lorsque l'utilisateur exécute une requête, deux types de fenêtres peuvent afficher le résultat. Si le résultat est une liste d'objets concrets, un browser concret de navigation (cf. 4.1.2) réduit aux objets satisfaisant la requête est ouvert. Les objets résultats peuvent être explorés dans toute leur structure.

Si le résultat de la requête est simplement une liste de valeurs (chemin de projection s'arrêtant à un attribut), la liste des valeurs est affichée dans une fenêtre, comme l'indique la figure 4.16.

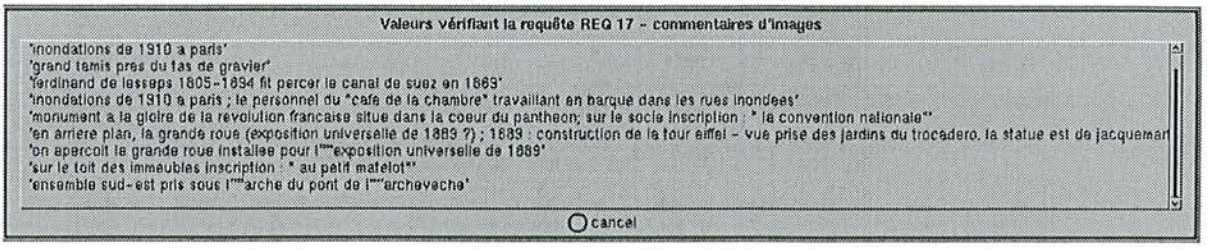


FIG. 4.16 – *Valeurs résultant d'une requête*

4.2 Modélisation d'images sous EMIR

Le prototype EMIR a été expérimenté sur une base d'images du vieux Paris des années 1900 [LAH95]. Cette base avait aussi servi de plateforme d'expérimentation pour le système de recherche d'information RIVAGE [HCK90].

Nous sommes partis de la base descriptive de RIVAGE (titre, nom de l'auteur et liste de mots-clés pour chaque image) et avons défini un schéma conceptuel EMIR permettant d'accueillir les diverses facettes de la description d'une image, dans un cadre bien structuré.

La figure 4.17 donne la description EMIR de la catégorie *Image*. La description conceptuelle d'une image en général est réduite à certains attributs (date de prise, connotations, ...) ainsi qu'à un composant (auteur).

Plus généralement, il a fallu créer une hiérarchie de catégories pour représenter, en objets, tout ce qui apparaissait dans les images. La figure 4.18 représente une hiérarchie créée en analysant une vingtaine d'images. Cette hiérarchie se développe évidemment au fur et à mesure qu'on décrit de nouvelles images. Cependant, la vitesse de croissance diminue au fur et à mesure de la construction de la hiérarchie. Au départ, beaucoup de catégories sont créées. Mais au fur et à mesure que la hiérarchie se développe, les nouvelles images prises en compte donnent lieu à de moins en moins de nouvelles catégories.

Lors de la description d'une image particulière, tout ce qui y apparait est alors traduit en objets composants de l'objet qui représente l'image. La figure 4.19 montre une partie de la description d'une image dans le formalisme d'EMIR.

La description des images s'est donc enrichie par la distinction de plusieurs types d'attributs d'images (connotation, morphologie, ...), par la possibilité de décrire le contenu

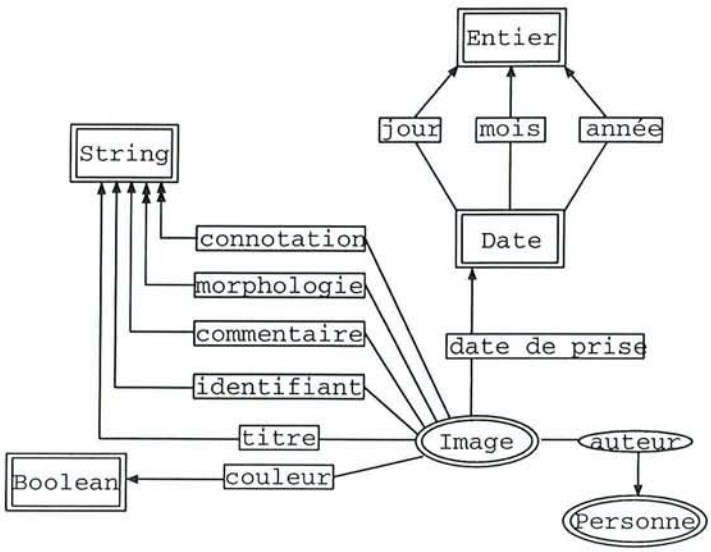


FIG. 4.17 – *La catégorie Image dans le formalisme EMIR*

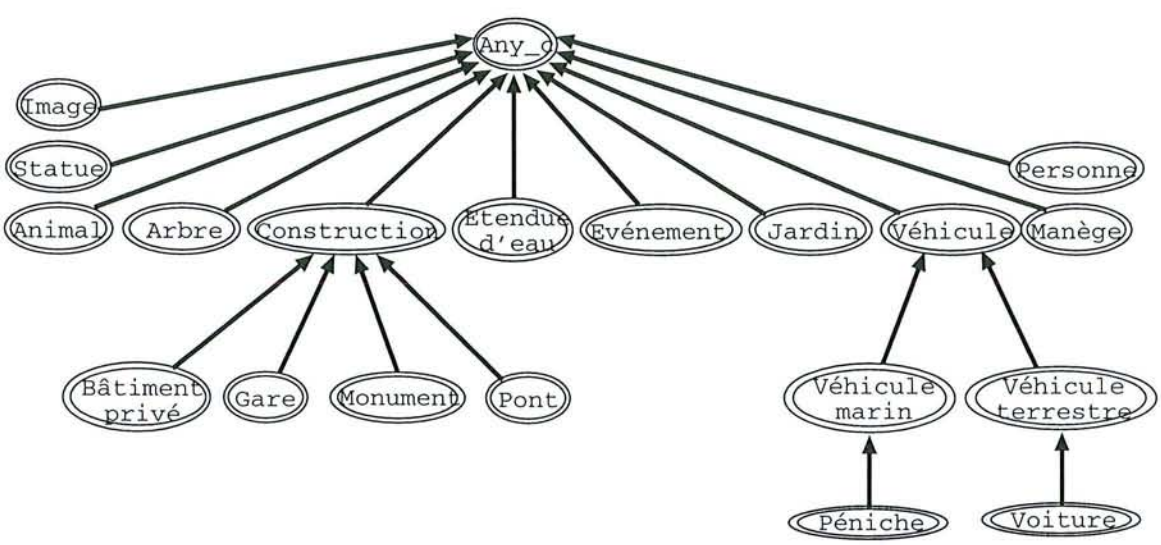


FIG. 4.18 – *Hiérarchie de catégories pour la modélisation d'images*

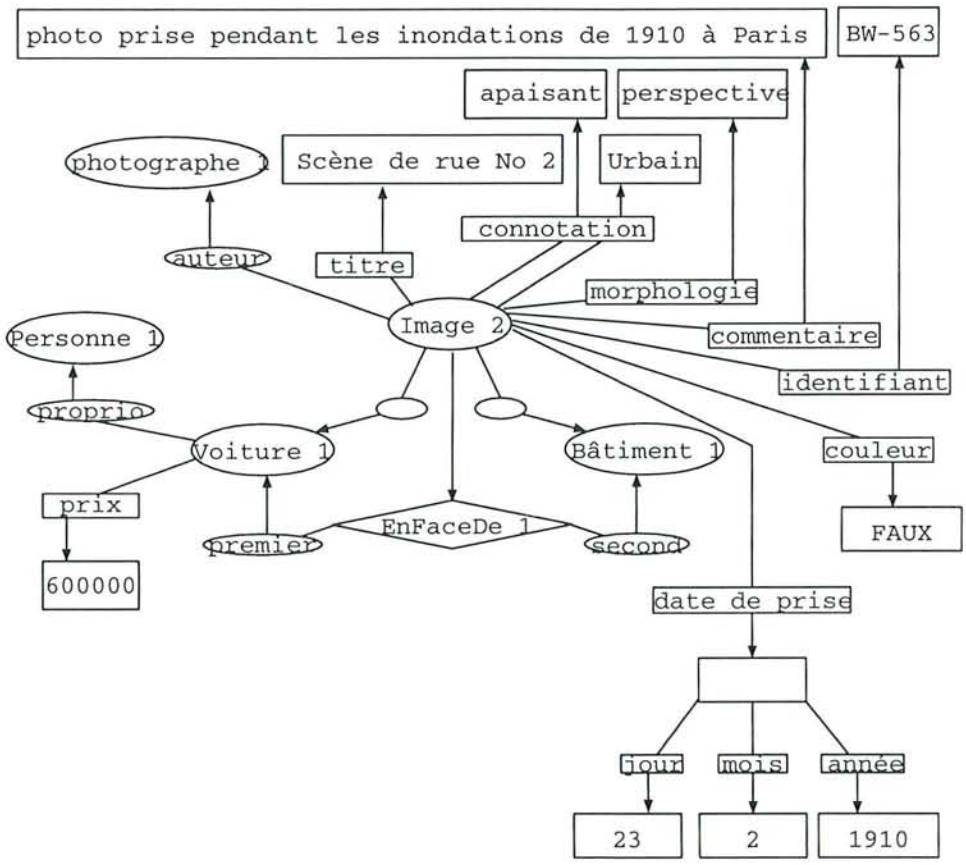


FIG. 4.19 – *Partie de la description d'une image dans EMIR*

par des objets qui peuvent à leur tour être décrits sous les trois angles, et enfin par la prise en compte de relations sémantiques (topologiques entre autre) reliant les composants de l'image.

Il y avait cependant des mots-clés de la description initiale qui ne pouvaient être facilement traduits dans le schéma conceptuel que nous avions préparé. Nous avons donc ajouté un attribut supplémentaire, appelé *mots-clés*, dans la classe *Image*. Cet attribut recevait tous les mots-clés de la description d'une image qui ne pouvaient être placés ailleurs dans la structure de l'objet EMIR décrivant l'image.

Ces mots-clés étaient cependant très rares et correspondaient souvent à des particularités de l'indexation par mots-clés récupérée de RIVAGE (comme des noms de saison, été, hiver, ..., des noms de matière, fer, bois, ...).

Cette expérimentation a clairement montré l'intérêt de décrire les objets sous les trois angles (description, composition, topologie) et de les munir de structures individuelles, dans le cadre de la recherche basée sur le contenu dans une base d'images.

Par comparaison avec les systèmes de recherche d'images où la description des images se fait par l'intermédiaire de mots-clés (comme RIVAGE), la description avec EMIR a apporté plus de précision dans le processus de description d'une part et dans le processus de recherche d'autre part.

La figure 4.20 donne par exemple le résultat de la recherche d'images insolites contenant un véhicule tiré par un cheval devant la gare du nord.

Nous avons utilisé un visionneur d'images, appelé *Mosaïque*, développé initialement pour RIVAGE [CHA93], pour mieux afficher les résultats des requêtes EMIR lorsque celles-ci s'adressent à des objets de type image [ISS95]. Ce visionneur est alors ouvert en plus du browser concret de navigation, afin d'avoir en parallèle les images résultantes de la requête et leur description.

Par ailleurs, nous avons développé [ISS95] un module d'affichage d'images qui permet de tenir compte des positions des différents composants dans l'image (cf. figure 4.21).

Ces positions relatives sont représentées dans EMIR par des relations concrètes de la topologie des images. Ces relations concrètes sont reliées à une relation, appelée *Position du composant*, qui lie un objet concret de type *Image* à un objet concret quelconque et qui possède un attribut multivalué appelé *points* renfermant les coordonnées des sommets d'un polygone. Ce polygone, saisi graphiquement par l'utilisateur, délimite la position d'un composant dans l'image.

Une expérience intéressante a été menée avec des étudiantes en DESS Audit et Conception de Systèmes d'Informations, qui n'étaient pas familiarisées avec le modèle EMIR. Le but était de savoir si celui-ci était d'une utilisation naturelle pour modéliser et rechercher des images.

Après leur avoir présenté le modèle par une démonstration du prototype et leur avoir fourni quelques publications à lire sur le modèle, nous leur avons fourni quelques images et leurs descriptions dans RIVAGE, sous forme de listes de mots-clés. Le travail demandé était alors de *traduire* et de *compléter* la description dans l'environnement EMIR, en

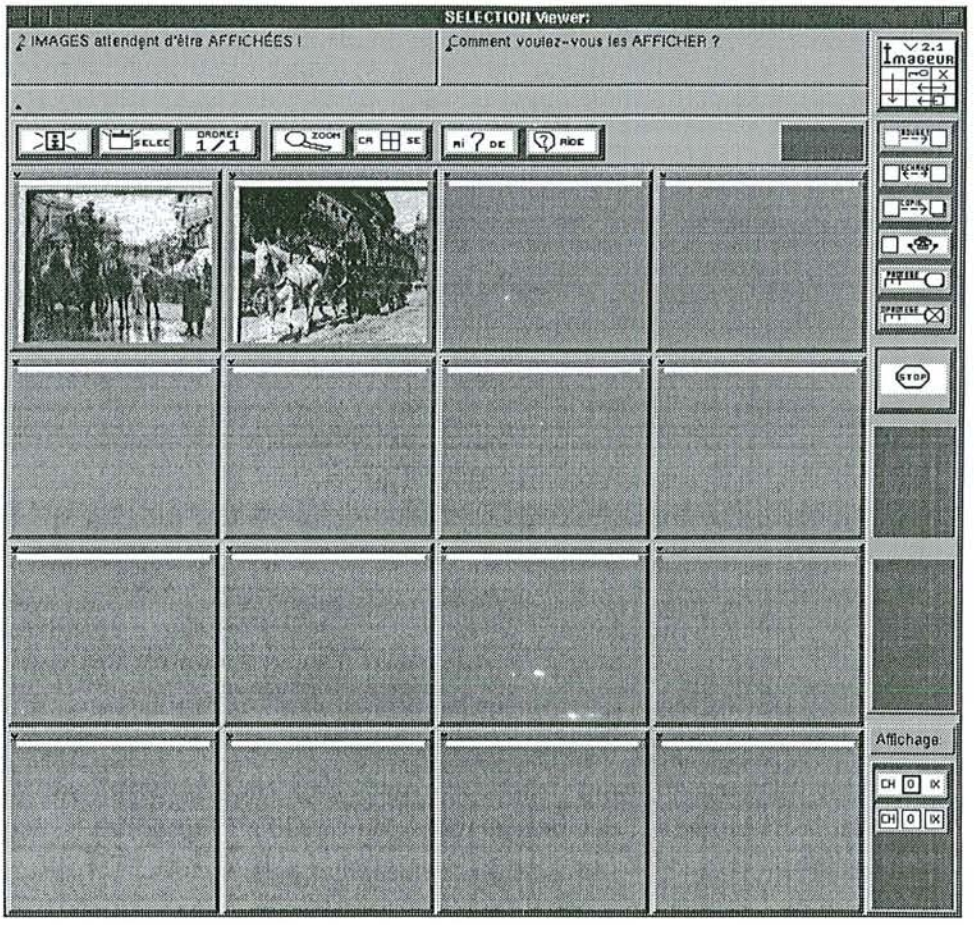


FIG. 4.20 – *Affichage d'images en mosaïque*

FIG. 4.21 – *Affichage individuel d'images avec positionnement des composants*

tenant compte d'un certain nombre de catégories et de relations déjà définies et en en rajoutant d'autres au besoin. Par ailleurs, il leur était demandé d'imaginer des requêtes susceptibles d'être posées à la base d'images constituée et de les réaliser dans le prototype EMIR.

L'expérience a été concluante, dans la mesure où les deux étudiantes ont rapidement appréhendé la technique en modélisant les images fournies et ont facilement formulé un certain nombre de requêtes variées.

4.3 L'application architecturale

Le but de cette application menée en collaboration avec le Centre de Recherche en Architecture et Ingénierie (CRAI) de Nancy est de modéliser des projets architecturaux en cours d'élaboration, et ceci pour différents objectifs.

D'abord, il s'agit d'assister le concepteur de projet dans son processus de modélisation, en lui fournissant des mécanismes de vérification de la cohérence. Cette vérification repose sur des contraintes d'intégrité définies par l'utilisateur selon la technique présentée dans la partie 3.1.2. La spécification des contraintes d'intégrité sémantique qui y a été présentée a d'ailleurs été largement inspirée de cette application.

Ensuite, il s'agit de construire une base de projets architecturaux qui servira de référence pour une étude plus vaste, dont l'objectif est de réutiliser l'expérience acquise dans des projets antérieurs, lors de l'élaboration d'un nouveau projet.

Cette application était intéressante sous plusieurs points de vue. D'abord, les projets architecturaux ont ceci de commun avec les images : leur structure n'est pas facilement prévisible à un niveau conceptuel (dans des classes fixant la structure de leurs objets reliés). Cependant, ils s'en distinguent par le fait que le nombre de catégories d'objets est limité. Il est possible d'énumérer les différents types d'objets composant un projet (on sait qu'il va être composé de murs, éventuellement de parois intérieures, de baies, ...), mais pas la structure exacte d'un projet, d'une manière générale (chaque projet a ses propres spécificités).

A la différence de la modélisation des images, la hiérarchie des catégories de l'application architecturale est assez limitée et peut être fixée avant de commencer la saisie des projets. Les types d'objets manipulés ne prolifèrent pas à mesure que de nouveaux projets sont saisis, comme c'était le cas pour les images. La figure 4.22 donne la hiérarchie utilisée pour saisir quelques projets, pour l'application avec le CRAI.

L'une des particularités de l'application architecturale réside dans le fait que les relations sémantiques utilisées se prêtent particulièrement bien à la représentation hiérarchique du modèle EMIR, décrite dans la partie 2.3.1.2, qui traite du lien d'héritage entre relations sémantiques.

La figure 4.23 donne la hiérarchie de relations utilisée pour l'application architecturale. Cet exemple montre que le concept d'héritage entre relations peut être tout aussi utile que l'héritage entre classes (catégories dans notre cas).

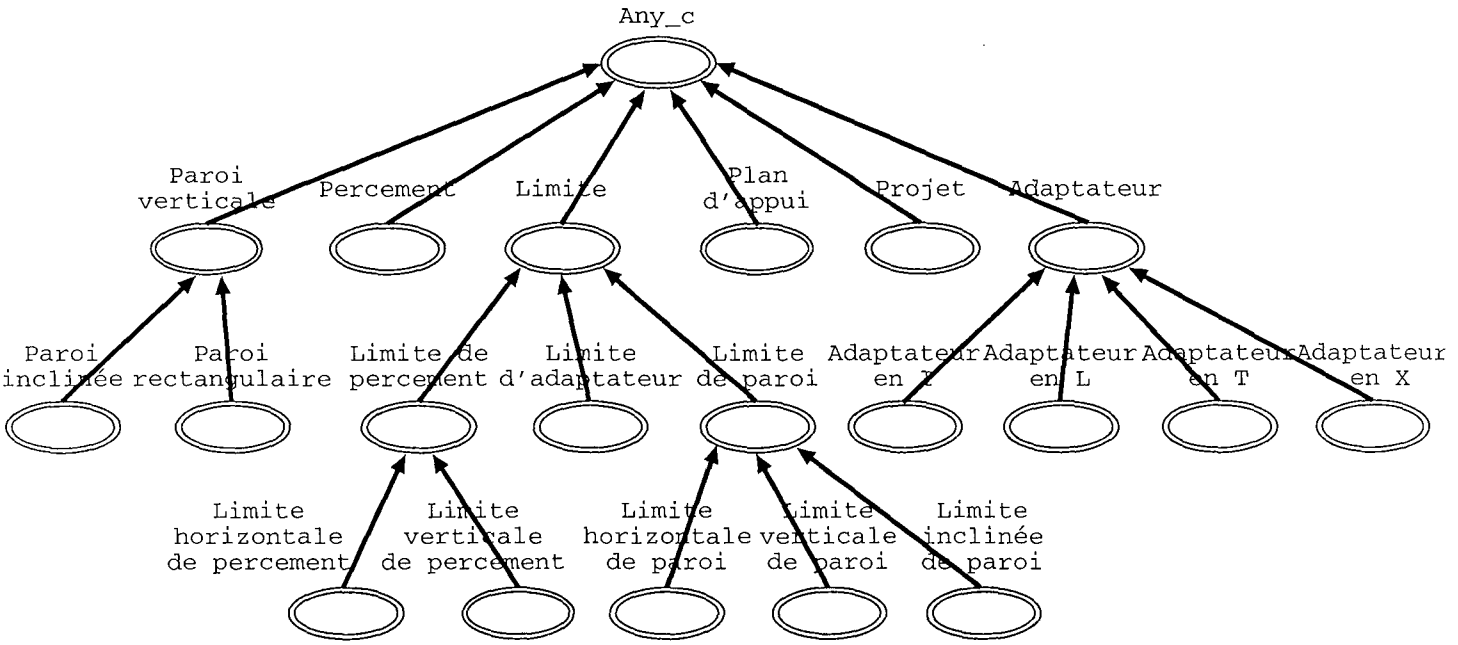


FIG. 4.22 – *Hiérarchie des catégories de l'application architecturale*

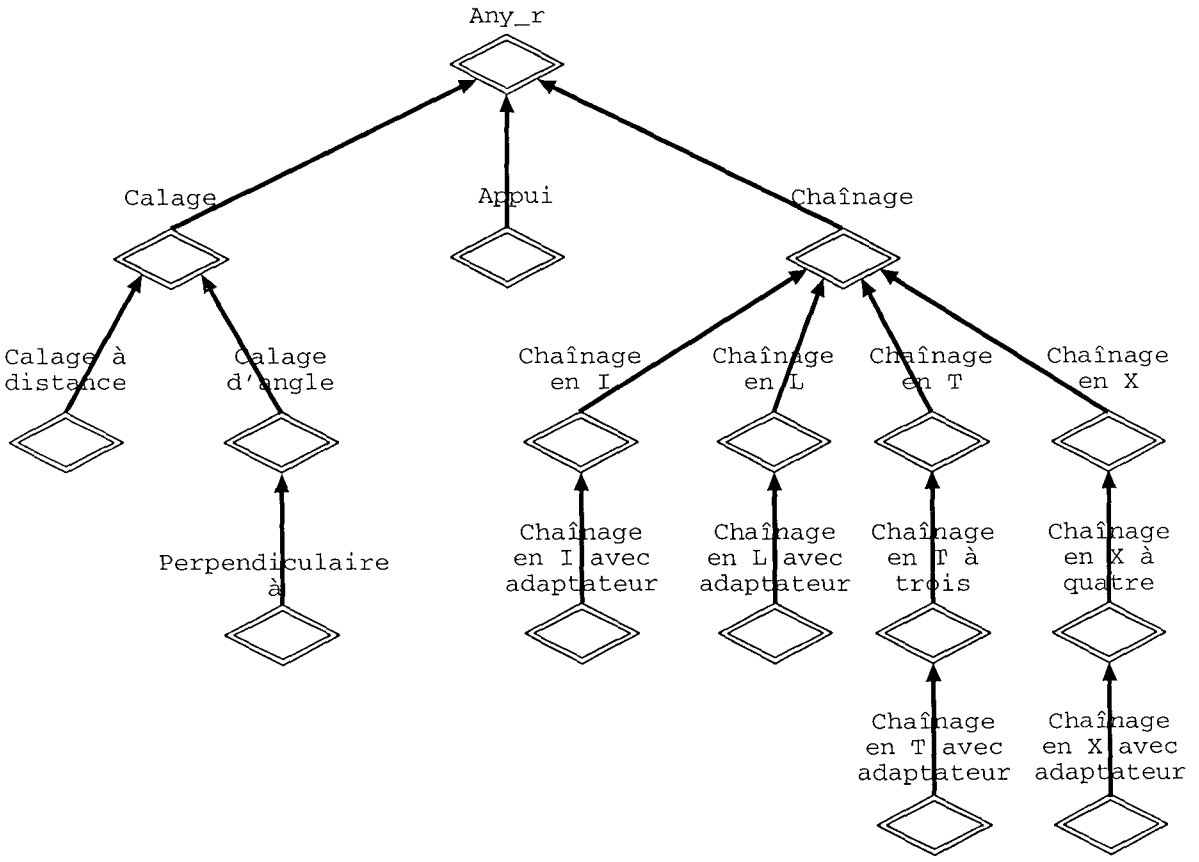


FIG. 4.23 – *Hiérarchie des relations de l'application architecturale*

Par exemple, les relations de chaînage expriment en général que deux ou plusieurs parois verticales se rencontrent en un axe (vertical) de l'espace (figure 4.24).

Le chaînage en I exprime que, vues de haut, les silhouettes des parois forment une ligne continue : l'une est le prolongement de l'autre.

Le chaînage en L exprime que les silhouettes forment un angle droit.

Ces deux types de chaînage mettent en jeu exactement deux parois.

Le chaînage en T exprime que les silhouettes forment un "T". Ce chaînage peut mettre en jeu deux ou trois parois.

Enfin, le chaînage en X exprime que les silhouettes forment une croix. Ce chaînage peut mettre en jeu jusqu'à quatre parois.

Ces différentes relations représentent des spécialisations successives de la relation générale de chaînage, auxquelles viennent s'adjoindre d'autres spécialisations liées à la présence d'objets architecturaux spéciaux - les adaptateurs - entre les parois chaînées.

La définition du lien d'héritage entre relations permet, à l'instar de l'héritage entre catégories de partager l'information, économiser les redéfinitions inutiles ... Par exemple, le fait de définir une contrainte d'intégrité sur la relation générique *Chaînage* qui stipule que les parois liées doivent avoir la même hauteur, permet de partager cette contrainte entre toutes les relations de chaînage spécifiques.

Comme pour la modélisation des images, le lien de réalisation a montré tout son intérêt lors de la modélisation de projets architecturaux. La figure 4.25 montre que la définition conceptuelle d'un projet architectural est assez réduite. Le concepteur de l'application spécifie seulement les structures qui lui paraissent évidentes pour un objet de type *Projet.*

Un projet architectural n'est vraiment saisi avec toutes ses particularités qu'au niveau concret. Un exemple est donné dans la figure 4.26 qui montre une partie de la modélisation d'un projet concret avec EMIR.

Le premier objectif de l'application architecturale qui concerne la vérification de la cohérence des projets a été exploré dans un premier travail [ISS95]. Le second qui concerne, à plus long terme, la réutilisation de l'expérience est laissé à une collaboration future.

La vérification de la cohérence des projets architecturaux se fait en plusieurs phases (figure 4.27). Elle repose sur une coopération entre un logiciel de saisie graphique de projets : arTec, développé au CRAI et le prototype EMIR. Au cours de l'élaboration d'un projet architectural à l'aide du modeleur graphique arTec, l'utilisateur déclenche à certains instants la procédure de vérification de la cohérence sur la version courante de sa modélisation.

Le concepteur saisit graphiquement un projet dans l'environnement d'arTec. Lors de l'appel à la procédure de vérification de contraintes, ce logiciel génère un fichier de descriptions des entités saisies (fichier ASCII). Ces descriptions sont ensuite passées par un module d'interfaçage avec EMIR, qui génère un schéma concret EMIR, à partir du fichier de descriptions arTec et du schéma conceptuel général de l'application architecturale. Ce dernier schéma comprend en particulier des contraintes définies au niveau des catégories ou des relations. Une première vérification de la cohérence concerne ces contraintes.

L'utilisateur peut ensuite définir des contraintes d'intégrité sur les objets eux-mêmes.

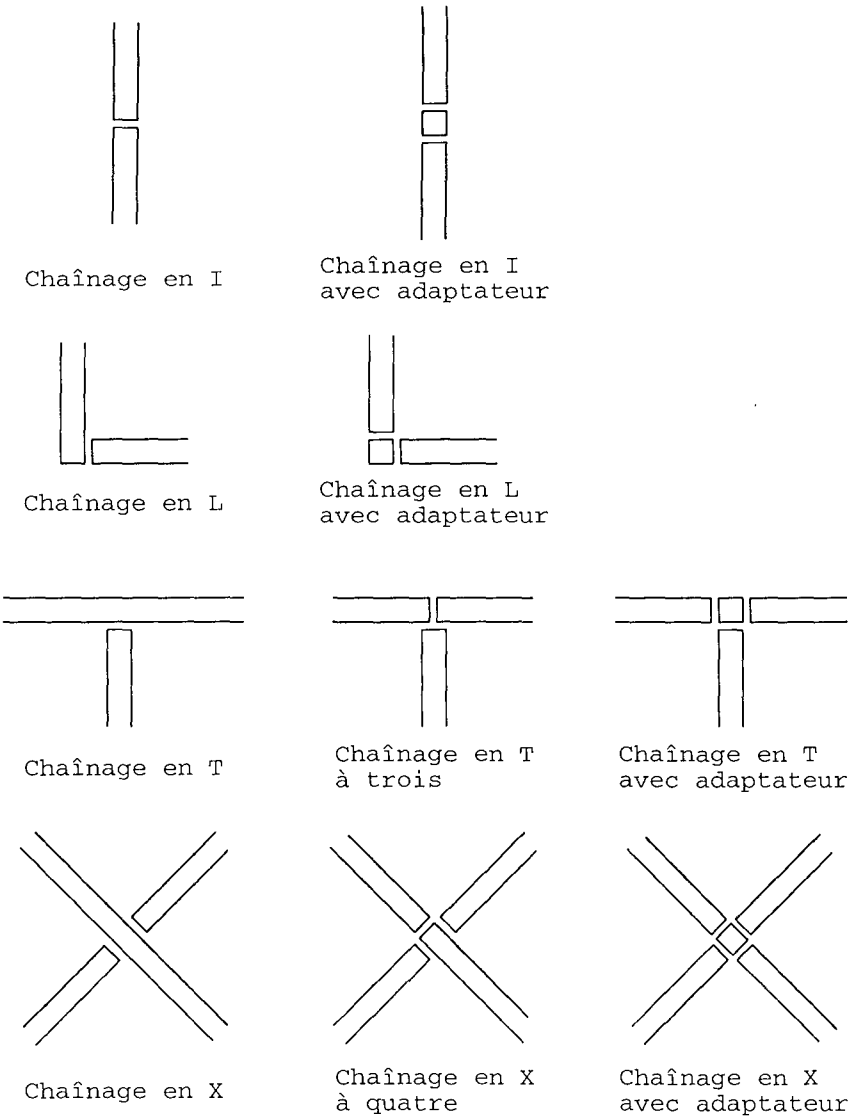


FIG. 4.24 – *Représentation graphique des relations de chaînage entre parois verticales*

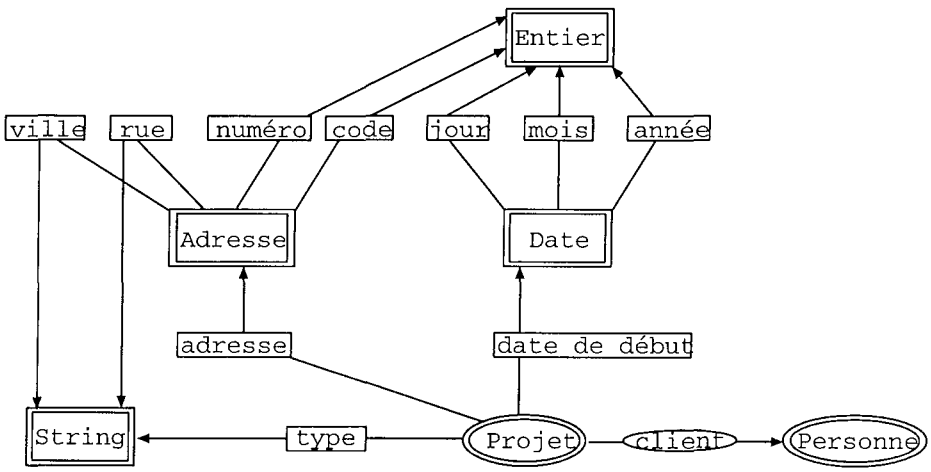


FIG. 4.25 – *La catégorie Projet issue de l'application architecturale*

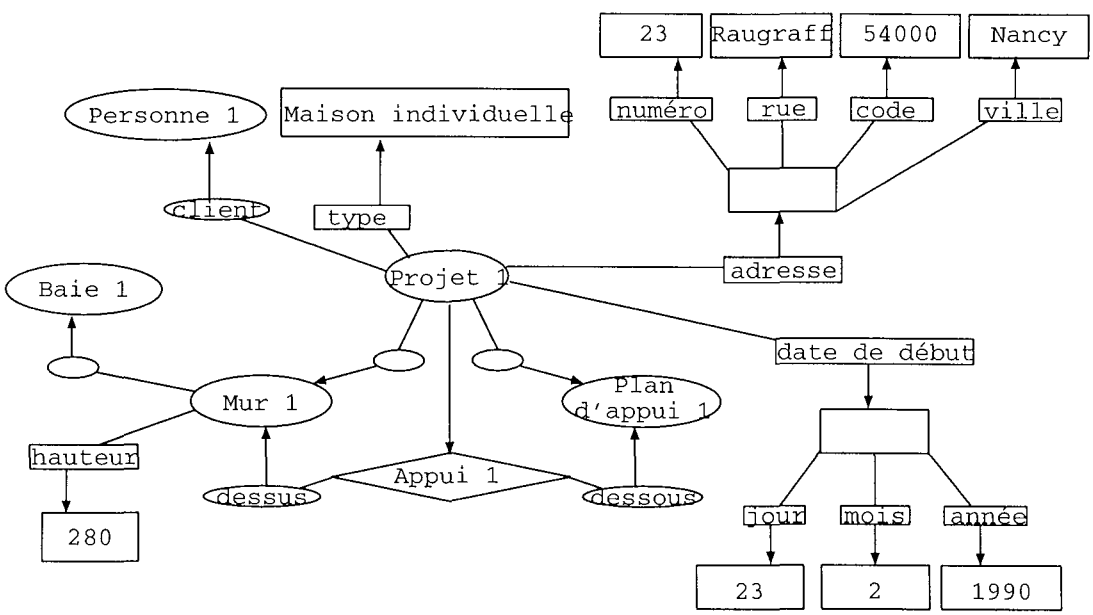


FIG. 4.26 – *Partie de la description d'un projet architectural dans EMIR*

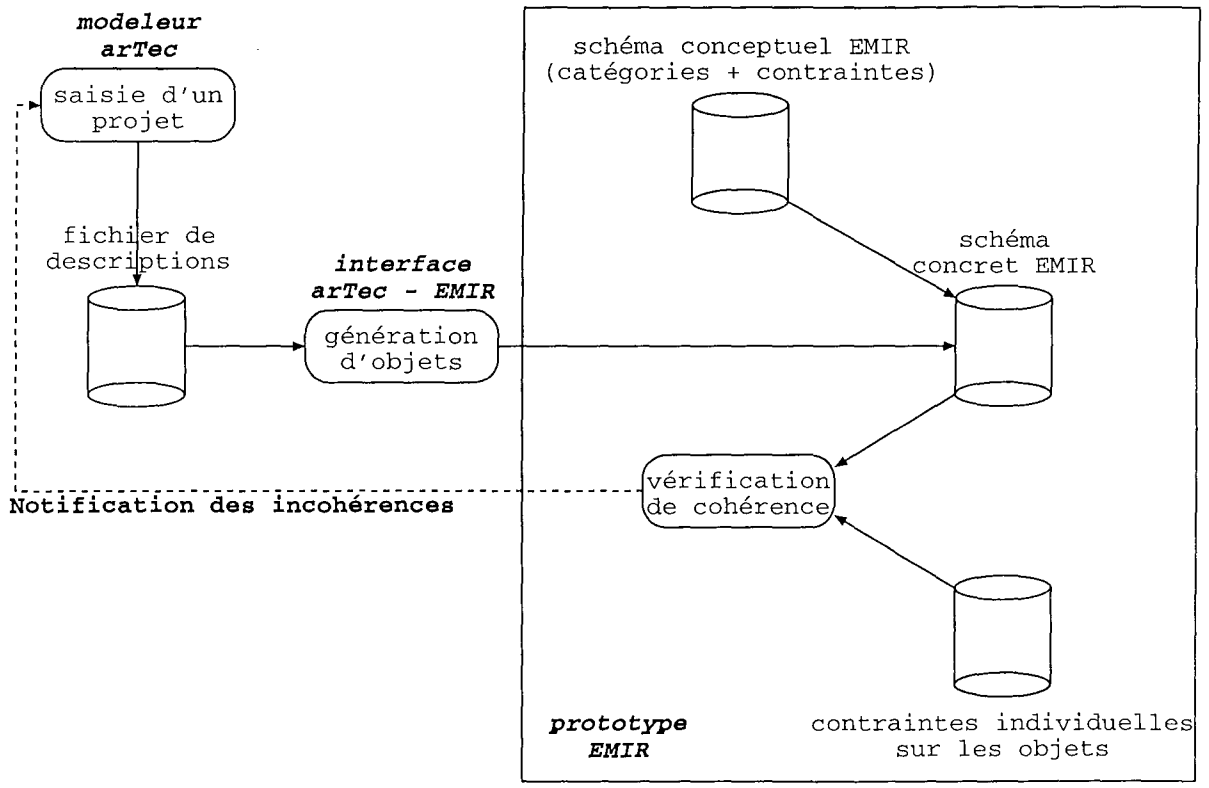


FIG. 4.27 – *Phases de vérification de cohérence d'un projet architectural*

Par exemple, s'il désire garder une certaine distance entre deux murs particuliers au cours des évolutions futures du projet (comme ça a été expliqué dans 3.1.2.3, figure 3.5), il définit une contrainte attachée à un certain objet concret. Cette contrainte devra être respectée lors de futurs appels à la procédure de vérification de la cohérence. Si les murs y sont trop rapprochés, l'utilisateur est notifié du fait que cette contrainte a été violée.

L'application architecturale était intéressante de plusieurs points de vue. D'abord, elle a montré la possibilité d'utiliser le modèle EMIR pour des applications réelles. Elle a appuyé l'intérêt du lien de réalisation. Elle nous a beaucoup inspiré pour la définition du langage de contraintes. De plus, elle a permis de montrer (avec la modélisation d'images) qu'EMIR permet de réaliser des applications variées.

Conclusion

Les nouvelles applications des bases de données nécessitent de plus en plus de flexibilité au niveau du modèle de données. L'approche orientée objet des bases de données, basée sur le modèle des classes, résout certes un grand nombre de problèmes de modélisation d'objets complexes, mais le paradigme des classes représente une contrainte trop forte pour la modélisation d'objets à forte structure individuelle, comme c'est le cas pour les bases documentaires.

Partis de l'idée que des modèles plus souples au niveau du schéma d'application sont à définir afin de ne pas trop contraindre le processus de conception de ce type de bases de données, nous avons présenté un modèle de données dédié à la modélisation d'objets complexes à forte structure individuelle : le modèle EMIR (Extended Model for Information Retrieval).

Ce modèle a pour but de contribuer à la conception de nouvelles bases de données où les objets n'ont pas des structures figées à un niveau conceptuel. De ce point de vue, les bases de données documentaires (bases d'images en particulier) constituent un excellent domaine d'application.

Le modèle EMIR s'inspire de travaux effectués dans le domaine des langages de programmation par objets, qui définissent des modèles hybrides se situant à mi-chemin entre les modèles de classes et les modèles de prototypes. Ces approches hybrides utilisent des classes, sans que celles-ci contiennent la structure exacte des objets qui leur sont reliés.

Les objets du modèle EMIR sont regroupés en catégories, qui permettent des accès groupés aux objets. Une catégorie contient une structure minimale que possèdent tous les objets qui lui sont reliés. Chaque objet est libre de compléter cette structure minimale par une partie individuelle. Ce lien particulier entre objets et catégories d'objets est appelé lien de réalisation (cf. chapitre 2).

Les objets EMIR ont une structure tri-partite : la description d'un objet est un ensemble de valeurs d'attributs, sa composition est un ensemble d'objets composants et sa topologie est un ensemble de liens entre ses composants. Cette dernière partie de la structure d'un objet est particulièrement utile pour la modélisation d'images et pour la prise en compte de liens entre objets en général, quand ces liens n'ont d'existence que dans le cadre de la composition d'un objet global.

Le maintien de l'intégrité d'une base EMIR en général, indépendamment des contraintes de chaque application, nous a conduit à spécifier de manière opérationnelle la réaction de la base à chaque opération élémentaire de la part de l'utilisateur (cf. chapitre 3 et annexe B).

De plus, nous avons défini, dans le chapitre 3, un langage de spécification de contraintes d'intégrité sémantiques (dépendantes des applications) particulièrement adapté à gérer la structure tri-partite des objets EMIR, ainsi que le lien de réalisation.

Ces deux caractéristiques du modèle sont aussi prises en compte dans le langage de requêtes, également défini dans le chapitre 3, et qui permet de formuler des requêtes s'adressant aussi bien à la structure des objets qu'à leur contenu.

Ces fonctionnalités de maintien de l'intégrité et de recherche dans une base sont fondamentales dans le domaine des bases de données. S'il est vrai que la plupart des notions introduites dans le domaine de la modélisation sémantique de bases de données trouvent leur origine dans le monde de la représentation des connaissances pour le raisonnement en intelligence artificielle, ce genre de fonctionnalités est particulier au domaine de la gestion de bases de données.

Pour ce qui concerne le modèle EMIR, il est possible de retrouver l'une ou l'autre de ses caractéristiques dans le domaine de la représentation de connaissances (instanciation flexible que nous appelons ici *réalisation*, liens sémantiques entre objets, ...). Des travaux autour des réseaux sémantiques et des graphes conceptuels, notamment, voire des logiques de description, se sont également intéressé à l'une ou l'autre des fonctionnalités offertes par le modèle.

Notre approche ici était résolument tournée vers le domaine des bases de données, ce qui a donné lieu à l'étude des fonctionnalités de maintien de l'intégrité et d'interrogation et qui doit être complété par l'étude d'autres aspects des bases de données comme l'indexation ou la gestion des accés concurrents.

Le modèle EMIR a été validé dans le cadre d'un prototype réalisé en Smalltalk-80, et expérimenté sur deux applications assez différentes : la modélisation sémantique d'images par leur contenu et la modélisation de projets architecturaux (cf. chapitre 4).

Cette étape de validation est très importante dans la définition de nouveaux modèles de données. Dans notre cas, elle a permis de montrer que le modèle que nous proposons est bien fondé et bien adapté à des applications variées, issues aussi bien du domaine de la recherche d'information que du domaine des bases de données pour la Conception Assistée par Ordinateur, voire même d'autres domaines ayant besoin de modéliser des objets à structures très hétérogènes.

Le prototype réalisé montre, de par sa taille (plus de 120 classes Smalltalk et plus de 20 000 lignes de code), que la solution que nous proposons pour gérer ce genre de populations d'objets n'est pas triviale et qu'elle mérite d'être étendue par des fonctionnalités permettant à terme d'aboutir à un produit commercialisable, ce qui est pour nous un objectif important.

A ce propos, le choix de l'environnement de développement (Smalltalk-80, en l'occurence) a été dicté par l'expérience de ce langage acquise dans l'équipe et par la facilité qu'offre l'environnement de Smalltalk-80 pour définir des prototypes d'applications, à travers le paradigme MVC (Model - View - Controller) permettant de définir rapidement des interfaces efficaces pour des applications orientées objet.

La nature semi-interprétée du langage le rend cependant peu adapté à des applications commercialisables, pour des raisons de performance. A ce titre, un langage orienté objet compilé (Self ou C++) pourra être choisi pour reprogrammer le prototype.

Un second type d'extension du système est ébauché dans l'annexe C ; il concerne la gestion de larges quantités d'objets. Pour ce faire, il nous semble nécessaire d'intégrer nos propositions dans un système de gestion de bases de données orientées objet, afin par

exemple, d'assurer la persistence des objets et les accès concurrents à une base EMIR.

Dans cette optique, nous avons présenté une technique pour traduire les bases de données et les requêtes EMIR dans un langage normalisé d'interrogation de bases de données orientées objet : le langage OQL défini par l'ODMG en 1993.

L'application d'EMIR à la modélisation sémantique d'images par leur contenu a permis de montrer l'adéquation du modèle à ce genre d'applications qui, rappelons-le, constituait la motivation première du projet. La vingtaine d'images décrites jusqu'à présent a donné lieu à la création de plus de 250 objets (images et descripteurs confondus).

Quant à l'application architecturale, elle a montré l'adéquation du modèle à d'autres applications que celles auxquelles il était initialement destiné (celles issues du domaine de la recherche d'information). Elle a d'ailleurs fait ressentir le besoin de gérer des quantités importantes d'objets, dans la mesure où, par exemple, un seul projet architectural, assez simple (maison individuelle avec un seul étage, représentée dans la figure 3.5) a donné lieu à la création de plus de 400 objets.

Le lien de réalisation s'est montré très intéressant dans ce genre d'applications aussi. Comme le soulignait la thèse de Palisser [PAL89], la gestion des objets CAO (en architecture par exemple) nécessite un modèle fournissant une certaine flexibilité dans la gestion des schémas d'application. Le processus de conception est progressif et le schéma d'une application est obtenu à la suite d'un certain nombre de tentatives. Certaines propriétés des objets ne sont pas connues à une certaine étape et d'autres, connues, sont remises en cause ultérieurement.

Grâce au lien de réalisation, EMIR supporte une conception très incrémentale, dans la mesure où le schéma conceptuel d'une application peut être réduit à un minimum et éventuellement affiné ultérieurement.

Il nous semble utile de définir, dans le futur, des procédures permettant de tels affinements, à partir d'un schéma conceptuel minimal et d'une base d'objets avec leurs structures individuelles. L'affinement consisterait à alimenter le schéma conceptuel avec des structures issues de l'analyse du contenu réel des objets.

Ces procédures doivent, à notre avis, faire partie d'un processus global (à définir) pour la conception d'une application EMIR.

Enfin, signalons que nous nous intéressons aussi à la représentation multiple d'objets [NM95, NLC96]. Une de nos perspectives est d'enrichir le modèle EMIR avec ce paradigme.

Bibliographie

[AB86] ABITEBOUL (S.) et BIDOIT (N.). – Non First Normal Form Relations : an algebra allowing data restructuring. *Journal of Computer and System Sciences*, vol. 33, n3, 1986, pp. 361–393.

[AB88] ABITEBOUL (S.) et BEERI (C.). – *On the Power of Languages for the Manipulation of Complex Objects.* – Rapport technique nRR-0846, INRIA, 1988.

[ACO85] ALBANO (A.), CARDELLI (L.) et ORSINI (R.). – Galileo : A Strongly-Typed, Interactive Conceptual Language. *ACM Transactions on Database Systems*, vol. 10, n2, 1985, pp. 230–260.

[AF94] AMGHAR (Y.) et FLORY (A.). – Contraintes d'intégrité dans les bases de données orientées objet. *Ingénierie des Systèmes d'Informations*, vol. 2, n5, 1994, pp. 507–532.

[AH87] ABITEBOUL (S.) et HULL (R.). – IFO : A Formal Semantic Database Model. *ACM Transactions on Database Systems*, vol. 12, n4, 1987, pp. 525–565.

[AK89] ABITEBOUL (S.) et KANELLAKIS (P.C.). – Object Identity as a Query Language Primitive. *Dans : ACM-SIGMOD'89*, pp. 159–173. – 1989.

[ASL89] ALASHQUR (A.M.), SU (S.Y.W.) et LAM (H.). – OQL : A Query Language for Manipulating Object-Oriented Databases. *Dans : Very Large Databases, VLDB'89*, pp. 433–442. – 1989.

[AV89] ABITEBOUL (S.) et VIANU (V.). – A Transaction-Based Approach to Relational Database Specification. *Journal of the ACM*, vol. 36, n4, 1989, pp. 758–789.

[BCG+87] BANERJEE (J.), CHOU (H.), GARZA (J.F.), KIM (W.), WOELK (D.), BALLOU (N.) et KIM (H.). – Data Model Issues for Object-Oriented Applications. *ACM Transactions on (Office) Information Systems*, vol. 5, n1, 1987, pp. 3–26.

[BCMS93] BRUNET (J.), CAUVET (C.), MEDDAHI (D.) et SEMMAK (F.). – Object-Oriented Analysis in Practice. *Dans : Conference on Advanced Information*

Systems Engineering, CAiSE'93. pp. 293–308. – Springer-Verlag, Lecture Notes in Computer Science, No 685, 1993.

[BDT91] BRIALES (M.J.) et DE TROYER (O.). – Object-Oriented Integrity Enforcement in a Relational Environment. *Dans : British National Conference on Databases, BNCOD'91*, pp. 38–68. – Wolverhampton, UK, 1991.

[BK86] BANCILHON (F.) et KHOSHOFIAN (S.). – A Calculus for Complex Objects. *Dans : ACM Symposium on Principles of Database Systems, PODS'86*, pp. 53–60. – 1986.

[BLM95] BONANNO (N.), LAHLOU (Y.) et MOUADDIB (N.). – Les liens inter-objets : Discussion et proposition de règles pour les identifier. *Dans : Congrès Inforsid'95.* – Grenoble, France, 1995.

[BLMN94] BONANNO (N.), LAHLOU (Y.), MOUADDIB (N.) et NAJA (H.). – *Les méthodes de conception orientées objet. Vers une méthode dédiée EMIR.* – Rapport technique n94-R-144, CRIN - CNRS, Nancy, France, 1994.

[BM91] BOUZEGHOUB (M.) et METAIS (E.). – Semantic Modeling of Object Oriented Databases. *Dans : Very Large Databases, VLDB'91*, pp. 3–14. – Barcelona, Spain, 1991.

[BOO91] BOOCH (G.). – *Object Oriented Design with Applications.* – Benjamin / Cumings, 1991.

[BOR86] BORNING (A.H.). – Classes Versus Prototypes in Object-Oriented Languages. *Dans : ACM / IEEE Fall Joint Computer Conference*, pp. 36–40. – 1986.

[BRI92] BRIFFAULT (X.). – Modélisation informatique de l'expression de la localisation en langage naturel. – Thèse de l'Université de Paris VI, 1992.

[BS94] BOUZEGHOUB (M.) et SIMON (E.). – Integrity Specification and Enforcement in Databases. *Ingénierie des Systèmes d'Informations*, vol. 2, n 2, 1994, pp. 225–246.

[cac90] Communications of the ACM. – Special issue on Hypertexts, 1990. Vol. 33, No. 3.

[CAT94] CATTELL (R.G.G.) (édité par). – *The Object Database Standard: ODMG-93.* – Morgan Kaufmann, 1994.

[CCJ+93] CHRISMENT (C.), COMPAROT (C.), JULIEN (C.), LAMBOLEZ (P.Y.) et SEDES (F.). – Hyperdocument Structuration, Management and Exchange in an Object-Oriented database. *Dans : Symposium on Document Analysis and Information Retrieval.* – Las Vegas - Nevada, 1993.

[CDV88] CAREY (M.J.), DeWITT (D.J.) et VANDENBERG (S.L.). – A Data Model and Query Language for EXODUS. *Dans : ACM-SIGMOD'88*, pp. 413–423. – 1988.

[CHA93] CHAFFIOL (D.). – RIVAGE et IMAGEUR. – Rapport de stage ESIAL 2, Univ. Nancy I, 1993.

[CHE76] CHEN (P.P.S.). – The Entity - Relationship Model. Toward a Unified View of Data. *ACM Transactions on Database Systems*, vol. 1, n1, 1976, pp. 9–36.

[CM84] COPELAND (G.) et MAIER (D.). – Making Smalltalk a Database System. *Dans : ACM-SIGMOD'84*, pp. 316–325. – 1984.

[COD70] CODD (E.F.). – A relational Data Model for Large Shared Data Banks. *Communications of the ACM*, vol. 13, n6, 1970, pp. 377–387.

[COD79] CODD (E.F.). – Extending the Relational Model to Capture More Meaning. *ACM Transactions on Database Systems*, vol. 4, n4, 1979, pp. 397–434.

[COL90] COLBY (L.S.). – A Recursive Algebra for Nested Relations. *Information Systems*, vol. 15, n5, 1990, pp. 567–582.

[CON87] CONKLIN (J.). – Hypertext: An Introduction and Survey. *IEEE Computer*, vol. 20, n9, 1987, pp. 17–41.

[CP92] CORTE (P.) et PRESENZA (D.). – Understanding Data Behavior from its Static Structure. *Dans : The 5th International Conference on Putting into Practice Methods and Tools for Information System Design*, pp. 291–302. – Nantes, France, 1992.

[CPC94] COMPAROT-POUSSIER (C.) et CHRISMENT (C.). – Hyperbase pour la Gestion Electronique de Documents Techniques. *Ingénierie des Systèmes d'Informations*, vol. 2, n5, 1994, pp. 533–570.

[CQ92] COOPER (R.) et QIN (Z.). – A graphical Data Modelling Program with Constraint Specification and Management. *Dans : British National Conference on Databases, BNCOD'92.* pp. 192–208. – Springer-Verlag, Lecture Notes in Computer Science, No 618, 1992.

[CRZNM88] CHUNG (K.), RIOS-ZERTUCHE (D.), NIXON (B.) et MYLOPOULOS (J.). – Process Management and Assertion Enforcement for a Semantic Data Model. *Dans : Extending Database Technology, EDBT'88.* pp. 469–487. – Springer-Verlag, Lecture Notes in Computer Science, No 303, 1988.

[CUCH91] CHAMBERS (C.), UNGAR (D.), CHANG (B.W.) et HÖLZLE (U.). – Parents are Shared Parts of Objects: Inheritance and Encapsulation in SELF. *LISP and Symbolic Computation*, vol. 4, n3, 1991, pp. 207–222.

[CYDT88] CHANG (S.K.), YAN (C.W.), DIMITROFF (D.C.) et T. (ARNDT). – An Intelligent Image Database System. *IEEE Transactions on Software Engineering*, vol. 14, n5, 1988, pp. 681–688.

[DA82] DELOBEL (C.) et ADIBA (M.). – *Bases de données et systèmes relationnels.* – Dunod Informatique, 1982.

[Da90] DEUX (O.) et al. – The Story of O_2. *IEEE Transactions on Knowledge and Data Engineering*, vol. 2, n1, 1990, pp. 91–108.

[DES90] DESFRAY (P.). – A Method for Object Oriented Programming: The Class - Relationship Method. *Dans : Technology of Object-Oriented Languages and Systems, TOOLS'90*, pp. 121–131. – 1990.

[DJE94] DJERABA (C.). – Objets composites dans un modèle à objets. *Dans : Bases de Données Avancées, BDA'94*, pp. 305–324. – Clermont-Ferrand, France, 1994.

[DLR91] DELOBEL (C.), LECLUSE (C.) et RICHARD (P.). – *Bases de données : des systèmes relationnels aux systèmes à objets.* – InterEditions, 1991.

[DMC92] DONY (C.), MALENFANT (J.) et COINTE (P.). – Prototype-Based Languages: From a New Taxonomy to Constructive Proposals and their Validation. *Dans : Object-Oriented Programming Systems, Languages and applications, OOPSLA'92*, pp. 201–217. – 1992.

[DNR93] DJERABA (C.), NGUYEN (G.T.) et RIEU (D.). – Objets composites et liens de dépendance dans un système à base de connaissance. *Dans : Congrès Inforsid'93*, pp. 353–372. – Lille, France, 1993.

[DSA91] DSA (Danish Standards Association) (édité par). – *SGML - ODA, Présentation des concepts et comparaison fonctionnelle.* – AFNOR Technique, 1991.

[DV90] DANIEL-VATONNE (M.C.). – Hypertextes : des principes communs et des variations. *Technique et Science Informatiques*, vol. 9, n6, 1990, pp. 475–492.

[DWL92] DEERWESTER (S.C.), WACLENA (K.) et LAMAR (M.). – A Textual Object Management System. *Dans : ACM-SIGIR'92*, pp. 126–139. – Copenhagen, Denmark, 1992.

[EWH85] ELMASRI (R.), WEELDREYER (J.) et HEVNER (A.). – The category concept: An extension to the Entity-Relationship Model. *Data & Knowledge Engineering*, 1985, pp. 75–116.

[FAL85] FALOUTSOS (C.). – Access Methods for Text. *ACM Computing Surveys*, vol. 17, n1, 1985, pp. 49–74.

[FBY92] FRAKES (W.B.) et BAEZA-YATES (R.) (édité par). – *Information Retrieval, Data Structures & Algorithms.* – Prentice Hall, 1992.

[FT83] FISCHER (P.C.) et THOMAS (S.J.). – Operators for Non-First-Normal-Form Relations. *Dans : IEEE COMPSAC*, pp. 464–475. – 1983.

[FTS96] FOUCAUT (O.), THIÉRY (O.) et SMAÏLI (K.). – *Conception des systèmes d'information et programmation événementielle.* – InterEditions, 1996.

[GD92] GROSSMAN (D.A.) et DRISCOLL (J.R.). – Structuring Text within a Relational System. *Dans : Database and Expert Systems Applications, DEXA '92*, pp. 72–77. – 1992.

[GPVG89] GYSSENS (M.), PAREDAENS (J.) et VAN GUCHT (D.). – A Grammar-Based Approach towards Unifying Hierarchical data Models (Extended Abstract). *Dans : ACM-SIGMOD'89*, pp. 263–272. – 1989.

[GR83] GOLDBERG (A.) et ROBSON (D.). – *Smalltalk-80: The Language and its Implementation.* – Addison-Wesley. 1983.

[GT87] GONNET (G.H.) et TOMPA (F.W.). – Mind Your Grammar: a New Approach to Modelling Text. *Dans : Very Large Databases, VLDB'87*, pp. 339–346. – Brighton, UK, 1987.

[GWJ91] GUPTA (A.), WEYMOUTH (T.) et JAIN (R.). – Semantic Queries with Pictures: The VIMSYS Model. *Dans : Very Large Databases, VLDB'91*, pp. 69–79. – Barcelona - Spain, 1991.

[HAL89] HALIN (G.). – Apprentissage pour la recherche interactive d'images : Processus EXPRIM et prototype RIVAGE. – Thèse de l'Université de Nancy I, 1989.

[HCK90] HALIN (G.), CREHANGE (M.) et KEREKES (P.). – Machine-Learning and Vectorial Matching for an Image Retrieval Model : EXPRIM and the RIVAGE System. *Dans : ACM-SIGIR'90*, pp. 99–114. – 1990.

[HEU89] HEUER (A.). – A Data Model for Complex Objects Based on a Semantic Database Model and Nested Relations. *Dans : Nested relations and complex objects in databases.* pp. 297–312. – Springer-Verlag, Lecture Notes in Computer Science, No 361, 1989.

[HHLC92] HOU (T.), HSU (A.), LIU (P.) et CHIU (M.). – Content-Based Indexing Technique Using Relative Geometry Features. *Dans : Image Storage and Retrieval Systems, IS&T / SPIE Symposium on Electronic Imaging - Science and technology, SPIE'92*, pp. 59–68. – San Jose, California, 1992.

[HK87] HULL (R.) et KING (R.). – Semantic Database Modelling: Survey, Applications and Research Issues. *ACM Computing Surveys*, vol. 19, n3, 1987, pp. 201–260.

[HK92] HIRATA (K.) et KATO (T.). – Query by Visual Example - Content Based Image Retrieval. *Dans : Extending Database Technology, EDBT'92.* pp. 56–71. – Springer-Verlag, Lecture Notes in Computer Science, No 580, 1992.

[HLMM92] HIBLER (J.N.D.), LEUNG (C.H.C.), MANNOCK (K.L.) et MWARA (N.). – A System for Content-Based Storage and Retrieval in an Image Database. *Dans : Image Storage and Retrieval Systems, IS&T / SPIE Symposium on Electronic Imaging - Science and technology, SPIE'92*, pp. 80–92. – San Jose, California, 1992.

[HM81] HAMMER (M.M.) et McLEOD (D.J.). – Database Description with SDM : A Semantic Database Model. *ACM Transactions on Database Systems*, vol. 6, n3, 1981, pp. 351–386.

[HM92] HALIN (G.) et MOUADDIB (N.). – An Object Oriented Approach to Design a Content-Based Image Retrieval Model. *Dans : Image Storage and Retrieval Systems, IS&T / SPIE Symposium on Electronic Imaging - Science and technology, SPIE'92*, pp. 100–111. – San Jose, California, 1992.

[HUL89] HULL (R.). – Four Views of Complex Objects: A Sophisticate's Introduction. *Dans : Nested relations and complex objects in databases.* pp. 87–116. – Springer-Verlag, Lecture Notes in Computer Science, No 361, 1989.

[HY84] HULL (R.) et YAP (C.K.). – The Format Model : A Theory of Database Organization. *Journal of the ACM*, vol. 31, n3, 1984, pp. 518–537.

[ISO86] ISO, International organization for standardization. – *Information Processing - Text and Office Systems - Standard Generalized Markup Language (SGML)*, 1986, iso 8879 édition.

[ISO87] ISO, International organization for standardization. – *Information Processing - Text and Office Systems - Office Document Architecture (ODA)*, 1987, iso 8613 édition.

[ISO92] ISO, International organization for standardization. – *Information technology - Hypermedia/Time-based structuring language (HyTime) : international standard*, 1992, iso/iec 10744 édition.

[ISS95] ISSELE (N.). – Environnement de conception et de recherche d'information dédié au modèle EMIR. – Thèse CNAM, 1995.

[JS82] JAESCHKE (G.) et SCHEK (H.J.). – Remarks on the Algebra of Non First Normal Form Relations. *Dans : ACM-SIGMOD'82*, pp. 124–138. – 1982.

[KAT92] KATO (T.). – Database Architecture for Content-Based Image Retrieval. *Dans : Image Storage and Retrieval Systems, IS&T / SPIE Symposium on Electronic Imaging - Science and technology, SPIE'92*, pp. 112–123. – San Jose, California, 1992.

[KBC+87] KIM (W.), BANERJEE (J.), CHOU (H.T.), GARZA (J.F.) et WOELK (D.). – Composite Object Support in an Object-Oriented Database System. *Dans : Object-Oriented Programming Systems, Languages and applications, OOPSLA'87*, pp. 118–125. – 1987.

[KBG89] KIM (W.), BERTINO (E.) et GARZA (J.F.). – Composite Objects Revisited. *Dans : ACM-SIGMOD'89*, pp. 337–347. – 1989.

[KC86] KHOSHOFIAN (S.N.) et COPELAND (G.P.). – Object Identity. *Dans : Object-Oriented Programming Systems, Languages and applications, OOPSLA'86*, pp. 406–416. – 1986.

[KEN79] KENT (W.). – Limitations of Record-Based Information Models. *ACM Transactions on Database Systems*, vol. 4, n1, 1979, pp. 107–131.

[KHO93] KHOSHOFIAN (S.). – *Object-Oriented Databases.* – Wiley Professional Computing, 1993.

[KIM89] KIM (W.). – A Model of Queries for Object-Oriented Databases. *Dans : Very Large Databases, VLDB'89*, pp. 423–432. – 1989.

[KIM90] KIM (W.). – Object-Oriented Databases: definitions and research directions. *IEEE Transactions on Knowledge and Data Engineering*, vol. 2, n3, 1990, pp. 327–341.

[KKH94] KIYOKI (Y.), KITAGAWA (T.) et HAYAMA (T.). – A Metadatabase System for Semantic Image Search by a Mathematical Model of Meaning. *SIGMOD Record*, vol. 23, n4, 1994, pp. 34–41.

[KKL91] KEIM (D.A.), KIM (K.C.) et LUM (V.). – A Friendly and Intelligent Approach to Data Retrieval in a Multimedia DBMS. *Dans : Database and Expert Systems Applications, DEXA'91*, pp. 102–111. – Berlin, Germany, 1991.

[KLS92] KIM (W.), LEE (Y.J.) et SEO (J.). – A Framework for Supporting Triggers in Object-Oriented Database Systems. *International Journal of Intelligent & Cooperative Information Systems*, vol. 1, n1, 1992, pp. 127–143.

[KLU82] KLUG (A.). – Equivalence of Relational Algebra and Relational Calculus Query Languages Having Aggregate Functions. *Journal of the ACM*, vol. 29, n3, 1982, pp. 699–717.

[KV84] KUPER (G.M.) et VARDI (M.Y.). – The Logical data Model. *Dans : ACM Transactions on Database Systems*, pp. 379–413. – 1984.

[LAH95] LAHLOU (Y.). – Modeling Complex Objects in Content-Based Image Retrieval. *Dans : Storage and Retrieval for Image and Video Databases III, IS&T / SPIE Symposium on Electronic Imaging - Science and technology, SPIE'95*, pp. 104–115. – San Jose, California, 1995.

[LAH96] LAHLOU (Y.). – Using an Object-Oriented Data Model as a Meta-Model for Information Retrieval. *Dans : IEEE Conference on Metadata.* – Silver Spring, Maryland, 1996. http://www.nml.org/resources/misc/metadata/proceedings/meta_home.html.

[LEO89] LEONARD (D.). – Conception et représentation des objets à comportement complexe. *Dans : Congrès Inforsid'89*, pp. 285–298. – 1989.

[LIE86] LIEBERMAN (H.). – Using Prototypical Objects to Implement Shared Behavior in Object Oriented Systems. *Dans : Object-Oriented Programming Systems, Languages and applications, OOPSLA'86*, pp. 214–223. – 1986.

[LM96] LAHLOU (Y.) et MOUADDIB (N.). – Relaxing the Instantiation Link: Towards a Content-Based Data Model for Information Retrieval. *Dans : Conference on Advanced Information Systems Engineering, CAiSE'96.* pp. 540–561. – Springer-Verlag, Lecture Notes in Computer Science, No 1080, 1996.

[LNM94] LAHLOU (Y.), NAJA (H.) et MOUADDIB (N.). – *Les textes et l'informatique*, chap. Modélisation orientée objet de textes. Présentation d'un modèle basé sur le contenu : EMIR, pp. 115–144. – Didier Erudition, 1994.

[LOE94] LOEFFEN (A.). – Text Databases: A Survey of Text Models and Systems. *SIGMOD Record*, vol. 23, n1, 1994, pp. 97–106.

[LR89] LECLUSE (C.) et RICHARD (P.). – The O_2 Database Programming Language. *Dans : Very Large Databases, VLDB'89*, pp. 411–422. – 1989.

[LTP86] LALONDE (W.R.), THOMAS (D.A.) et PUGH (J.R.). – An Exemplar Based Smalltalk. *Dans : Object-Oriented Programming Systems, Languages and applications, OOPSLA'86*, pp. 322–330. – 1986.

[LUT89] LUTZ (E.). – Knowledge Based Classification of Office Documents. *Bigre + Globule*, vol. 63-64, 1989, pp. 353–362.

[MAC90] MACLEOD (I.A.). – Storage and retrieval of Structured Documents. *Information Processing & Management*, vol. 26, n2, 1990, pp. 197–208.

[MAC91a] MACLEOD (I.A.). – A Query Language for Retrieving Information from Hierarchic Text Structures. *The Computer Journal*, vol. 34, n3, 1991, pp. 254–264.

[MAC91b] MACLEOD (I.A.). – Text Retrieval and the Relational Model. *Journal of the American Society for Information Science*, vol. 42, n3, 1991, pp. 155–165.

[MAG94] MAGNAN (M.). – Réutilisation de composants : Les exceptions dans les objets composites. – Thèse de l'Université de Montpellier II, 1994.

[MAR91] MARSHALL (R.). – Manipulating Full-Text Scientific Databases : A Logic-based Semantico-pragmatic Approach. *The Computer Journal*, vol. 34, n3, 1991, pp. 245–253.

[MAS87] MASUNAGA (Y.). – Multimedia Databases: A Formal Framework. *Dans: IEEE Office Automation Symposium*, pp. 36–45. – 1987.

[MBHL91] MACLEOD (I.A.), BARNARD (D.T.), HAMILTON (D.) et LEVISON (M.). – SGML Documents and Non-Linear Text Retrieval. *Dans: Recherche d'Information Assistée par Ordinateur, RIAO'91*, pp. 226–244. – Barcelona, Spain, 1991.

[MBW80] MYLOPOULOS (J.), BERNSTEIN (P.A.) et WONG (H.K.T.). – A Language Facility for Designing Database-Intensive Applications. *ACM Transactions on Database Systems*, vol. 5, n2, 1980, pp. 185–207.

[MEC95] MECHKOUR (M.). – EMIR^2. Un modèle étendu de représentation et de correspondance d'images pour la recherche d'informations. Application à un corpus d'images historiques. – Thèse de l'Université Joseph Fourier - Grenoble I, 1995.

[MEG95] MEGHINI (C.). – An Image Retrieval Model Based on Classical Logic. *Dans: ACM-SIGIR'95*, pp. 300–308. – Seattle - Washigton, 1995.

[MLH94] MOUADDIB (N.), LAHLOU (Y.) et HALIN (G.). – EMIR: A Content-Based Data Model. *Dans: Maghrebian Conference on Software Engineering and Artificial Intelligence, MCSEAI'94*, pp. 503–512. – Rabat, Morocco, 1994.

[MNC+89] MASINI (G.), NAPOLI (A.), COLNET (D.), LEONARD (D.) et TOMBRE (K.). – *Les langages à objets.* – InterEditions, 1989.

[MO94] MOULINE (S.) et OQUENDO (F.). – Algèbre des objets classes : une algèbre pour les bases de données orientées objet. *Dans: Langages et Modèles à Objets, LMO'94.* – Grenoble, France, 1994.

[MRT91] MEGHINI (C.), RABITTI (F.) et THANOS (C.). – Conceptual Modeling of Multimedia Documents. *IEEE Computer*, 1991, pp. 23–29.

[MWW89] MEYER (J.J.), WEIGAND (H.) et WIERINGA (R.). – A Specification Language for Static, Dynamic and Deontic Integrity Constraints. *Dans: The 2nd Symposium on Mathematical Fundamentals of database Systems.* pp. 347–366. – Springer-Verlag, Lecture Notes in Computer Science, No 364, 1989.

[NAP92] NAPOLI (A.). – Représentations à objets et raisonnement par classification en intelligence artificielle. – Thèse d'Etat, Univ. Nancy I, 1992.

[NLC96] NAJA (H.), LAHLOU (Y.) et COMTE (B.). – Une approche basée sur les points de vue pour la modélisation et l'interrogation de données dentaires. *Dans: Congrès Inforsid'96*, pp. 119–137. – Bordeaux - France, 1996.

[NM95] NAJA (H.) et MOUADDIB (N.). – The Multiple Representation in an Architectural Application. *Dans : Database and Expert Systems Applications, DEXA'95.* pp. 237–246. – Springer-Verlag, Lecture Notes in Computer Science, No 978, 1995.

[NML93] NAJA (H.), MOUADDIB (N.) et LAHLOU (Y.). – *Extension du modèle EMIR pour la gestion de la dynamique des objets.* – Rapport technique n 93-R-193, CRIN - CNRS, Nancy, France, 1993.

[NQZ91] NASSIF (R.), QIU (Y.) et ZHU (J.). – Extending the Object-Oriented Paradigm to Support Relationships and Constraints. *Dans : Object-Oriented Databases: Analysis, Design & Construction, IFIP DS-4*, pp. 305–329. – 1991.

[NR91] NGUYEN (G.T.) et RIEU (D.). – Database Issues in Object-Oriented Design. *Dans : Technology of Object-Oriented Languages and Systems, TOOLS'91.* – Paris - France, 1991.

[OD91] O'DOCHERTY (M.H.) et DASKALAKIS (C.N.). – Multimedia Information Systems - The Management and Semantic Retrieval of all Electronic Data Types. *The Computer Journal*, vol. 34, n3, 1991, pp. 225–238.

[ODE94] ODELL (J.J.). – Six Different Kinds of Composition. *Journal of Object-Oriented Programming*, vol. 5, n8, 1994, pp. 10–15.

[OOM87] OZSOYOGLU (G.), OZSOYOGLU (Z.M.) et MATOS (V.). – Extending Relational Algebra and Relational Calculus with Set-Valued Attributes and Aggregate Functions. *ACM Transactions on Database Systems*, vol. 12, n4, 1987, pp. 566–592.

[ORI92] ORIA (V.). – Construction d'un SGBD d'images. *Dans : Conférence Africaine sur la Recherche en Informatique, CARI'92*, pp. 313–324. – Yaounde, Cameroun, 1992.

[PAL89] PALISSER (C.). – Charly, un gestionnaire de versions pour la CAO en architecture. – Thèse de l'Université d'Aix-Marseille III, 1989.

[PM88] PECKHAM (J.) et MARYANSKI (F.). – Semantic Data Models. *ACM Computing Surveys*, vol. 20, n3, 1988, pp. 153–189.

[PMB+89] PECKHAM (J.), MARYANSKI (F.), BESHERS (G.), CHAPMAN (H.) et DEMURJIAN (S.A.). – Constraint Based Analysis of Database Update Propagation. *Dans : International Conference on Information Systems, ICIS'89*, pp. 9–18. – Boston, Massachussets, 1989.

[PON93] PONCELET (P.). – Contribution à la conception de bases de données avancées : modélisation, évolution et dérivation. – Thèse de l'Université de Nice, Sophia-Antipolis, 1993.

[QV89] QUINT (V.) et VATTON (I.). – Modularity in Structured Documents. *Bigre + Globule*, vol. 63-64, 1989, pp. 170–177.

[RKS88] ROTH (M.A.), KORTH (H.F.) et SILBERSCHATZ (A.). – Extended Algebra and Calculus for Nested Relational databases. *ACM Transactions on Database Systems*, vol. 13, n4, 1988, pp. 389–417.

[RS92a] RABITTI (F.) et SAVINO (P.). – Automatic Image Indexation to Support Content-Based Retrieval. *Information Processing & Management*, vol. 28, n 5, 1992, pp. 547–565.

[RS92b] RABITTI (F.) et SAVINO (P.). – Querying Semantic Image Databases. *Dans : Image Storage and Retrieval Systems, IS&T / SPIE Symposium on Electronic Imaging - Science and technology, SPIE'92*, pp. 69–78. – San Jose, California, 1992.

[RT88] RAYMOND (D.R.) et TOMPA (F.W.). – Hypertext and the Oxford English Dictionary. *Communications of the ACM*, vol. 31, n7, 1988, pp. 871–879.

[RUM87] RUMBAUGH (J.). – Relations as Semantic Constructs in an Object-Oriented Language. *Dans : Object-Oriented Programming Systems, Languages and applications, OOPSLA'87*, pp. 466–481. – 1987.

[SA91] SU (S.Y.W.) et ALASHQUR (A.M.). – A Pattern-Based Constraint Specification Language for Object-Oriented Databases. *Dans : COMPCON Spring 91*, pp. 522–531. – San Francisco, California, 1991.

[SCH90] SCHWARZ (C.). – Content-Based Text Handling. *Information Processing & Management*, vol. 26, n2, 1990, pp. 219–226.

[SCI89] SCIORE (E.). – Object Specialization. *ACM Transactions on (Office) Information Systems*, vol. 7, n2, 1989, pp. 103–122.

[SGL93] SU (S.Y.W.), GUO (M.) et LAM (H.). – Association Algebra: A Mathematical Foundation for Object-Oriented Databases. *IEEE Transactions on Knowledge and Data Engineering*, vol. 5, n5, 1993, pp. 775–798.

[SHI81] SHIPMAN (D.). – The Functional Data Model and the Data Language DAPLEX. *ACM Transactions on Database Systems*, vol. 6, n 1, 1981, pp. 140–173.

[SLU89] STEIN (L.A.), LIEBERMAN (H.) et UNGAR (D.). – A Shared View of Sharing: *The Treaty of Orlando. Dans : Object-Oriented Concepts, Databases, and Applications*, éd. par KIM (W.) et LOCHOVSKY (F.H.), pp. 31–48. – Addison-Wesley, 1989.

[SM83] SALTON (G.) et McGILL (M.J.). – *Introduction to Modern Information Retrieval.* – McGraw-Hill, 1983.

[SM88] SHLAER (S.) et MELLOR (S.J.). – *Object-Oriented Systems Analysis. Modeling the World in Data.* – Yourdon Press, 1988.

[SMA94] SMAÏL (M.). – Raisonnement à base de cas pour une recherche évolutive d'information ; Prototype Cabri-n. Vers la définition d'un cadre d'acquisition de connaissances. – Thèse de l'Université de Nancy I, 1994.

[SS77] SMITH (J.M.) et SMITH (D.C.P.). – Database abstractions: Aggregation and Generalization. *ACM Transactions on Database Systems*, vol. 2, n 2, 1977, pp. 105–133.

[SS86] SCHEK (H.J.) et SCHOLL (M.H.). – The Relational Model with Relation-Valued Attributes. *Information Systems*, vol. 11, n2, 1986, pp. 137–147.

[SS90] SCHEK (H.J.) et SCHOLL (M.H.). – Evolution of Data Models. *Dans: Database Systems of the 90's.* pp. 135–153. – Springer-Verlag, Lecture Notes in Computer Science, No 466, 1990.

[STR91] STRAUBE (D.D.). – Queries and Query Processing in Object-Oriented Database Systems. – Thèse de l'Universitè d'Alberta, Canada, 1991.

[SU83] SU (S.Y.W.). – SAM*: A Semantic Associative Model for Corporate and Scientific Statistical Databases. *Information Systems*, vol. 29, 1983, pp. 151–199.

[SUB95] SUBTIL (P.). – Amélioration de la flexibilité dans les bases d'objets flous. Expérimentation dans le cadre du logiciel FIRMS. – Thèse de l'Université Henri Poincaré - Nancy I, 1995.

[SZ90] SHAW (G.M.) et ZDONIK (S.B.). – A Query Algebra for Object-Oriented Databases. *Dans: International Conference on Data Engineering, ICDE'90*, pp. 154–162. – 1990.

[TOU91] TOUIR (A.). – The Design of an Efficient Data Structure for Manipulating Data in an Image Database System. *Dans: Database and Expert Systems Applications, DEXA'91*, pp. 191–196. – Berlin, Germany, 1991.

[TYF86] TEOREY (T.J.), YANG (D.) et FRY (J.P.). – A Logical Design Methodology for Relational Databases using the Extended Entity-Relationship Model. *ACM Computing Surveys*, vol. 18, n2, 1986, pp. 197–222.

[UCCH91] UNGAR (D.), CHAMBERS (C.), CHANG (B.W.) et HÖLZLE (U.). – Organizing Programs without Classes. *LISP and Symbolic Computation*, vol. 4, n3, 1991, pp. 223–242.

[UD88] URBAN (S.D.) et DELCAMBRE (L.M.L.). – Constraint Analysis: A Tool for Explaining the Semantics of Complex Objects. *Dans: The 2nd International Workshop on Advances in Object-Oriented Database Systems.* pp. 156–161. – Springer-Verlag, Lecture Notes in Computer Science, No 334, 1988.

[UD90] URBAN (S.D.) et DELCAMBRE (L.M.L.). – Constraint Analysis: A Design Process for Specifying Operations on Objects. *IEEE Transactions on Knowledge and Data Engineering*, vol. 2, n4, 1990, pp. 391–400.

[US87] UNGAR (D.) et SMITH (R.B.). – SELF: The Power of Simplicity. *Dans : Object-Oriented Programming Systems, Languages and applications, OOPSLA '87*, pp. 227–242. – 1987.

[US91] UNGAR (D.) et SMITH (R.B.). – SELF: The Power of Simplicity. *LISP and Symbolic Computation*, vol. 4, n3, 1991, pp. 187–205.

[WBJ90] WIRFS-BROCK (R.J.) et JOHNSON (R.E.). – Surveying Current Research in Object-Oriented Design. *Communications of the ACM*, vol. 33, n9, 1990, pp. 104–124.

[WCH87] WINSTON (M.E.), CHAFFIN (R.) et HERRMANN (D.). – A Taxonomy of Part-Whole Relations. *Cognitive Science*, vol. 11, 1987, pp. 417–444.

[WKL86] WOELK (D.), KIM (W.) et LUTHER (W.). – An Object-Oriented Approach to Multimedia Databases. *Dans : ACM-SIGMOD'86*, pp. 311–325. – 1986.

[WLK87] WOELK (D.), LUTHER (W.) et KIM (W.). – Multimedia Applications and Database Requirements. *Dans : IEEE Office Automation Symposium*, pp. 180–189. – 1987.

[WLS95] WHALEN (T.), LEE (E.S.) et SAFAYENI (F.). – The Retrieval of images from Image Databases: Trademarks. *Behaviour & Information Technology*, vol. 14, n1, 1995, pp. 3–13.

[WNM+95] WU (J.K.), NARASIMHALU (A.D.), MEHTRE (B.M.), LAM (C.P.) et GAO (Y.J.). – CORE: a Content-Based Retrieval Engine for Multimedia Information Systems. *Mutimedia Systems*, 1995, pp. 25–41.

[WOR91] WORBOYS (M.F.). – Database Specification using Transaction Sets. *Dans : International Workshop on Specifications of Database Systems*, pp. 300–311. – Glasgow, UK, 1991.

Annexe A

Formalisme graphique du modèle EMIR

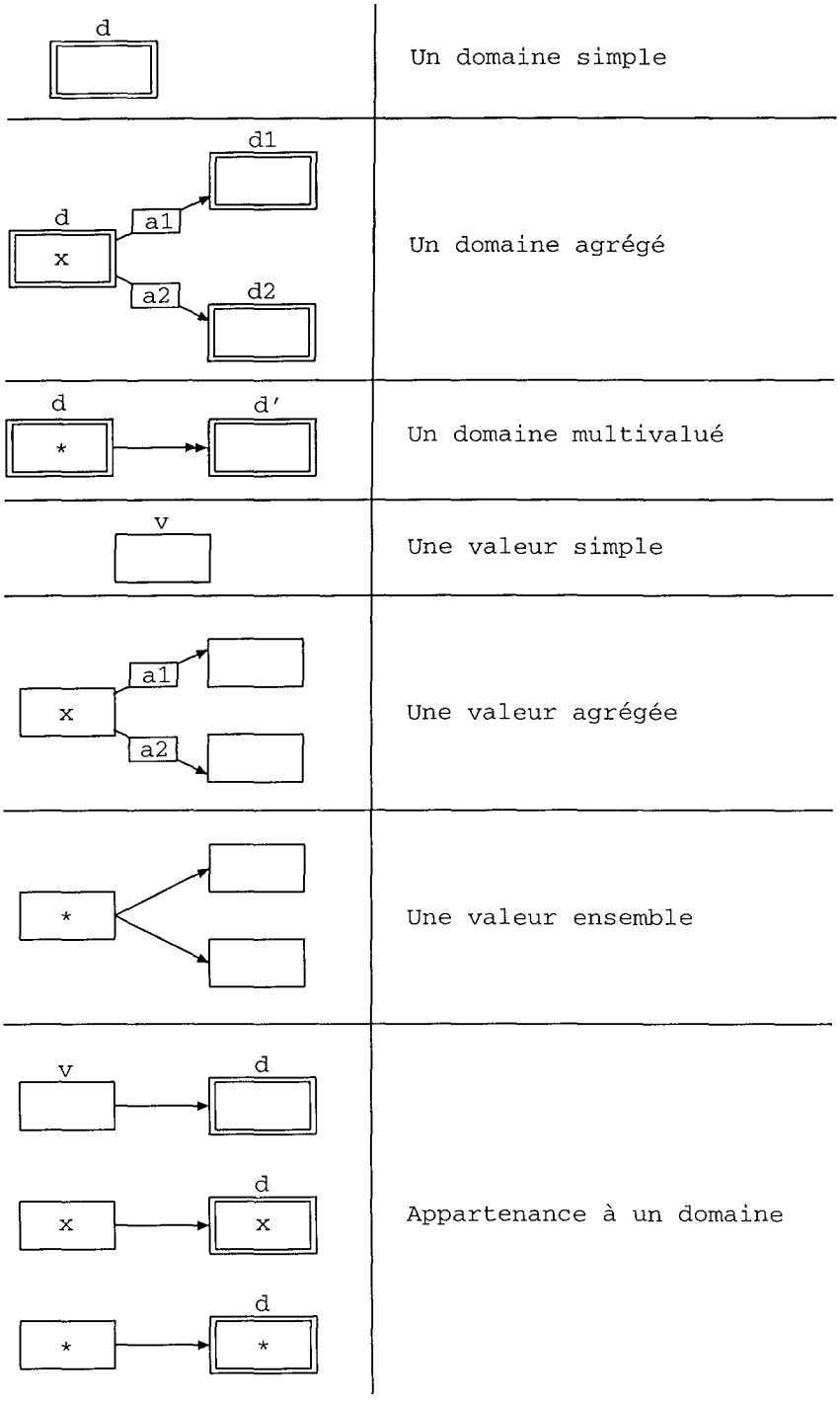


FIG. A.1 – *Formalisme graphique pour les domaines et les valeurs dans EMIR*

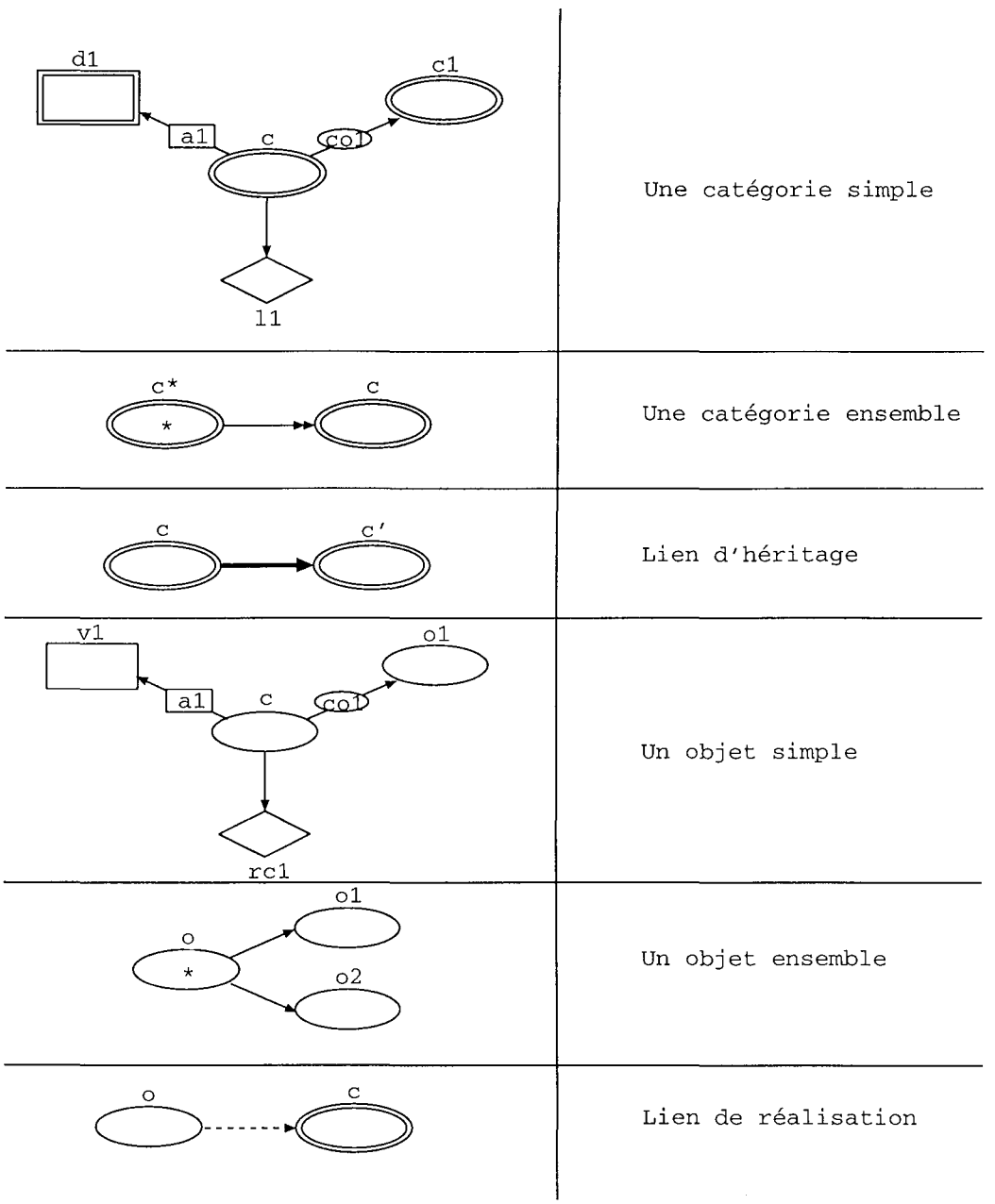


FIG. A.2 – *Formalisme graphique pour les catégories et les objets dans EMIR*

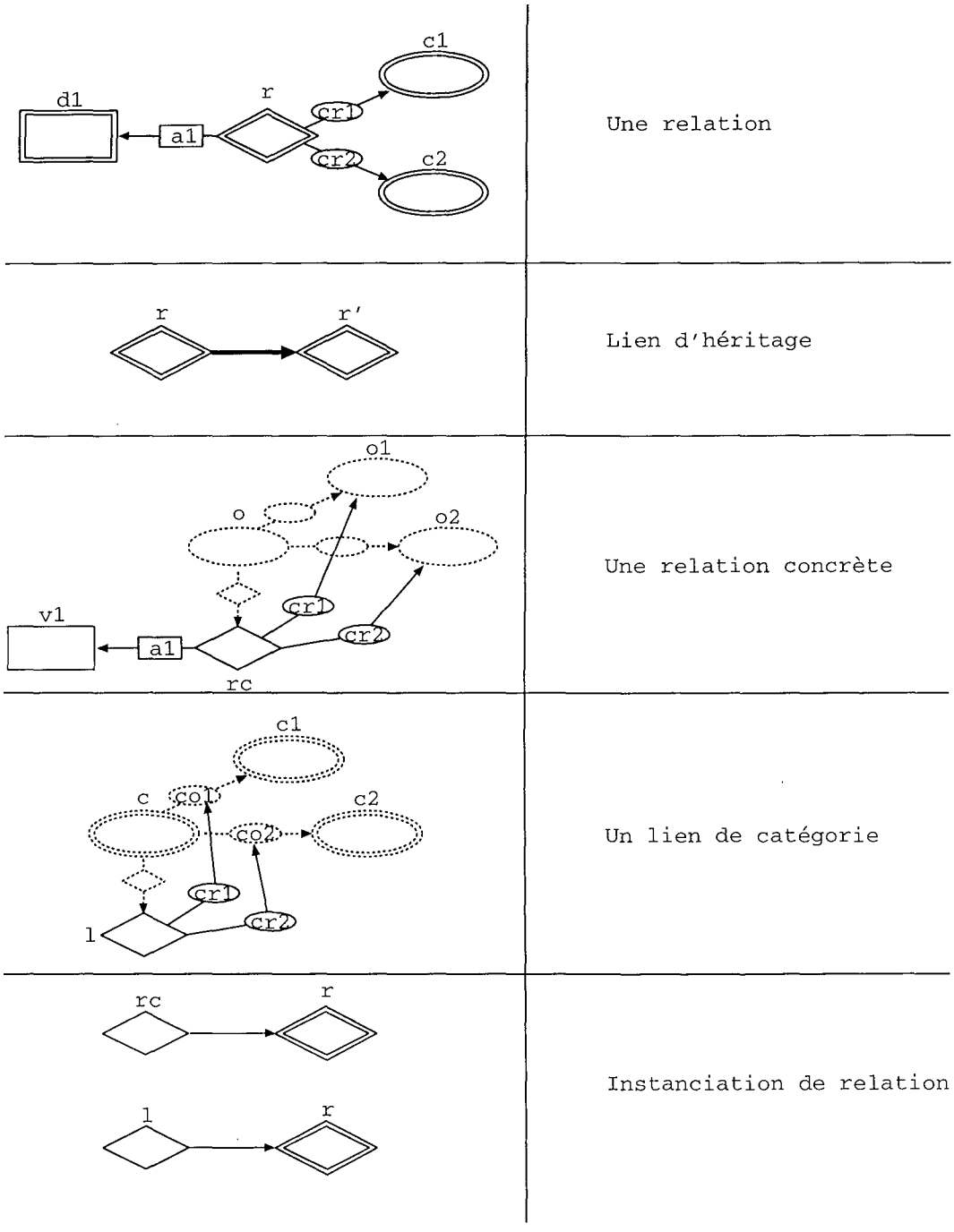


FIG. A.3 – *Formalisme graphique pour les relations, les relations concrètes et les liens de catégorie dans EMIR*

Annexe B

Opérations de maintien de l'intégrité

B.1 Opérations de création

OCDS : Opération de Création d'un Domaine Simple.

Paramètres :
d : string
$tv \in \{B, I, R, S, M\}$
Préconditions :
$d \notin \mathbf{D}.nom$ ⟵ IND
Actions :
$\mathbf{D} \cup = \{d\}$
$[\![d]\!] := tv$

OCDD : Opération de Création d'un Domaine simple Discret.

Paramètres :
d : string
$tv \in \{I, R, S\}$
$v \in V(tv)$
Préconditions :
$d \notin \mathbf{D}.nom$ ⟵ IND
Actions :
$\mathbf{D} \cup = \{d\}$
$[\![d]\!] := tv$
$d.valeurs := \{v\}$ ⟵ IDD
$defaut(d) := v$

OCDC : Opération de Création d'un Domaine simple Continu.

Paramètres :
d : string
$tv \in \{I, R, S\}$
$v_1, v_2 \in V(tv)$
$borneG, borneD$: Booléens
Préconditions :
$d \notin \mathbf{D}.nom$ ⟵ IND
$v_1 \leq v_2$ ⟵ IDC
Actions :
$\mathbf{D} \cup = \{d\}$
$[\![d]\!] := tv$
$d.bornes := [(])v_1, v_2[(])$ (selon les valeurs des paramètres)
$defaut(d) :=$ valeur moyenne

OCDM : Opération de Création d'un Domaine Multivalué.

Paramètres :
d : string

$d' \in \mathcal{D}$
Préconditions :
$d \notin \mathbf{D}.nom$ ⟵ IND
Actions :
$\mathbf{D} \cup = \{d\}$
$d.sous\_domaine := d'$
$DuseD \cup = \{(d, d', 1)\}$
pour $(d', d'', *) \in DuseD$
    $DuseD \cup = \{(d, d'', 1)\}$

OCDA : Opération de Création d'un Domaine Agrégé.
Paramètres :
d : string
$a_1, ..., a_n$ strings différents, $n \geq 2$
$d_1, ..., d_n \in \mathcal{D}$, $n \geq 2$
Préconditions :
$d \notin \mathbf{D}.nom$ ⟵ IND
Actions :
$\mathbf{D} \cup = \{d\}$
$d.description := \{a_i : d_i, 1 \leq i \leq n\}$
pour $i \in [1, n]$
    si $(d, d_i, m) \in DuseD$
        $DuseD - = \{(d, d_i, m)\} \cup = \{(d, d_i, m+1)\}$
    sinon
        $DuseD \cup = \{(d, d_i, 1)\}$
    pour $(d_i, d_i', *) \in DuseD$
        si $(d, d_i', m) \in DuseD$
            $DuseD - = \{(d, d_i', m)\} \cup = \{(d, d_i', m+1)\}$
        sinon
            $DuseD \cup = \{(d, d_i', 1)\}$

OCR : Opération de Création d'une Relation.
Paramètres :
r : string
$r' \in \mathbf{R}$
Préconditions :
$r \notin \mathbf{R}.nom$ ⟵ INR
Actions :
$\mathbf{R} \cup = \{r\}$
$r.description := r'.description$ ⟵ IHRG
pour $a : d \in r'.description$
    $RuseD \cup = \{(r, d)\}$
$r.composition := r'.composition$ ⟵ IHRG

pour $ro : c \in r'.composition$
$RuseC \cup = \{(r, c)\}$
$RgenR \cup = \{(r, r)\}$ (initialisation des fonctions de renommage σ_c et σ_a)
pour $(r'', r') \in RgenR$
$RgenR \cup = \{(r'', r)\}$ (initialisation des fonctions de renommage σ_c et σ_a)

OCC : Opération de Création d'une Catégorie.
Paramètres :
c : string
$c' \in \mathbf{C}$
Préconditions :
$c \notin \mathbf{C}.nom$ ⟵ INC
Actions :
$\mathbf{C} \cup = \{c\}$
$c.description := c'.description$ ⟵ IHCG
pour $a : d \in c'.description$
$CuseD \cup = \{(c, d)\}$
$c.composition := c'.composition$ ⟵ IHCG
pour $ro : c''(t) \in c'.composition$
$CuseC \cup = \{(c, c'')\}$
$c.topologie := c'.topologie$ ⟵ IHCG
pour $l : r \in c'.topologie$
$CuseR \cup = \{(c, r)\}$
$CgenC \cup = \{(c, c)\}$ (initialisation des fonctions de renommage σ_a, σ_c et σ_l)
pour $(c'', c') \in CgenC$
$CgenC \cup = \{(c'', c)\}$ (initialisation des fonctions de renommage σ_a, σ_c et σ_l)

OCO : Opération de Création d'un Objet.
Paramètres :
o : string
$c \in \mathbf{C}$ ⟵ IRCE
Préconditions :
$o \notin \mathbf{O}.nom$ ⟵ INO
Actions :
$\mathbf{O} \cup = \{o\}$
$o.description := \{(a_i : defaut(d_i)), (a_i : d_i) \in c.description\}$ ⟵ IRCG
$o.composition := \{\}$
pour $ro_j : c_j(t_j) \in c.composition$
choisir o' parmi $\{o' \in \mathbf{O}, (o', c_j) \in OrealC$ ⟵ IRCG
si choix fait
$o.composition \cup = \{ro_j : o.ro_j(t_jO)\}$
(le nouveau type de lien t_jO est obtenu en ajoutant "O" (obligatoire) au type de lien t_j)

$OuseO \cup = \{(o, o.ro_j)\}$

sinon

$OCO\,(o.ro_j\,,\, c_j)$ ⟵ IRCG

$o.composition \cup = \{ro_j : o.ro_j(t_jO)\}$

$OuseO \cup = \{(o, o.ro_j)\}$

$o.topologie := \{\}$

pour $l_k : r_k \in c.topologie$

$OMOAL\,(o\,,\, l_k\,,\, r_k\,,\, o_1\,,\, \ldots\,,\, o_{n_k})$ ⟵ IRCG

(les objets o_u provenant de la composition de o, conformément à la définition du lien l_k dans c)

$OuseR \cup = \{(o, r_k)\}$

pour $(c', c) \in CgenC$

$OrealC \cup = \{(o, c')\}$

B.2 Opérations de renommage

ORD : Opération de Renommage d'un Domaine (y compris simple, discret et continu, multivalué et agrégé).
Paramètres :
$d \in \mathbf{D}$
d' : string
Préconditions :
$d' \notin \mathbf{D}.nom$ ⟵ IND
Actions :

ORDARA : Opération de Renommage dans un Domaine agrégé : Renommage d'un Attribut.
Paramètres :
$d \in \mathbf{D}$
a : string
a' : string
Préconditions :
$a : d' \in d.description$
$a' : * \notin d.description$ ⟵ INAD
Actions :
$d.description -= \{a : d'\} \cup= \{a' : d'\}$

ORR : Opération de Renommage d'une Relation.
Paramètres :
$r \in \mathbf{R}$
r' : string
Préconditions :
$r' \notin \mathbf{R}.nom$ ⟵ INR
Actions :

ORRRA : Opération de Renommage dans une Relation : Renommage d'un Attribut.
Paramètres :
$r \in \mathbf{R}$
a : string
a' : string
Préconditions :
$a : d \in r.description$
$a' : * \notin r.description$ ⟵ INAR
Actions :
$r.description -= \{a : d\} \cup= \{a' : d\}$
pour $(r, r') \in RgenR$

$ORRRA(r', a, a')$

ORRRC : Opération de Renommage dans une Relation : Renommage d'un Composant.

Paramètres :
$r \in \mathbf{R}$
ro : string
ro' : string
Préconditions :
$ro : c \in r.composition$
$ro' : * \notin r.composition$ ⟵ INCR
Actions :
$r.composition -= \{ro : c\} \cup= \{ro' : c\}$
$pour\ (r, r') \in RgenR$
$ORRRC\ (r', ro, ro')$

ORC : Opération de Renommage d'une Catégorie.

Paramètres :
$c \in \mathbf{C}$
c' : string
Préconditions :
$c' \notin \mathbf{C}.nom$ ⟵ INC
Actions :

ORCRA : Opération de Renommage dans une Catégorie : Renommage d'un Attribut.

Paramètres :
$c \in \mathbf{C}$
a : string
a' : string
Préconditions :
$a : d \in c.description$
$a' : * \notin c.description$ ⟵ INAC
Actions :
$c.description -= \{a : d\} \cup= \{a' : d\}$
$pour\ (c, c') \in CgenC$
$ORCRA\ (c', a, a')$

ORCRC : Opération de Renommage dans une Catégorie : Renommage d'un Composant.

Paramètres :
$c \in \mathbf{C}$
a : string
ro' : string
Préconditions :

$ro : c'(t) \in c.composition$
$ro' : * \notin c.composition$ ⟵ INCC
Actions :
$c.composition -= \{ro : c'(t)\} \cup= \{ro' : c'(t)\}$
pour $(c, c') \in CgenC$
$\quad ORCRC\,(c', ro, ro')$

ORCRL : Opération de Renommage dans une Catégorie : Renommage d'un Lien.
Paramètres :
$c \in \mathbf{C}$
l : string
l' : string
Préconditions :
$l : r \in c.topologie$
$l' : * \notin c.topologie$ ⟵ INLC
Actions :
$c.topologie -= \{l : r\} \cup= \{l' : r\}$
pour $(c, c') \in CgenC$
$\quad ORCRL\,(c', l, l')$

ORO : Opération de Renommage d'un Objet.
Paramètres :
$o \in \mathbf{O}$
o' : string
Préconditions :
$o' \notin \mathbf{O}.nom$ ⟵ INO
Actions :

ORORC : Opération de Renommage dans un Objet : Renommage d'un Composant.
Paramètres :
$o \in \mathbf{O}$
ro : string
ro' : string
Préconditions :
$ro : o' \in o.composition$
$ro' : * \notin o.composition$ ⟵ INCO
$\forall c \in \mathbf{C}\, tq\,(o, c) \in OrealC, ro : c' \notin c.composition$ ⟵ IRCG
Actions :
$o.composition -= \{ro : o'\} \cup= \{ro' : o'\}$

ORORRC : Opération de Renommage dans un Objet : Renommage d'une Relation Concrète.
Paramètres :

$o \in \mathbf{O}$
rc : string
rc' : string
Préconditions :
$rc : r \in o.topologie$
$rc' : * \notin o.topologie$ ⟵ INLO
$\forall c \in \mathbf{C}\, tq\,(o, c) \in OrealC, rc : r \notin c.topologie$ ⟵ IRCG
Actions :
$o.topologie -= \{rc : r\} \cup= \{rc' : r\}$

B.3 Opérations de suppression

OSD : Opération de Suppression d'un Domaine (y compris simple, discret et continu, multivalué et agrégé).

Paramètres :

$d \in \mathbf{D}$

Préconditions :

$\{(*, d, *) \in DuseD\} = \{\}$

$\{(*, d) \in RuseD\} = \{\}$

$\{(*, d) \in CuseD\} = \{\}$

Actions :

$DuseD -= \{(d, *, *)\}$

$\mathbf{D} -= \{d\}$

OSR : Opération de Suppression d'une Relation.

Paramètres :

$r \in \mathbf{R}$

Préconditions :

pour $(r, r') \in RgenR$

$\{(*, r') \in CuseR\} = \{\}$

$\{(*, r') \in OuseR\} = \{\}$

Actions :

pour $(r, r') \in RgenR$

$RuseD -= \{(r', *)\}$

$RuseC -= \{(r', *)\}$

$\mathbf{R} -= \{r'\}$

OSC : Opération de Suppression d'une Catégorie.

Paramètres :

$c \in \mathbf{C}$

Préconditions :

pour $(c, c') \in CgenC$

$\{(*, c') \in CuseC\} = \{\}$

$\{(*, c') \in RuseC\} = \{\}$

Actions :

pour $(c, c') \in CgenC$

$CuseD -= \{(c', *)\}$

$CuseC -= \{(c', *)\}$

$CuseR -= \{(c', *)\}$

pour $(o, c') \in OrealC$

$OSO\,(o)$

$\mathbf{C} -= \{c'\}$

OSO : Opération de Suppression d'un Objet.

Paramètres :

$o \in \mathbf{O}$

Préconditions :

Actions :

pour $ro : o'(t) \in o.composition$

$OuseO -= \{(o, o')\}$

$si\,(t = DO) \vee (t = Do)$

$OSO\,(o')$ ⟵ ICOD

pour $(o', o) \in OuseO$

pour $ro : o(t) \in o'.composition$

$si\,(t = DO) \vee (t = IO)$

$OSO\,(o')$ ⟵ ICOO

sinon

$OuseO -= \{(o', o)\}$

pour $rc : r \in o.topologie$

$OuseR -= \{(o, r)\}$

$OrealC -= \{(o, *)\}$

$\mathbf{O} -= \{o\}$

B.4 Opérations de modification

OMDDAV : Opération de Modification d'un Domaine Discret : Ajout d'une Valeur.

Paramètres :
$d \in \mathbf{D}$
$v \in V(d)$
Préconditions :
Actions :
$d.valeurs \cup = \{v\}$

OMDDSV : Opération de Modification d'un Domaine simple Discret : Suppression d'une Valeur.

Paramètres :
$d \in \mathbf{D}$
$v \in d.valeurs$
Préconditions :
$card(d.valeurs) \geq 2$ ⟵ IDD
Actions :
si $defaut(d) = v$
    $defaut(d) :=$ une valeur choisie dans $d.valeurs - \{v\}$
pour $(c, d) \in CuseD$
    pour $(o, c) \in OrealC$
        pour $a : d \in c.description$
            si $a : v \in o.description$
                $OMOMA(o, a, defaut(d))$ ⟵ IRCG
pour $(r, d) \in RuseD$
    pour $(o, r) \in OuseR$
        pour $rc : r \in o.topologie$
            pour $a : d \in r.description$
                si $a : v \in rc.description$
                    $OMOMAL(o, rc, a, defaut(d))$ ⟵ IRCG
$d.valeurs - = \{v\}$

OMDC : Opération de Modification d'un Domaine simple Continu : nouvelles bornes.

Paramètres :
$d \in \mathbf{D}$
$v_1, v_2 \in V(d)$
$borneG, borneD$: Booléens
Préconditions :
$v_1 \leq v_2$ ⟵ IDC
Actions :
$d.bornes := [([)v_1, v_2](])$
$defaut(d) :=$ valeur moyenne
pour $(c, d) \in CuseD$

pour $(o, c) \in OrealC$
pour $a : d \in c.description$
si $(a : v \in o.description) \wedge (v \notin V(d))$
$OMOMA(o, a, defaut(d))$ ⟵ IRCG
pour $(r, d) \in RuseD$
pour $(o, r) \in OuseR$
pour $rc : r \in o.topologie$
pour $a : d \in r.description$
si $(a : v \in rc.description) \wedge (v \notin V(d))$
$OMOMAL(o, rc, a, defaut(d))$ ⟵ IRCG

OMDAAA : Opération de Modification d'un Domaine Agrégé : Ajout d'un Attribut.

Paramètres :
$d \in \mathbf{D}$
a : string
$d' \in \mathbf{D}$
Préconditions :
$a : * \notin d.description$ ⟵ INAD
$d' \neq d$ ⟵ IAD
$(d', d, *) \notin DuseD$ ⟵ IAD
Actions :
$d.description \cup = \{a : d'\}$
pour $d_h \in \{d\} \cup \{d'' \in \mathbf{D}, (d'', d, *) \in DuseD\}$
pour $d_b \in \{d'\} \cup \{d'' \in \mathbf{D}, (d', d'', *) \in DuseD\}$
si $(d_h, d_b, n) \in DuseD$
$DuseD - = \{(d_h, d_b, n)\} \cup = \{(d_h, d_b, n+1)\}$
sinon
$DuseD \cup = \{(d_h, d_b, 1)\}$
pour $(c, d) \in CuseD$
pour $(o, c) \in OrealC$
pour $a : d \in c.description$
$o.description \cup = \{a : defaut(d)\}$ ⟵ IRCG
pour $(r, d) \in RuseD$
pour $(o, r) \in OuseR$
pour $rc : r \in o.topologie$
pour $a : d \in r.description$
$rc.description \cup = \{a : defaut(d)\}$ ⟵ IRCG

OMDAMA : Opération de Modification d'un Domaine Agrégé : Modification du domaine d'un Attribut.

Paramètres :
$d \in \mathbf{D}$
a : string

$d' \in \mathbf{D}$
Préconditions :
$a : d'' \in d.description$
$d' \neq d$ ⟵ IAD
$(d', d, *) \notin DuseD$ ⟵ IAD
Actions :
$d.description -= \{a : d''\} \cup= \{a : d'\}$
pour $d_h \in \{d\} \cup \{d''' \in \mathbf{D}, (d''', d, *) \in DuseD\}$
  pour $d_b \in \{d''\} \cup \{d''' \in \mathbf{D}, (d'', d''', *) \in DuseD\}$
    si $(d_h, d_b, n) \in DuseD$ avec $n \geq 2$
      $DuseD -= \{(d_h, d_b, n)\} \cup= \{(d_h, d_b, n-1)\}$
    sinon
      $DuseD -= \{(d_h, d_b, n)\}$
  pour $d_b \in \{d'\} \cup \{d''' \in \mathbf{D}, (d', d''', *) \in DuseD\}$
    si $(d_h, d_b, n) \in DuseD$
      $DuseD -= \{(d_h, d_b, n)\} \cup= \{(d_h, d_b, n+1)\}$
    sinon
      $DuseD \cup= \{(d_h, d_b, 1)\}$
pour $(c, d) \in CuseD$
  pour $(o, c) \in OrealC$
    pour $a : d \in c.description$
      $o.description -= \{a : *\} \cup= \{a : defaut(d)\}$ ⟵ IRCA
pour $(r, d) \in RuseD$
  pour $(o, r) \in OuseR$
    pour $rc : r \in o.topologie$
      pour $a : d \in r.description$
        $rc.description -= \{a : *\} \cup= \{a : defaut(d)\}$ ⟵ IRCA

OMDASA : Opération de Modification d'un Domaine Agrégé : Suppression d'un Attribut.
Paramètres :
$d \in \mathbf{D}$
a : string
Préconditions :
$a : d' \in d.description$
Actions :
$d.description -= \{a : d'\}$
pour $d_h \in \{d\} \cup \{d'' \in \mathbf{D}, (d'', d, *) \in DuseD\}$
  pour $d_b \in \{d''\} \cup \{d'' \in \mathbf{D}, (d'', d'', *) \in DuseD\}$
    si $(d_h, d_b, n) \in DuseD$ avec $n \geq 2$
      $DuseD -= \{(d_h, d_b, n)\} \cup= \{(d_h, d_b, n-1)\}$
    sinon
      $DuseD -= \{(d_h, d_b, n)\}$
pour $(c, d) \in CuseD$

pour $(o, c) \in OrealC$
pour $a : d \in c.description$
$o.description -= \{a : *\} \cup = \{a : defaut(d)\}$ ⟵ IRCA
pour $(r, d) \in RuseD$
pour $(o, r) \in OuseR$
pour $rc : r \in o.topologie$
pour $a : d \in r.description$
$rc.description -= \{a : *\} \cup = \{a : defaut(d)\}$ ⟵ IRCA

OMRAA : Opération de Modification d'une Relation : Ajout d'un Attribut.

Paramètres :
$r \in \mathbf{R}$
a : string
$d \in \mathbf{D}$
Préconditions :
$a : * \notin r.description$ ⟵ INAR
Actions :
pour $(r, r_s) \in RgenR$
$r_s.description \cup = \{a : d\}$ ⟵ IHRG
$RuseD \cup = \{(r_s, d)\}$
pour $(o, r_s) \in OuseR$
pour $rc : r_s \in o.topologie$
$rc.description \cup = \{a : defaut(d)\}$ ⟵ IRCG

OMRMA : Opération de Modification d'une Relation : Modification du domaine d'un Attribut.

Paramètres :
$r \in \mathbf{R}$
a : string
$d \in \mathbf{D}$
Préconditions :
$a : d' \in r.description$
pour $(r', r) \in RgenR$
si $(r' \neq r) \wedge (a : d' \in r'.description)$ (en respectant les renommages)
$d \leq d'$ ⟵ IHRA
Actions :
pour $(r, r_s) \in RgenR$ (en respectant les renommages)
$r_s.description -= \{a : d'\} \cup = \{a : d\}$ ⟵ IHRG
si $* : d' \notin r_s.description$
$RuseD -= \{(r_s, d')\}$
$RuseD \cup = \{(r_s, d)\}$
pour $(o, r_s) \in OuseR$
pour $rc : r_s \in o.topologie$

$rc.description -= \{a : *\} \cup= \{a : defaut(d)\}$ ⟵ IRCG

OMRSA : Opération de Modification d'une Relation : Suppression d'un Attribut.

Paramètres :
$r \in \mathbf{R}$
a : string
Préconditions :
$a : d \in r.description$
pour $(r', r) \in RgenR$
si $r' \neq r$
$a : * \notin r'.description$ (en respectant les renommages) ⟵ IHRG
Actions :
pour $(r, r_s) \in RgenR$ (en respectant les renommages)
$r_s.description -= \{a : d\}$ ⟵ IHRG
si $* : d \notin r_s.description$
$RuseD -= \{(r_s, d)\}$
pour $(o, r_s) \in OuseR$
pour $rc : r_s \in o.topologie$
$rc.description -= \{a : *\}$ ⟵ IRCG

OMRAC : Opération de Modification d'une Relation : Ajout d'un Composant.

Paramètres :
$r \in \mathbf{R}$
ro : string
$c \in \mathbf{C}$
Préconditions :
$ro : * \notin r.composition$ ⟵ INCR
Actions :
pour $(r, r_s) \in RgenR$
$r_s.composition \cup= \{ro : c\}$ ⟵ IHRG
$RuseC \cup= \{(r_s, c)\}$
pour $(c, r_s) \in CuseR$
pour $l : r_s \in c.topologie$
$OMCSL(c, l)$
pour $(o, r_s) \in OuseR$
pour $rc : r_s \in o.topologie$
$OMOSL(c, rc)$

OMRMC : Opération de Modification d'une Relation : Modification de la catégorie d'un Composant.

Paramètres :
$r \in \mathbf{R}$
ro : string

$c \in \mathbf{C}$
Préconditions :
$ro : c' \in r.composition$
pour $(r', r) \in RgenR$
si $(r' \neq r) \wedge (ro : c' \in r'.composition)$ (en respectant les renommages)
$(c', c) \in CgenC$ ⟵ IHRC
Actions :
pour $(r, r_s) \in RgenR$ (en respectant les renommages)
$r_s.composition -= \{ro : c'\} \cup = \{ro : c\}$ ⟵ IHRG
si $* : c' \notin r_s.composition$
$RuseC -= \{(r_s, c')\}$
$RuseC \cup = \{(r_s, c)\}$
pour $(c, r_s) \in CuseR$
pour $l : r_s \in c.topologie$
$OMCSL(c, l)$
pour $(o, r_s) \in OuseR$
pour $rc : r_s \in o.topologie$
$OMOSL(c, rc)$

OMRSC : Opération de Modification d'une Relation : Suppression d'un Composant.
Paramètres :
$r \in \mathbf{R}$
ro : string
Préconditions :
$ro : c \in r.composition$
$card(r.composition) \geq 2$
pour $(r', r) \in RgenR$
si $r' \neq r$
$ro : * \notin r'.composition$ (en respectant les renommages) ⟵ IHRG
Actions :
pour $(r, r_s) \in RgenR$ (en respectant les renommages)
$r_s.composition -= \{ro : c\}$ ⟵ IHRG
si $* : c \notin r_s.composition$
$RuseC -= \{(r_s, c)\}$
pour $(c, r_s) \in CuseR$
pour $l : r_s \in c.topologie$
$l.composition -= \{ro : *\}$ ⟵ IRCG
pour $(o, r_s) \in OuseR$
pour $rc : r_s \in o.topologie$
$rc.composition -= \{ro : *\}$ ⟵ IRCG

OMCAA : Opération de Modification d'une Catégorie : Ajout d'un Attribut.
Paramètres :

$c \in \mathbf{C}$
a : string
$d \in \mathbf{D}$
Préconditions :
$a : * \notin c.description$ ⟵ INAC
Actions :
pour $(c, c_s) \in CgenC$
$c_s.description \cup = \{a : d\}$ ⟵ IHCG
$CuseD \cup = \{(c_s, d)\}$
pour $(o, c) \in OrealC$
$o.description \cup = \{a : defaut(d)\}$ ⟵ IRCG

OMCMA : Opération de Modification d'une Catégorie : Modification du domaine d'un Attribut.
Paramètres :
$c \in \mathbf{C}$
a : string
$d \in \mathbf{D}$
Préconditions :
$a : d' \in c.description$
pour $(c', c) \in CgenC$
si $(c' \neq c) \wedge (a : d' \in c'.description)$ (en respectant les renommages)
$d \leq d'$ ⟵ IHCA
Actions :
pour $(c, c_s) \in CgenC$ (en respectant les renommages)
$c_s.description - = \{a : d'\} \cup = \{a : d\}$ ⟵ IHCG
si $* : d' \notin c_s.description$
$CuseD - = \{(c_s, d')\}$
$CuseD \cup = \{(c_s, d)\}$
pour $(o, c) \in OrealC$
$o.description - = \{a : *\} \cup = \{a : defaut(d)\}$ ⟵ IRCG

OMCSA : Opération de Modification d'une Catégorie : Suppression d'un Attribut.
Paramètres :
$c \in \mathbf{C}$
a : string
Préconditions :
$a : d \in c.description$
pour $(c', c) \in CgenC$
si $c' \neq c$
$a : * \notin c'.description$ (en respectant les renommages) ⟵ IHCG
Actions :
pour $(c, c_s) \in CgenC$ (en respectant les renommages)

$c_s.description -= \{a : d\}$ ⟵ IHCG
$si * : d \notin c_s.description$
$CuseD -= \{(c_s, d)\}$
pour $(o, c) \in OrealC$
$o.description -= \{a : *\}$ ⟵ IRCG

OMCAC : Opération de Modification d'une Catégorie : Ajout d'un Composant.
Paramètres :
$c \in \mathbf{C}$
ro : string
$c' \in \mathbf{C}$
$t \in \{"D", "I"\}$
Préconditions :
$ro : * \notin c.composition$ ⟵ INCC
Actions :
pour $(c, c_s) \in CgenC$
$c_s.composition \cup = \{ro : c'(t)\}$ ⟵ IHCG
$CuseC \cup = \{(c_s, c')\}$
pour $(o, c) \in OrealC$
$OCO\,(o.ro\,,\, c')$
$o.composition \cup = \{ro : o.ro(tO)\}$ ⟵ IRCG
(le nouveau type de lien tO est obtenu en ajoutant "O" (obligatoire) au type de lien t)
$OuseO \cup = \{(o, o.ro)\}$

OMCMCC : Opération de Modification d'une Catégorie : Modification de la Catégorie d'un Composant.
Paramètres :
$c \in \mathbf{C}$
ro : string
$c' \in \mathbf{C}$
Préconditions :
$ro : c''(t) \in c.composition$
pour $(c_g, c) \in CgenC$
$si\,(c_g \neq c) \wedge (ro : c''(t) \in c_g.composition)$ (en respectant les renommages)
$(c'', c') \in CgenC$ ⟵ IHCC
Actions :
pour $(c, c_s) \in CgenC$ (en respectant les renommages)
$c_s.composition -= \{ro : c''(t)\} \cup = \{ro : c'(t)\}$ ⟵ IHCG
$si * : c'' \notin c_s.composition$
$CuseC -= \{(c_s, c'')\}$
$CuseC \cup = \{(c_s, c')\}$
$si\,(c'', c') \notin CgenC$

pour $(c, c_s) \in CgenC$ (en respectant les renommages)
pour $l : r \in c_s.topologie$
si $* : ro \in l$
$OMCSL\,(c_s, l)$
pour $(o, c) \in OrealC$
$OCO\,(o.ro\,,\, c')$
$OMOMOC\,(o, ro, o.ro)$ ⟵ IRCG

OMCMTC : Opération de Modification d'une Catégorie : Modification du Type de lien pour un Composant.
Paramètres :
$c \in \mathbf{C}$
ro : string
$t \in \{"D", "I"\}$
Préconditions :
$ro : c'(t') \in c.composition$
pour $(c_g, c) \in CgenC$
si $(c_g \neq c) \wedge (ro : c''(t'') \in c_g.composition)$ (en respectant les renommages)
$t \leq t''$ ⟵ IHCTC
Actions :
pour $(c, c_s) \in CgenC$ (en respectant les renommages)
$c_s.composition -= \{ro : c'(t')\} \cup= \{ro : c'(t)\}$ ⟵ IHCG
pour $(o, c) \in OrealC$
$o.composition -= \{ro : o'(t'')\} \cup= \{ro : o'(t'O)\}$ ⟵ IRCG
(le nouveau type de lien $t'O$ est obtenu en ajoutant "O" (obligatoire) au type de lien t')

OMCSC : Opération de Modification d'une Catégorie : Suppression d'un Composant.
Paramètres :
$c \in \mathbf{C}$
ro : string
Préconditions :
$ro : c'(t) \in c.composition$
pour $(c_g, c) \in CgenC$
si $c_g \neq c$
$ro : * \notin c_g.composition$ (en respectant les renommages) ⟵ IHCG
Actions :
pour $(c, c_s) \in CgenC$ (en respectant les renommages)
$c_s.composition -= \{ro : c'\}$ ⟵ IHCG
si $* : c' \notin c_s.composition$
$CuseC -= \{(c_s, c')\}$
pour $l : r \in c_s.topologie$
si $* : ro \in l$

$OMCSL(c_s, l)$
pour $(o, c) \in OrealC$
$OMOSC(o, ro, *)$ ⟵ IRCG

OMCAL : Opération de Modification d'une Catégorie : Ajout d'un Lien.
Paramètres :
$c \in \mathbf{C}$
l : string
$r \in \mathbf{R}$
$ro_1, ..., ro_n$ strings
Préconditions :
$r.composition = \{ro'_1 : c'_1, ..., ro'_n : c'_n\}$
pour $i \in [1, n]$
$ro_i : c_i \in c.composition$
$(c'_i, c_i) \in CgenC$
Actions :
$l := \{ro'_i : ro_i, 1 \leq i \leq n\}$
pour $(c, c_s) \in CgenC$
$c_s.topologie \cup = \{l : r\}$ ⟵ IHCG
$CuseR \cup = \{(c_s, r)\}$
pour $(o, c) \in OrealC$
pour $i \in [1, n]$
$o_i :=$ objet jouant le rôle ro_i dans $o.composition$
$OMOAL(o, r, o_1, ..., o_n)$ ⟵ IRCG

OMCML : Opération de Modification d'une Catégorie : Modification d'un composant de Lien.
Paramètres :
$c \in \mathbf{C}$
l : string
ro, ro' strings
Préconditions :
$l : r \in c.topologie$
$ro' : c' \in r.composition$
$ro : c'' \in c.composition$
$(c', c'') \in CgenC$
Actions :
pour $(c, c_s) \in CgenC$ (en respectant les renommages)
$l' := l - = \{ro' : *\} \cup = \{ro' : ro\}$
$c_s.topologie - = \{l : r\} \cup = \{l' : r\}$ ⟵ IHCG
pour $(o, c) \in OrealC$
$o' :=$ objet jouant le rôle ro dans $o.composition$
$OMOMCL(o, l, ro', o')$ ⟵ IRCG

OMCSL : Opération de Modification d'une Catégorie : Suppression d'un Lien.

Paramètres :
$c \in \mathbf{C}$
l : string
Préconditions :
$l : r \in c.topologie$
pour $(c_g, c) \in CgenC$
si $c_g \neq c$
$l : * \notin c_g.topologie$ (en respectant les renommages) ⟵ IHCG
Actions :
pour $(c, c_s) \in CgenC$ (en respectant les renommages)
$c_s.topologie -= \{l : r\}$ ⟵ IHCG
si $* : r \notin c_s.topologie$
$CuseR -= \{(c_s, r)\}$
pour $(o, c) \in OrealC$
$OMOSL(o, l)$ ⟵ IRCG

OMOMA : Opération de Modification d'un Objet : Modification de la valeur d'un Attribut.

Paramètres :
$o \in \mathbf{O}$
a : string
$v \in \mathbf{V}$
Préconditions :
$\exists c \in \mathbf{C}, ((o, c) \in OrealC) \wedge (a : d \in c.description) \wedge (v \in V(d))$ ⟵ IRCA
Actions :
$o.description -= \{a : v'\} \cup= \{a : v\}$ ⟵ IRCG

OMOAC : Opération de Modification d'un Objet : Ajout d'un Composant.

Paramètres :
$o \in \mathbf{O}$
ro : string
$o' \in \mathbf{O}$
$t \in \{"DO", "IO", "Do", "Io"\}$
Préconditions :
Actions :
$o.composition \cup= \{ro : o'(t)\}$
$OuseO \cup= \{(o, o')\}$

OMOMOC : Opération de Modification d'un Objet : Modification de l'Objet d'un Composant.

Paramètres :
$o \in \mathbf{O}$
ro : string
$o' \in \mathbf{O}$
Préconditions :
$\exists c \in \mathbf{C}, ((o,c) \in OrealC) \wedge (ro : c'(t) \in c.composition) \wedge ((o',c') \in OrealC)$ ⟵ IRCC
$ro' : o'(t') \in o.composition$
Actions :
$o.composition -= \{ro : o''(t'')\} \cup= \{ro : o'(tO), "vide" : o''(t'')\}$ ⟵ IRCG
(le nouveau type de lien tO est obtenu en ajoutant "O" (obligatoire) au type de lien t)
pour $(o,c) \in OrealC$
  pour $l : r \in c.topologie$
    si $ro'' : ro \in l$
      $OMOMCL(o,l,ro'',o')$ ⟵ IRCG

OMOMTC : Opération de Modification d'un objet : Modification du Type de lien pour un Composant.
Paramètres :
$o \in \mathbf{O}$
ro : string
$o' \in \mathbf{O}$
$t \in \{"DO","IO","Do","Io"\}$
Préconditions :
$ro : o'(t') \in o.composition$
pour $(o,c) \in OrealC$
  si $ro : c'(t'') \in c.composition$ (en respectant les renommages)
    $t \leq t''$ ⟵ IRCTC
Actions :
$o.composition -= \{ro : o'(t')\} \cup= \{ro : o'(t)\}$ ⟵ IRCG

OMOSC : Opération de Modification d'un Objet : Suppression d'un Composant.
Paramètres :
$o \in \mathbf{O}$
ro : string
$o' \in \mathbf{O}$
Préconditions :
$ro : o'(t) \in o.composition$
pour $(o,c) \in OrealC$
  $ro : * \notin c.composition$ ⟵ IRCG
Actions :
$o.composition -= \{ro : o'(t)\}$ ⟵ IRCG

$OuseO -= \{(o, o')\}$
pour $cr : r \in o.topologie$
    si $ro' : o' \in cr.composition$
        $OMOSL(o, cr)$ ⟵ IRCG

OMOAL : Opération de Modification d'un Objet : Ajout d'un Lien.

Paramètres :
$o \in \mathbf{O}$
rc : string
$r \in \mathbf{R}$
$o_1, ..., o_n \in \mathbf{O}$
Préconditions :
pour $i \in [1, n]$
    $ro_i : o_i \in o.composition$
$r.composition = \{ro'_1 : c_1, ..., ro'_n : c_n\}$
pour $i \in [1, n]$
    $(o_i, c_i) \in OrealC$
Actions :
$o.topologie \cup= \{rc : r\}$
pour $a : d \in r.description$
    $rc.description \cup= \{a : defaut(d)\}$
pour $i \in [1, n]$
    $rc.composition \cup= \{ro'_i : o_i\}$ ⟵ IRCL
$OuseR \cup= \{(o, r)\}$

OMOMAL : Opération de Modification d'un Objet : Modification de la valeur d'un Attribut de Lien.

Paramètres :
$o \in \mathbf{O}$
rc, a : strings
$v \in \mathbf{V}$
Préconditions :
$rc : r \in o.topologie$
$a : d \in r.description$
$v \in V(d)$ ⟵ IRCL
Actions :
$rc.description -= \{a : *\} \cup= \{a : v\}$ ⟵ IRCL

OMOMCL : Opération de Modification d'un Objet : Modification de l'objet d'un Composant de Lien.

Paramètres :
$o \in \mathbf{O}$
rc, ro : strings

$o' \in \mathbf{O}$
Préconditions :
$rc : r \in o.topologie$
$ro : c \in r.composition$
$o' \in o.composition$
$(o', c) \in OrealC$ ⟵ IRCL
Actions :
$rc.composition -= \{ro : *\} \cup= \{ro : o'\}$ ⟵ IRCL

OMOSL : Opération de Modification d'un Objet : Suppression d'un Lien.
Paramètres :
$o \in \mathbf{O}$
rc : string
Préconditions :
$rc : r \in o.topologie$
Actions :
$o.topologie -= \{rc : r\}$ ⟵ IRCL
$si * : r \notin o.topologie$
$OuseR -= \{(o, r)\}$

Annexe C

Implantation d'EMIR au dessus d'un SGBDOO

Dans cette partie nous ne présentons qu'une ébauche de travail [LAH96], qui doit être approfondie ultérieurement. Nous posons certaines restrictions sur le modèle EMIR, qui doivent être levées pour étendre l'étude au modèle général.

L'idée initiale de cette étude est le besoin de définir le modèle EMIR au dessus d'un SGBDOO, comme une évolution du prototype EMIR actuel, implémenté dans l'environnement Smalltalk-80. Autrement dit, au lieu d'utiliser le modèle objet de Smalltalk comme méta-modèle, utiliser plutôt le modèle d'un SGBDOO. L'intérêt est d'utiliser les performances du SGBD pour stocker de larges quantités de données.

Comme pour Smalltalk, il s'agit de définir un schéma des concepts du modèle EMIR au sein du SGBDOO hôte. Mais comme celui-ci dispose d'un langage de requêtes, nous allons aussi traduire les requêtes EMIR en requêtes du SGBDOO hôte.

La première question qui se pose est le choix du SGBDOO. O_2 est le premier SGBDOO français ; il est très répandu dans les universités françaises. Il nous a donc intéressé en premier lieu. Cependant, on n'en est pas encore à l'implémentation, mais simplement aux spécifications.

Pour spécifier le schéma EMIR et traduire ses requêtes, notre choix s'est posé sur le standard ODMG-93 [CAT94]. Ce standard a été défini par un certain nombre de constructeurs de SGBDOO, dont O_2, *Versant*, *Ontos* ... afin de donner le jour à une plateforme conceptuelle pour l'interopérabilité de leurs produits. Une spécification en ODMG-93 peut donc se traduire dans chacun des environnements des constructeurs qui y ont participé.

ODMG-93 définit un modèle orienté objet à base de classes, pour SGBDOO. Une classe a un nom, un éventuel ensemble de super-classes, une extension (ensemble nommé d'objets instanciant la classe), un ensemble d'attributs, un ensemble de liens avec d'autres classes et enfin, un ensemble d'opérations.

Un attribut est un lien d'une classe vers un domaine de valeurs, qui peuvent être simples (entiers, réels, ...), groupées (ensembles, listes, ...), structurées, ...

Un lien entre deux classes peut être de trois types : un-à-un, un-à-plusieurs ou plusieurs-à-plusieurs, selon les cas qui peuvent se présenter au niveau des instances. Un lien entre deux classes peut être traversé d'un seul côté ou des deux, on parle alors de liens symétriques. Chacun est alors désigné dans une classe, par un nom de lien, faisant aussi référence au lien symétrique.

Une opération possède une signature qui est un ensemble d'arguments et un éventuel résultat. Arguments et résultat sont soit des classes soit des domaines de valeurs. Les opérations peuvent soulever des exceptions à l'exécution ; on peut alors leur attacher des procédures à exécuter en cas d'exception.

Nous allons d'abord décrire dans C.1 quelques méta-classes d'EMIR dans le formalisme d'ODL (Object Description Language), le langage de description de données défini par l'ODMG [CAT94]. Ensuite, nous présentons dans C.2 une méthode de traduction des requêtes EMIR dans OQL (Object Query Language), le langage de requêtes défini par l'ODMG [CAT94].

C.1 Méta-classes

On va décrire par exemple les méta-classes *Category*, *Mandatory_ Component* et *Object_ Component*, dans ODL.

Le mot-clé `Interface` sert à déclarer une classe. Le mot-clé `extent` désigne son extension : nom d'un ensemble d'objets formé par toutes les instances de la classe. Le mot-clé `keys` désigne un ou plusieurs attributs qui identifient les instances de la classe.

Les attributs de la classe (resp. ses liens avec d'autres classes) sont précédés par le mot-clé `attribute` (resp. `relationship`). Le mot-clé `inverse` désigne le lien inverse d'un lien dans la classe liée.

Chaque méthode de la classe est représentée par son nom et sa signature. Le mot-clé `in` indique les paramètres en entrée de la méthode. Le mot-clé `raises` précède des codes de levée d'exceptions éventuelles.

```
Interface Category {
    extent categories;
    keys name;

    attribute string name;
    relationship List<Attribute> description;
    relationship List<Category Component> composition;
    relationship List<Link> topology;
    relationship List<Category> inherits
        inverse Category::inherited_by;
    relationship List<Category> inherited_by
        inverse Category::inherits;
    relationship List<Object> realizations
        inverse Object::category;
    relationship List<Relation Component> relRef
        inverse Relation_Component::category;

    addAttribute (in Attribute);
    removeAttribute (in Attribute) raises (non_existing_attribute);
    addComponent (in Category Component);
    removeAttribute (in Category Component) raises (non_existing_component);
    addLink (in Link);
    removeLink (in Link) raises (non_existing_link);
    addGeneric (in Category) raises (cyclic_inheritance);
    removeGeneric (in Category) raises (non_generic);

    addAttribute (in attribute);
    removeAttribute (in Attribute) raises (non_existing_attribute)
    ... autres op\'erations
}
```

```
Interface Mandatory_Component: Object_Component {
    extent mandatory_components;

    relationship Category_Component catComp
        inverse Category_Component::objRef;
}

Interface Object_Component {
    extent object_components;

    relationship Object object
        inverse Object::object_references;
    relationship List<Relationship_Component> relRef
        inverse Relationship_Component::objComp;

    update_object (in Object);
    ... autres op\'erations
}
```

En fait, la description dans ODL des méta-classes EMIR découle très simplement du schéma de la figure 4.1.

C.2 Traduction de requêtes

La syntaxe d'OQL est assez simple et assez proche de celle de SQL pour les bases de données relationnelles.

Une requête OQL est définie par la donnée de:

- un ensemble de définitions préalables ("*Query Definitions*") `{qd}`, qui forment un contexte pour la requête; une définition `qd: define v as e` affecte une expression `e` à une variable `v` locale à la requête et destinée à être utilisée dans le calcul du résultat,
- une expression du résultat de la requête ("*Query Expression*") `qe`, calculée sur le contenu de la base et qui utilise en particulier les valeurs des définitions préalables.

Il y a plusieurs types d'expressions [CAT94]:

- expression basiques (littéraux),
- expression arithmétiques (`qe1 + qe2`),
- expression de comparaison (`qe1 > qe2`),
- expression d'accès à la valeur d'un attribut (`qe.nom_attribut`),
- expression d'accès à la destination d'un lien (`qe->nom_lien`),

- expression de collection (`select qe1 from qe2 where qe3`),
- ...

Soit $q = (c, p, Cl, Q, L)$ une requête EMIR.

Soit $p = p_1...p_i...p_n$ le chemin de projection de q.

Pour simplifier la présentation, nous restreignons ce chemin à être composé uniquement de composants de catégorie (pas d'attributs ou de liens de catégories EMIR).

Soit $Cl = \{cl_1, ..., cl_j, ..., cl_m\}$ l'ensemble des critères de q.

Nous restreignons aussi chaque critère cl_j à une simple expression de comparaison entre deux chemins valides pour c, composés uniquement de composants de catégorie: $cl_j : p_{j1}...p_{jr} = p'_{j1}...p'_{js}$.

Soit $Q = \{q_1, ..., q_k, ..., q_p\}$ l'ensemble des sous-requêtes de q.

Soit $L = \{l_1, ..., l_l, ..., l_q\}$ l'ensemble des liens de q.

Nous nous restreindrons à des liens reliés à une relation r, qui n'a pas d'attributs et qui a deux composants, nommés *premier* et *second*, hérités de Any_r et reliés à la catégorie Any_c. De plus, nous restreignons les liens à mettre en jeu uniquement des sous-requêtes (pas de composants obligatoires ou de rôle *Self*). Pour $l \in [1..q]$, nous notons: $l_l : r(premier : q_{l_1}, second : q_{l_2})$.

Pour traduire la requête q en une requête OQL, nommée `oq`, nous supposons que `oq_1`, ..., `oq_p` sont les traductions en OQL des sous-requêtes q_1, ..., q_p.

D'abord, nous définissons une requête OQL, donnant la liste des objets concrets qui réalisent la catégorie c.

```
define objects as (element (select x
                            from x in Category
                            where x.name = c))->realizations
```

Pour chaque critère cl_j, nous définissons une requête `qc_j` qui restreint cette liste d'objets à ceux qui satisfont ce critère.

Pour cela, nous calculons les destinations successives `S_j1`, ..., `S_jr` et `S'_j1`, ..., `S'_js` des chemins $p_{j1}...p_{jr}$ et $p'_{j1}...p'_{js}$ sur les éléments de `objects`.

Pour des raisons d'uniformité de notation, nous définissons `S_j0` et `S'_j0` comme étant tous les deux égaux à `objects`.

Pour $u \in [1..r]$

```
define S_ju as (select x->object
                from x in (select (element (select y
                                            from y in z->composition
                                            where y->catComp.name = "p_ju"))
                           from z in S_j(u-1)))
```

Pour $v \in [1..s]$

```
define S'_jv as (select x->object
```

```
                from x in (select (element (select y
                                            from y in z->composition
                                            where y->catComp.name = "p'_jv"))
                                  from z in S'_j(v-1)))
```

A présent, pour chaque critère Cl_j nous restreignons successivement `objects` aux objets qui satisfont le critère, par les requêtes `qc_j` suivantes. Nous définissons aussi `qc_0` comme étant égal à `objects`.

Pour $j \in [1..m]$

```
define array_c_j as array(1,2,...,count[qc_{j-1}])
define qc_j as (select qc_{j-1}[i]
                from i in array_c_j
                where S_jr[i] = S'_js[i])
```

La liste finale `qc_m` est composée des objets qui réalisent la catégorie c et qui satisfont tous les critères.

Ensuite, pour chaque sous-requête q_k, nous définissons une requête OQL nommée `qs_k`, qui donne, pour chaque membre de `qc_m`, la liste de ses composants qui satisfont q_k.
Ces listes seront utilisées autant pour traduire les sous-requêtes que les liens.

Pour $k \in [1..p]$

```
define qs_k as (select (select x->object
                        from x in o->composition
                        where x->object in oq_k)
                from o in qc_m)
```

Les objets satisfaisant les sous-requêtes sont ceux dont *toutes* les listes `qs_k` correspondantes sont non vides.

```
define array_s as array(1,2,...,count[qc_{m}])
define qs as (select qc_m[i]
              from i in array_s
              where qs_1[i] "non empty" and ... and qs_p[i] "non empty")
```

Nous passons maintenant aux liens l_1, ..., l_l, ..., l_q. Pour chaque l_l : $r(premier : q_{l_1}, second : q_{l_2})$, nous restreignons la liste précédente `qs` aux objets qui respectent ce lien, par l'intermédiaire de la requête `qt_l`. Encore une fois, nous définissons `qt_0` comme étant égal à `qs`.

Pour $l \in [1..q]$

```
define array_t_l as array(1,2,...,count[qt_(l-1)])
define qt_l as (select qt_(l-1)[i]
```

```
from i in array_t_1
where exists link in qt_1[i]->topology:
  element (select x->object
           from x in link->composition
           where x->relComp.name = "first") in qs_11[i]
  and
  element (select x->object
           from x in link->composition
           where x->relComp.name = "second") in qs_12[i]
  and
  link->relation.name = "r")
```

Enfin, nous projetons le résultat `qt_q` à travers le chemin de projection $p_1...p_i...p_n$, en utilisant les requêtes `qp_i` suivantes. Nous définissons (une dernière fois) `qp_0` comme étant égal à `qt_q`.

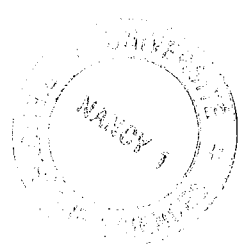
Pour $i \in [1..n]$

```
define qp_i as (select x->object
                from x in (select (element (select y
                                            from y in z->composition
                                            where y->catComp.name = "p_i"))
                           from z in qp_(i-1)))
```

L'expression de la requête `oq` est égale à la requête finale `qp_n`. Les requêtes intermédiaires sont ses définitions préalables.

La méthode de traduction de requêtes EMIR en requêtes OQL, présentée ci dessus, n'est évidemment pas optimisée. Son seul objectif est de montrer que c'est réalisable. Elle doit être améliorée avant son implémentation.

Signalons simplement que la technique d'exécution de requêtes dans le prototype EMIR actuel, programmé dans l'environnement Smalltalk, suit à peu près les mêmes étapes : filtre de catégorie, filtres de critères, filtres de sous-requêtes et enfin filtres de liens topologiques.



Université HENRI POINCARÉ NANCY 1

Nom : **LAHLOU**

Prénom : **Youssef**

DOCTORAT de l'UNIVERSITE HENRI POINCARE, NANCY-I

en INFORMATIQUE

VU, APPROUVÉ ET PERMIS D'IMPRIMER

Nancy, le 20 JUIN 1996 UHP 45

Le Président de l'Université

UNIVERSITÉ HENRI POINCARÉ NANCY 1 LE PRÉSIDENT FINANCE

3C-F9

UNIVERSITÉ HENRI POINCARÉ - NANCY 1
24-30, rue Lionnois B.P.3069 54013 Nancy cedex
Tél : 83 32 81 81 Fax : 83 32 95 90
Adresse électronique : _____ @presn1.u-nancy.fr