Relaxing the Instantiation Link: Towards a Content-Based Data Model for Information Retrieval

Youssef Lahlou Noureddine Mouaddib

CRIN-CNRS,
BP 239, Campus scientifique,
54506 Vandoeuvre-les-Nancy, France.
Tel : (33) 83 59 20 61.
Fax : (33) 83 41 30 79.
E-Mails : {lahlou, mouaddib}@loria.fr

Abstract

In this article, we define a semantic data model, where objects' structures are richer than those of their classes. This model is intended to be used in applications such as content-based information retrieval, whose particularity is the fact that objects do not have a predefined conceptual structure that can be fully abstracted in a class schema.

For this purpose, we relax the classical instantiation link between classes and objects, and rebaptize it the "realization" link.

Some of the problems that arise because of this relaxation are investigated, especially when it comes to querying the database. We present a query language for the model that respects the realization link.

The implementation of the model uses a class-based object-oriented model as a meta-model. All of our model concepts are classes in the meta-model, that will be instanciated, for each application of the model, into objects representing all the application features (classes, objects, attributes, queries, ...).

Key Words:

Semantic Model, Information Retrieval, Relaxed Instantiation, Realization Link, Query Language, Meta-Model.

Introduction

The present study focuses on data models for handling complex information (as commonly encountered in CAD-CAM and multimedia database systems),

in comparison with Information Retrieval applications where objects are not very structured. The major task is to find a suitable mapping for information about real-world entities into machine representations which, from the user's conceptual viewpoint, ensures the least loss in expressiveness for the entities and enhances user expectations about the retrieval process.

The Relational Model [COD70] (along with its extensions [RKS88]) was the first data model to provide an efficient representation theory and a mechanism for query evaluation. It however turns out to be strongly limited when it comes to coping with complex information. One reason for this is that such a model disseminates semantic information over a large set of relations, the pertinence of which is not always obvious to the user. A second reason for this has to do with the value-based characteristic of the relational model [HUL89].

Semantic Data Models [HK87] arose as an attempt to overcome the inherent limitations of the relational theory. Their major contribution is to bridge the semantic gap between the user's perception of a real-world application and his conceptual and, to a lesser extent, physical representation of the application. Semantic models played an important role as a transition from relational models towards the new generation of object-oriented database environments [CAT94].

Unstructured objects such as texts and images handled by Information Retrieval systems cannot be modelled through only simple use of hierarchical structures, or abstraction into class structures. Common applications more often require object content indexing rather than grouping into well-structured classes. Furthermore, unstructured objects may also refer to structured objects and vice versa, especially in multimedia environments [WLK87].

The proposed model is able to cope with the representation and the manipulation of both unstructured objects as encountered in content-based models as in Information Retrieval, and structured objects as in semantic or object-oriented models.

Systems based on the relational data model and dedicated to cope with unstructured objects (such as images) are encountered in the current literature [HLMM92]. Our concern is to define a semantic data model, capable of responding to a wide range of specific problems related to data modelling in Information Retrieval. The model draws its conceptual inspiration from Semantic Data Models, and allows objects to have an individual flexible structure, rather than their classes'.

We thus relax the traditional instantiation link between an object and its class, and rebaptize it the "realization link". A class is no longer a set of objects having the same structure; it is only a minimal structure that have to be implemented by each object tied to it.

This loss of similarity between object and class structures induces some problems when it comes to querying the database. We explore such problems and show that they can be overcome by embedding our whole model into a higher (meta) level (structured) model.

Section 1 presents an overview of data models and information retrieval. Section 2 then sets out an informal definition of the concepts of our model while section 3 gives a formal specification of them. Section 4 presents the query specification language while section 5 describes an implementation of our model and an image application of it. We then conclude by setting out the future development envisaged for study.

1 Data Models and Information Retrieval

In this section, we explore the main approaches for modelling complex information in 1.1; we present the specificities of Information Retrieval applications in 1.2. This will lead us to give some motivations behind the definition of a new model, in 1.3.

1.1 Complex objects modelling

The relational model with its 1NF constraint provides a less suitable environment for handling complex information. As mentioned above, the pertinence of the associated sets of relations is not always obvious and natural to the user.

Attempts have been made to extend, improve and overcome the limitations of the relational model through the introduction of surrogates [COD79] and/or more complex data constructors than the flat classical relations [RKS88]. Advanced relational models (based on NF2 (e.g. nested) relations) provide a way of using the power of the relational calculus and algebra to deal with complex information. Such complex object models provide some abstractions for object modelling (aggregation, grouping, ...) whose effects on algebra and calculus are investigated. The inherent value-based approach does however represent a major limitation for complex modeling [HUL89].

Semantic Data Models arose as an attempt to bridge the gap between the application semantics and the logical machine representation of the data it carries. Even though every semantic model has its own concepts and characteristics, they all share some underlying concepts [HK87] such as:

- Classes of objects,
- Standard abstraction of object types (generalization, specialization, association, ...),
- Structured types of objects (aggregation, grouping),
- Semantic relationships between objects.

The Object-Oriented approach [KIM90, CAT94] combines advances in programming languages and knowledge representation in Artificial Intelligence. An object has a structure (static characteristics) and a behavior (associated procedures). These two aspects are encapsulated within the object and one can access

the structure only through the associated behavior. Objects are abstracted into classes with each object being an instance of one or more classes. A class is thus a collection of objects sharing a common structure and behavior. The common structure is a collection of instance variables. An instance variable refers to an object class and has, as a value, an object of that class. The inheritance mechanism allows the transmission of structure and behavior from a generic class to specific ones. With the exception of object behavior, (the static part of) the object-oriented approach embraces the major characteristics of semantic data models.

1.2 Information Retrieval

In some applications, the manipulated objects are not easy to structure and group into classes, because of the different levels of description required and the possibility of introducing subjective descriptions. Text, image and other multimedia databases are examples of such applications.

In the preceding applications, a distinction is made between three description levels [KKL91]:

- the primitive data level: the contents of the physical representation of objects before any analysis or indexing (strings, pixel matrixes, ...),

- the contextual data level: this concerns object contexts (length or width of images, words count in texts, author name, ...),

- the semantic data level: this embraces information contained in, referenced or connotated by the objects. They are the most difficult to model and require several specific treatements. They may be distinguished into objective and subjective semantic data.

Text databases and text captioning of image databases can be indexed using key-words [HCK90, HLMM92, RS92], or subsequent to a natural language analysis step [SCH90].

Image databases contain more information than do simple text captions or key-words such as graphical items, subjective contexts (cultural connotations, ...) and so on. The authors in [HLMM92] further proposed the distinction between pre-iconographic, iconographic and iconological contents of an image. For the objective part, images can be seen [RS92] as component objects inter-linked by object-to-object relations.

In such database applications, the data model is more an indexing-retrieval model than a structuring model, inasmuch as object categorization (classification) into structures is all but feasible. Some approaches [GPVG89] try to overcome this problem by using a formal grammar instead of a structured model for the description of their objects.

1.3 Motivations

The central aim is to design a data-model capable of coping with two opposing paradigms involved in multimedia-like applications, where an object can either be structured or remain unstructured.

An image in an image database is difficult to stucture into a class (category) since it is practically impossible for the structure (contents) of an image to be totally defined at the conceptual level. All the same, an image may refer to other objects (contained therein) that are structured or need structuring simply because such objects, like cars, houses or horses are intrinsically structured.

To reach our main goal, we propose a data model which adapts to the two information retrieval contexts with facilities for representing classical paradigms of (structured) semantic data models (generalization, aggregation, ...) for the objects in the base, without necessarily imposing on the latter a predefined structure within their classes.

An object in our approach is a weak instantiation or a realization of a class, and has a specific individual structure. An image for instance, has some usual features (e.g size, photographer, date, ...) and additional individual components (e.g persons, flowers, ... whatever). Usual features are abstracted into an image class, that describes them (e.g. the size of an image is a real value). Individual features are defined within a particular image object.

2 Informal definition of the concepts

In this section, we informally present the main concepts of our model starting with object types, moving on through classes to objects. Particularly, we define our intent in relaxing the instantiation link between objects and classes.

2.1 Types and classes

According to the definitions given in [HUL89], we make a distinction between the two different paradigms of *type* and *class*.

An object type is the static definition of a structure. A structure is a distribution of data in a specific way. Each part of the distribution is in turn distributed in a certain way. A type is thus an intentional definition of objects. This definition is similar to classical definitions of object types [HUL89, AB88].

Types are organised into a lattice corresponding to a partial order; that is, a "sub-type" holds the complete definition of a "super-type" and augments it with additional features. Since types are nothing more than structure definitions, wherever a super-type can be used (in the definition of other types), it represents itself along with all its sub-types.

On the other hand, a class is usually a set of objects (instances) of the same type, that is the type of the class itself [HUL89]. In our model a class is not a set of objects. A class is the association of a class name and an object type. The definition of a class is the assignment of a name to an object type and some attribute names to each part of the static structure of the type.

Classes are organized into a lattice of generalization / specialization links. They can therefore be refined and incrementally defined using more specific constraints and new attributes.

2.2 Objects

An object is the machine representation of a real universe (of discourse) entity. It has a unique and exclusive identifier and a structure that involves other objects. This incremental definition of objects referencing each other assumes that there exists a collection of primitive objects, having no references to others and called values or terminal objects.

Realization is a mechanism that links an object to a class. In other words, an object can never exist without a class to which it is tied: in this sense the object is said to be a realization of the associated class.

The mechanism of realization can be seen as the operation of giving a value to the attributes of the class by assigning an object to each of them.

The possibility is then left for objects to have in turn other additional objects in their structure, in addition to those fixed by the class. This enables objects to have an individual composition rather than their abstraction into classes. From this comes the improved flexibility and extensiveness of the model.

Thus, the link between an object and its class is different from the traditional "instantiation" link, where an instance of a class has exactly the structure abstracted in that class.

According to the definition of super-types and sub-types, we can characterize the realization mechanism by the fact that the type of the realization (the object) is a sub-type of the class type, since it holds information and structure, additional to that of the class type.

As we stated in paragraph 1.3, an image object holds more information than does the image class, to which it is tied: flowers, hourses, ... (or whatever appears in the image), are added to the common image features e.g. size, photographer, ... etc.

2.3 Overview

It follows from the preceding presentation that our model has two related levels of abstraction, a concrete level (objects) and a conceptual level (classes, types).

For the purpose of taking the realization mechanism into account, we use a third level of abstraction called the meta-level. In this level, all the concepts of the model (classes, objects, ...) are described in terms of meta-classes, as is summarized in figure 1. This figure points out the strategy of implementation of the model within a class-based object-oriented model, such as that presented in [CAT94].

2.4 Queries

Queries are intentional expressions on the structures of objects satisfying them. For this purpose, the query specifier is aware, at query specification time, of the structures of the classes (s)he will use in his (her) query, knowing that instances

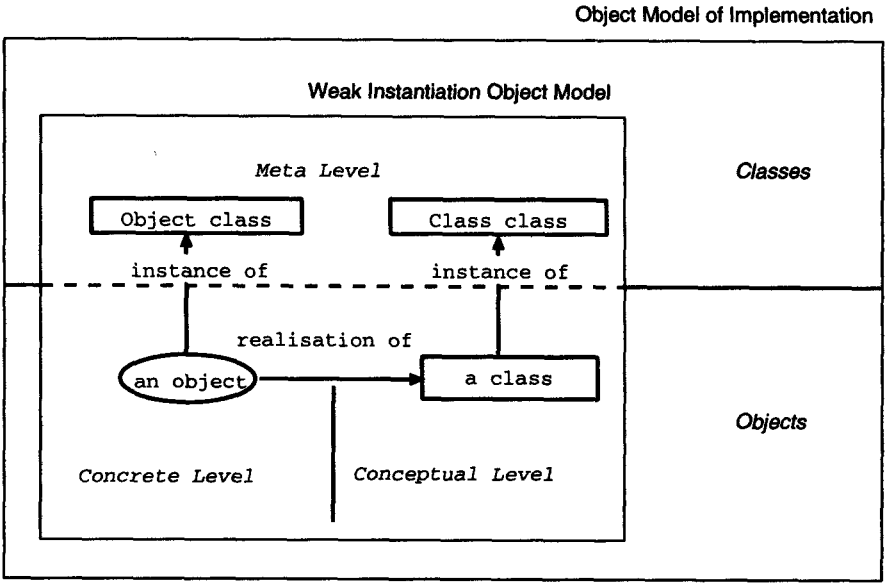

Figure 1: The realization mechanism

of those classes would have the same structure. The query only specifies some values for the structures.

In our model, little about object structures is known at the conceptual level (that of classes). Only the minimal structure of objects tied to a class (by the realization mechanism), i.e., the class structure, is known. Thus, queries have to deal with the additional structure of objects in order to fully make use of the expressiveness of the model. The query language must provide the user with features allowing her (him) to query both the class structure and the individual structure of objects.

3 Formal definition of the concepts

The main features of our model are types (cf. 3.1), classes (cf. 3.2) and objects (cf. 3.3). We give a formal definition for each of them and describe in 3.4 what a database is, according to our model. Finally, we exhibit some problems related to the flexible nature of the model.

3.1 Types

3.1.1 Definition

We assume the existence of some *terminal types*, corresponding, for example, to basic data types: integer (I), real (R), string(S), ... and a special type, called the "empty type" and noted $\top$.

We refer to $\mathcal{T}$ as the set of all object types, which are recursively defined as follows:

- terminal types and the empty type are types,
- if t is a type, then t^* is also a type, called a *set type*,
- if $t_1, ..., t_n$ are types, then $t_1 \times ... \times t_n$ is also a type, called a *tuple type*.

Examples:
Let R, S and I be terminal types.
$R \times S^* \times I$ and $(R \times I)^*$ are types.

3.1.2 Partial order on types

Assuming that its restriction to terminal types is known, we recursively define a partial order $\leq_\tau$ on $\mathcal{T}$ by:

- $\forall t \in \mathcal{T}$, $t \leq_\tau \top$; in other words, $\top$ is the greatest element of $(\mathcal{T}, \leq_\tau)$
- if $\exists i \in [1, n], t_i \leq_\tau t$ then $t_1 \times ... \times t_n \leq_\tau t$
- $t^* \leq_\tau t'^*$ iff $t \leq_\tau t'$
- if $t = t_1 \times ... \times t_n$ and $t' = t'_1 \times ... \times t'_{n'}$, then $t \leq_\tau t'$ iff:
  - $n' \leq n$
  - $t_i \leq_\tau t'_i$ for $1 \leq i \leq n'$

Example:
$R \times S^* \times I \leq_\tau R \times I$.
Indeed, $R \times S^* \times I = R \times (S^* \times I)$,
and we have: $R \leq_\tau R$ and $S^* \times I \leq_\tau I$.

3.2 Classes

3.2.1 Definition

Object classes, are members of the set $\mathcal{C}$; they are defined as follows:

- a *terminal class* is defined by the assignment of a class name c to a terminal type,
- a *simple class* is defined by a name c and a list $\langle a_1 : c_1, ..., a_n : c_n \rangle$, possibly empty and called the structure of c ; we note $c = \langle a_1 : c_1, ..., a_n : c_n \rangle$; a_i are attribute names while c_i are classes ; if the structure of a simple class c is empty, then c is said to be an "empty class",
- a *set class* c^* is defined for each simple class c.

We assume for the purpose of this study that the recursive definition of classes is cycle free. In other words, a class c can not refer to a class c' that is a specific class of c (in the sense of the inheritance graph defined in 3.2.3).

Examples:
$Integer, String, Real$, ..., terminal classes
$Date = \langle day : Integer, month : Integer, year : Integer \rangle$
$Person = \langle name : String, first\_name : String, age : Integer \rangle$
$Image = \langle photograph : Person, date : Date, location : String, characteristics : String^* \rangle$

3.2.2 Associated type for a class

To each class c, we associate an object type $[\![c]\!]$, recursively defined as follows:

- terminal classes:
  The associated type of a terminal class is the terminal type upon which the class is defined,
- simple classes:
  If $c = \langle a_1 : c_1, ..., a_n : c_n \rangle$, then $[\![c]\!] = [\![c_1]\!] \times ... \times [\![c_n]\!]$; if $n = 0$ then $[\![c]\!] = \top$,
- set classes:
  $[\![c^*]\!] = [\![c]\!]^*$.

Example:
$$\begin{aligned}[\![Image]\!] &= [\![Person]\!] \times [\![Date]\!] \times [\![String]\!] \times [\![String^*]\!] \\ &= ([\![String]\!] \times [\![String]\!] \times [\![Integer]\!]) \times \\ &\quad ([\![Integer]\!] \times [\![Integer]\!] \times [\![Integer]\!]) \times [\![String]\!] \times [\![String]\!]^* \\ &= (S \times S \times I) \times (I \times I \times I) \times S \times S^*\end{aligned}$$

3.2.3 Inheritance link between classes

The inheritance link between classes is a partial order, noted $\leq_\gamma$, such that:

- its restriction to terminal classes is deduced from the partial order on the associated terminal types: $c_1 \leq_\gamma c_2$ iff $[\![c_1]\!] \leq_\tau [\![c_2]\!]$,
- it is *user defined* for simple classes, in such a way that:
  if $c_1 = \langle a_{11} : c_{11}, ..., a_{1n_1} : c_{1n_1} \rangle \leq_\gamma c_2 = \langle a_{21} : c_{21}, ..., a_{2n_2} : c_{2n_2} \rangle$, then:
  - $n_2 \leq n_1$,
  - there exists an injection σ from $[1, n_2]$ to $[1, n_1]$, called "attributes renaming", such that $\forall i \in [1, n_2], c_{1\sigma(i)} \leq_\gamma c_{2i}$,

- $c_1^* \leq_\gamma c_2^*$ iff $c_1 \leq_\gamma c_2$.

If $c_1 \leq_\gamma c_2$, we will say that c_1 inherits from c_2.

Example:
If we have:
$Employee = \langle ssn : String, name : String, first\_name : String, age : Integer$
$salary : Real, addresses : Address^* \rangle$,
with:
$Address = \langle number : Integer, street : String, zip\_code : Integer, town : String \rangle$,
then:
$Employee \leq_\gamma Person$, given the renaming function:

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

because:
$c_{1\sigma(1)} = c_{12} = String \leq_\gamma String = c_{21}$,
$c_{1\sigma(2)} = c_{13} = String \leq_\gamma String = c_{22}$,
$c_{1\sigma(3)} = c_{14} = Integer \leq_\gamma Integer = c_{23}$.

The role of the σ function is to take into account the possible renamings and reorganisations of attributes across the inheritance link, particularily in the case of multiple inheritance. For the sake of simplicity, we assume from now on, without loss of generality, that this function equals the identity function.

3.3 Objects

3.3.1 Definition

An object $o = (i, s)$, is an element of $\mathcal{O}$ (the set of all objects), and is defined by a unique and exclusive identifier $i(o) \in \mathcal{I}$ (where $\mathcal{I}$ is the set of all object identifiers) and a structure $s(o)$.

We distinguish three kinds of objets according to the nature of their structure:

- *terminal objects*, where the structure is a value associated to a given terminal type;
- *simple objects*, where the structure is a list $\langle a_1 : o_1, ..., a_n : o_n \rangle$, possibly empty, where a_i are attribute names and o_i are component objects; a special attribute name is "X", it will be used whenever no semantics can be attached to the role that the component plays in the composition of the main object; if $s(o)$ is the empty list, o is called an "empty object" ;
- *set objects*, where the structure is a set $\{o_1, ..., o_p\}$, o_i being objects.

Examples: We give some examples of objects' structures.

$s(o_1) = \langle photograph : o_2,\ date : o_3,\ location : o_4,\ characteristics : o_5,$
$\quad X : o_6,\ X : o_7 \rangle$
$s(o_2) = \langle name : o_8, first\_name : o_9, age : o_{10} \rangle$
$s(o_3) = \langle day : o_{11}, month : o_{12}, year : o_{13} \rangle$
$s(o_4) = "Paris"$
$s(o_5) = \{o_{14}, o_{15}\}$
$s(o_6) = \langle name : o_{16}, first\_name : o_{17}, age : o_{18}, salary : o_{19}, addresses : o_{20} \rangle$
$s(o_7) = \langle name : o_{21}, first\_name : o_{22}, age : o_{23} \rangle$

$s(o_8) = "Martin"$	$s(o_{16}) = "Meunier"$
$s(o_9) = "Georges"$	$s(o_{17}) = "Jean"$
$s(o_{10}) = 65$	$s(o_{18}) = 50$
$s(o_{11}) = 14$	$s(o_{19}) = 7645,34$
$s(o_{12}) = 5$	$s(o_{20}) = \{o_{24}, o_{25}\}$
$s(o_{13}) = 1968$	$s(o_{21}) = "Duchemin"$
$s(o_{14}) = "portrait"$	$s(o_{22}) = "Emile"$
$s(o_{15}) = "black\ \&\ white"$	$s(o_{23}) = 23$

$s(o_{24}) = \langle number : o_{26}, street : o_{27}, zip\_code : o_{28}, town : o_{29} \rangle$
$s(o_{25}) = \langle number : o_{30}, street : o_{31}, zip\_code : o_{28}, town : o_{29} \rangle$
$s(o_{26}) = 17$
$s(o_{27}) = "Rue\ des\ lilas"$
$s(o_{28}) = 54000$
$s(o_{29}) = "Nancy"$
$s(o_{30}) = 19$
$s(o_{31}) = "Rue\ des\ roses"$

3.3.2 Associated type for an object

To each object o, we associate an object type $[\![o]\!]$, recursively defined as follows:

- Terminal objects:
  The type associated to a terminal object o is the terminal type to which is tied the value $s(o)$.
- Simple objects:
  If $s(o) = \langle a_1 : o_1, ..., a_n : o_n \rangle$, then $[\![o]\!] = [\![o_1]\!] \times ... \times [\![o_n]\!]$. If $n = 0$ then $[\![o]\!] = \top$.
- Set objects:
  If $s(o) = \{o_1, ..., o_p\}$, then:

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

  where *sup* means the upper bound in the sens of the partial order on object types ($\leq_\tau$).

Remark:
The latter definition is consistent, since $(\mathcal{T}, \leq_\tau)$ admits a greatest element: $\top$.

3.3.3 Realization link

The realization link is a binary relation, noted $\leftarrow$, defined on the set $\mathcal{O} \times \mathcal{C}$, in such a way that:

- if o is a terminal object and c is a terminal class of the same type, then $o \leftarrow c$,
- its restriction to the cartesian product of the set of simple objects by the set of simple classes, is *user defined*, in such a way that if $o \leftarrow c$ and $c = \langle a_1 : c_1, ..., a_n : c_n \rangle$, then:
  - $s(o) = \langle a_1 : o_1, ..., a_{n'} : o_{n'} \rangle$, with $n' \geq n$,
  - $\forall i \in [1, n], o_i \leftarrow c_i$,
- if o is a set object such that $s(o) = \{o_1, ..., o_p\}$, then $o \leftarrow c^*$ iff $\forall i \in [1, p], o_i \leftarrow c$,
- if $o \leftarrow c$ and $c \leq_\gamma c'$, then $o \leftarrow c'$.

If $o \leftarrow c$, we will say that o realizes c.

Assuming this definition, we can exhibit the fundamental following rule:

$$\boxed{\text{If } o \leftarrow c, \text{ then } [\![o]\!] \leq_\tau [\![c]\!]}$$

Proof:

See appendix A.

Examples:
From the previous object and class examples, we have:
$o_1 \leftarrow Image$, $o_3 \leftarrow Date$
$o_2 \leftarrow Person$, $o_6 \leftarrow Employee$, $o_6 \leftarrow Person$
$o_4 \leftarrow String$, $o_5 \leftarrow String^*$, $o_{10} \leftarrow Integer$, $o_{19} \leftarrow Real$
$o_{20} \leftarrow Address^*$, $o_{24} \leftarrow Address$, $o_{25} \leftarrow Address$

Remark:
In order to provide objects with richer structures than their classes', we separately defined the type and class paradigms. In contrast to the definition given in [HUL89], a class is not a set of objects in our model. It references a type that is more generic (in the sens of the partial order on types) than that of all its realizations (objects).

3.4 Databases

A database according to our model, is a triple $((\mathcal{C}, \leq_\gamma), \mathcal{O}, \leftarrow)$, where $(\mathcal{C}, \leq_\gamma)$ is a set of classes, latticed by a partial order, representing the inheritance link; $\mathcal{O}$ is

a set of objects and $\leftarrow$ is a binary relation on $\mathcal{O} \times \mathcal{C}$ representing the realization link.

One can notice that this definition involves both the notions of database schema (in the class set and its partial order) and database instances (in fact realizations, in the object set and the realization relation).

As mentioned earlier, the user defines the classes, the objects, the partial order on simple classes and the realization link on simple objects. The remainder of the database structure is deduced, according to the previous inheritance and realization rules.

3.5 Discussion

We shall point out some advantages of the realization link in 3.5.1, but also note some drawbacks in 3.5.2.

3.5.1 Advantages from relaxing the instantiation link

- The realization link frees objects from the imposed structure constraint induced by their class.
- In modelling some kinds of applications (particularly in the Information Retrieval field), the database designer is not always able to catch the exact structure of all classes, but only a part of it, representing a minimal stable and standard structure. The advantage of the realization link resides in allowing objects afterwards to have individual structures that might be further abstracted into the class, if the designer so decides.
- When exceptions are encountered among objects, i.e. when some objects have several different properties compared to their classes, the realization link prevents users from turning to some artificial design solutions to deal with such problems, like creating lots of sub-classes, one for each exception.
- The realization link is a good "design tool" when dealing with evolutionary applications, where objects' structures are often time-changing, in a way that can not be predicted in advance (non-stable life cycle). Such phenomena are often encountered in CAD-CAM or architectural applications. The realization link allows objects to evolve quite independently from their classes.

  Besides, it provides a basis for a possible "reverse engineering", which allows one to redefine class structures, once objects have stopped evolving. This is a classical problem in evolutionary databases.

3.5.2 Some drawbacks from the realization link

The main problems arising from the model we defined earlier appear when it comes to querying the database or specifying integrity constraints among it. In

classical structured models (relational, semantic, object-oriented), query formulation or integrity constraints specification are based on class structures, assuming that objects only value those structures with other objects or values.

In the remainder of this paper, we will focus on query formulation and processing, and defer integrity constraint investigation to a later study.

So, when querying a database for 50 years old employees, living in Nancy, the user is aware *a priori* of the existence of a class named *Employee*, representing employees and having an attribute giving their age (say *age*) and an other giving the town they live in (say *address*). The query can then be expressed by the set:

$$\{o \in Employee, o.age = 50 \wedge o.address.town = "Nancy"\}$$

As is the case in structured data models, this kind of query can also be specified in our model, since each object realizing the *Employee* class has an *age* attribute, and an *address* attribute, and each object realizing the class *Address* has a *town* attribute. In order to fully make use of the power of the realization link, queries have to deal also with the additional part of object structures that is not in their class structures. But this structure is not known at the conceptual level; for instance, the o_1 object, realizing the *Image* class uses, in its (additional individual) structure, the o_6 object, realizing the *Employee* class. This is not a mandatory (predictable) reference in the *Image* class.

The main question is how to make the user be able to specify such queries as images of 50 years old employees, living in Nancy?

In our opinion, an easy solution is to formulate queries the same way as objects: a part tied to class structures and an other tied to individual contents.

The first part comprises some criteria expressible at the conceptual level (i.e., involving properties from the target class). The second part is a set of sub-queries that have to be satisfied by component objects (mandatory and additional) of the main object satisfying the query. This strategy makes a good use of the expressive power of our model.

4 Queries

In this section, we define in 4.2 a model of queries suited to our data model. We first give some basic notions in 4.1.

4.1 Basic notions

For specifying queries, we need to define some preliminary notions, such as valid paths and path destinations for classes and objects.

A *path* is a list of attributes names, separated by points; e.g. *address.town* or *age.salary*.

This definition yields arbitrary paths; thus, we define valid paths for classes and objects, and along with it, we define valid path destinations, noted $Dest(c,p)$ (for a class c) and $Dest(o,p)$ (for an object o).

4.1.1 Valid paths and their destinations for a class

For obvious reasons, we define these notions only for simple and set classes (not for terminal classes).

- Simple Classes:
  - a path of length 1: $p = a$, is valid for a class $c = \langle a_1 : c_1, ..., a_m : c_m \rangle$ iff $\exists j \in [1,m], a_j = a$. Then we have: $Dest(c,p) = c_j$,
  - a path $p = a_1...a_n\ (n > 1)$, is valid for a class c, iff $p' = a_1...a_{n-1}$ is valid for c and $p'' = a_n$ is valid for $Dest(c,p')$. Then we have: $Dest(c,p) = Dest(Dest(c,p'),p'')$.
- Set classes:
  a path is valid for c^* iff it is valid for c ; and we have: $Dest(c^*,p) = (Dest(c,p))^*$.

Example:

$$\begin{aligned} Dest(Image, date.month) &= Dest(Dest(Image, date), month) \\ &= Dest(Date, month) \\ &= Integer. \end{aligned}$$

4.1.2 Valid paths and their destinations for an object

A path p is valid for an object o, iff $\exists c \in \mathcal{C}$, $o \leftarrow c$ and p is valid for c. Now, let us define the destination of a valid path for an object.

- Paths of length 1:
  - Simple objects: $Dest(o,a) = o.a$: the object corresponding to attribute a in $s(o)$.
  - Set objects: $Dest(o,a) = o'$ such that: $s(o') = \{Dest(o_i,a), o_i \in s(o)\}$.
    Remark:
    In this case, we create a new object not already existing in the object base. It is the only case of automatic object creation.
- Paths of length $n > 1$:
  $Dest(o, a_1...a_n) = Dest(Dest(o, a_1...a_{n-1}), a_n)$.

Example:

$$\begin{aligned} Dest(o_1, photograph.name) &= Dest(Dest(o_1, photograph), name) \\ &= Dest(o_2, name) \\ &= o_8. \end{aligned}$$

4.2 The query model

In 4.2.1, we define the general syntax of a query, and in 4.2.2 its semantics.

4.2.1 Query syntax

A query is a quadruple $q = (c, Cl, Q, p)$ where:

- c is a class, called *target* of q,
- $Cl = \{cl_1, ..., cl_n\}$ is a set of clauses, called *criteria* of q,
- $Q = \{q_1, ..., q_m\}$, is a set of queries, called *sub-queries* of q,
- p is a valid path for c, called *projection* of q.

Criteria, sub-queries and/or projection may be empty.

A clause is a disjunction of literals: $cl_i = l_{i1} \vee ... \vee l_{iq_i}$.

A literal l_{ij} takes one of the following forms:

- $p = p'$ or $p \neq p'$, where p and p' are valid paths for c, such that $Dest(c,p) \leq_\gamma Dest(c,p')$ or $Dest(c,p') \leq_\gamma Dest(c,p)$,
- $p = o_0$ or $p \neq o_0$, where p is a valid path for c and o_0 is a terminal object, such that $o_0 \leftarrow Dest(c,p)$,
- $p \in p'$ or $p \notin p'$, where p and p' are valid paths for c, such that $Dest(c,p') \leq_\gamma (Dest(c,p))^*$ or $(Dest(c,p))^* \leq_\gamma Dest(c,p')$,
- $o_0 \in p$ or $o_0 \notin p$, where p is a valid path for c and o_0 is a terminal object, such that $Dest(c,p) = c'^*$ and $o_0 \leftarrow c'$,
- $p \subseteq p'$ or $p \not\subseteq p'$, where p and p' are valid paths for c, such that $Dest(c,p) = c'^*$ and $Dest(c,p') = c''^*$, having $c' \leq_\gamma c''$ or $c'' \leq_\gamma c'$,
- $o_0 \subseteq p$ or $o_0 \not\subseteq p$, where p is a valid path for c such that $Dest(c,p) = c'^*$, and o_0 is a set object, the structure of which is a set of terminal objects, and such that $o_0 \leftarrow c'^*$.

If $c = c'^*$, then we define: $q_* = (c', Cl, Q, p)$.

Before giving the formal semantics of such queries, we outline their informal meaning: query q looks for objects realizing class c, satisfying all clause members of Cl and referencing, for each query member of Q, at least one object resulting from this query. The result of q is the set of path p destinations for all objects satisfying those three conditions.

In consequence, as was the case for classes, the recursive definition of queries must be cycle free.

Examples:
The query $q_1 = (c_1, Cl_1, Q_1, p_1)$ looks for 50 years old employees, living in Nancy.

$c_1 = Employee$
$Cl_1 = \{age = o_{01}, o_{02} \in addresses.town\}$, with $s(o_{01}) = 50$ and $s(o_{02}) =$ "$Nancy$"
$Q_1 = \emptyset$
p_1 is empty.

The query $q_2 = (c_2, Cl_2, Q_2, p_2)$ looks for images of such employees.
$c_2 = Image$
$Cl_2 = \emptyset$
$Q_2 = \{q_1\}$
p_2 is empty.

The realization link is taken into account in our query specification language, within the set Q, that specifies criteria on all referenced objects of the main object. Conceptual knowledge on objects, coming from their class structure is used in the set Cl.

Thus, a query uses more structures than only those of the target class.

4.2.2 Query semantics

We define the formal semantics of a query $q = (c, Cl, Q, p)$ as being an application μ_q, defined on the powerset of $\mathcal{O}$ as follows:

$$\begin{array}{lll} \mu_q : \mathcal{P}(\mathcal{O}) & \longrightarrow & \mathcal{P}(\mathcal{O}) \\ O & \longmapsto & \mu_q(O) = \{Dest(o,p), o \in O \wedge o \text{ satisfies } q\}. \end{array}$$

The satisfaction of a query by an object is defined as follows:

- A simple object o such that $s(o) = \langle a_1 : o_1, ..., a_n : o_n \rangle$ satisfies a query $q = (c, Cl, Q, p)$, iff the three following conditions are satisfied:
  - $o \leftarrow c$.
  - The evaluation of all clause members of Cl within o leads to True. This evaluation is done in an intuitive way, according to the kind of each literal in the clause. Within a literal, a path p is evaluated to $Dest(o,p)$.
  - $\forall q' = (c', Cl', Q', p') \in Q, \exists i \in [1,n], \exists o' \in \mathcal{O}$, o' satisfies $q' \wedge Dest(o', p') = o_i$.
- A set object o satisfies a query q, iff $\forall o' \in s(o), o'$ satisfies q_*.

Examples:
Object o_6 satisfies query q_1 and as a consequence, object o_1 satisfies query q_2.

5 Implementation

To validate our approach, we developped a prototype within the Smalltalk-80 object oriented programming environment [GR83], on a SPARC station.

Each of the model concepts (objects, classes, attributes, queries, ...) has been represented by a class in Smalltalk, that is a *meta-class* in our model. Accurate methods are attached to each class. Those methods implement the "philosophy" of our model, by describing the behaviour of each concept.

The object-oriented data model of smalltalk is then used as a *meta-model* for our model.

Each feature of our model (an object, a class, a query, ...) is represented as an object in Smalltalk, that is a *meta-object* in our model. It is an instance of the related meta-class (cf. figure 1).

To the concepts presented throughout the paper, we added two important features:

- the distinction between objects and values; that is terminal objects are treated as values (no identifiers);
- the capability to manage semantic relationships between objects; e.g. a person is *on* a boat, a house is *in the back* of an image.

Those features have not been specified in this paper, in order to define the realization mechanism (that is the main originality of our model), in a simple and clear way. But they are also integrated into the mechanism.

The prototype allows one to:

- create, modify or delete classes and relationships, within a *conceptual level browser* (cf. figure 2),
- create, modify or delete objects, within a *concrete level browser* that has the same look as the precedent,
- create, modify or delete queries, within a *query browser* that also has the same look as the conceptual browser.

In the actual version, queries are evaluated among the object base, according to the query semantics presented in 4.2.2, extended to capture relationships.

We have experimented our approach on an image database containing images of Paris in the 1900's, that has also served as an example database for the RIVAGE system, also developped in our team [HCK90]. Figure 3 gives the results of a query looking for images containing two aquatic vehicules, side by side.

When queries address images, the resulting objects (images) are visualized using an image viewer, that has been developped for the RIVAGE system [CHA93] (figure 3).

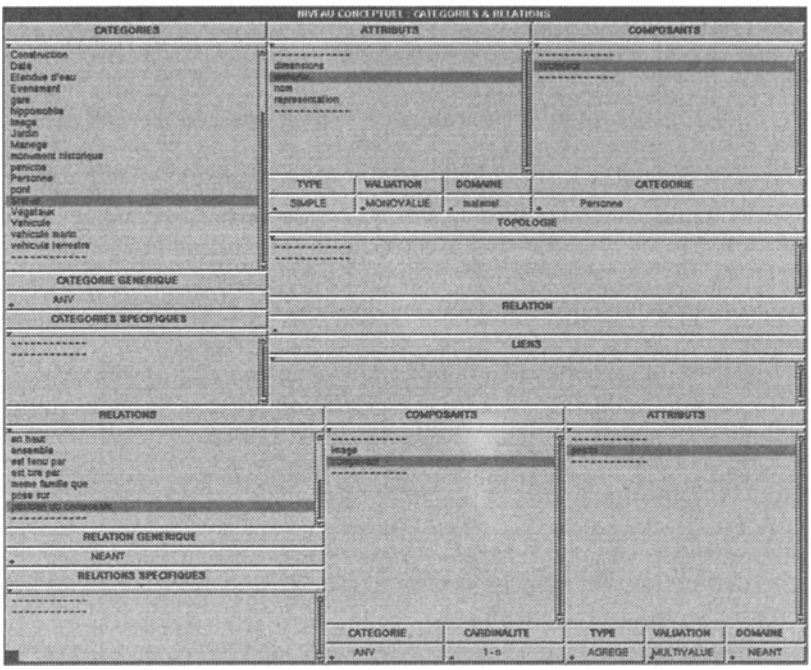

Figure 2: Conceptual browser

This experimentation clearly showed that queries on image databases can be extended to capture complex objects and semantic links in image contents, rather than the simple use of keywords.

Conclusion

We have presented a data model intended to model and query complex objects having a strong individual structure, as those usually encountered in Content-Based Information Retrieval (images, texts, ...).

One particularity of the model is to provide objects related to a given class with an extra individual structure, in addition to that of the class (common to all its related objects).

The classical instantiation link between an object and its class has been relaxed and rebaptized "realization link", in order to clearly outline the gained freedom for objects, with regard to their classes.

We have also presented a query specification language suited to our data model and showed that all the model concepts can be represented as meta-classes in a class-based object-oriented model.

Figure 3: Image browser

The realization link shows itself particularily useful in the Content-Based Information Retrieval field, as has been stated in some of our previous research [MLH94, LAH95].

The perspectives envisaged for this work are to explore integrity constraints specification and maintainance, according to our model features. In classical databases, integrity constraint specification, like query specification, uses some conceptual knowledge about class structures, to specify criteria among objects.

Since in our model, objects have richer structures than their classes, it would be interesting to use the additional individual object structures in constraint specification, as we have done for queries.

References

[AB88] S. ABITEBOUL and C. BEERI. On the power of languages for the manipulation of complex objects. Technical Report RR-0846, INRIA, 1988.

[CAT94] R.G.G. CATTELL, editor. *The Object Database Standard: ODMG-93.* Morgan Kaufmann, 1994.

[CHA93] D. CHAFFIOL. Rivage et imageur. Rapport de stage ESIAL 2, Univ. Nancy I, 1993.

[COD70] E.F. CODD. A relational data model for large shared data banks. *Communications of the ACM*, 13(6):377–387, 1970.

[COD79] E.F. CODD. Extending the relational model to capture more meaning. *ACM Transactions on Database Systems*, 4(4):397–434, 1979.

[GPVG89] M. GYSSENS, J. PAREDAENS, and D. VAN GUCHT. A grammar-based approach towards unifying hierarchical data models (extended abstract). In *ACM-SIGMOD'89*, pages 263–272, 1989.

[GR83] A. GOLDBERG and D. ROBSON. *Smalltalk80: The Language and its Implementation.* Addison-Wesley, 1983.

[HCK90] G. HALIN, M. CREHANGE, and P. KEREKES. Machine-learning and vectorial matching for an image retrieval model : Exprim and the rivage system. In *ACM-SIGIR'90*, pages 99–114, 1990.

[HK87] R. HULL and R. KING. Semantic database modelling: Survey, applications and research issues. *ACM Computing Surveys*, 19(3):201–260, 1987.

[HLMM92] J.N.D. HIBLER, C.H.C. LEUNG, K.L. MANNOCK, and N. MWARA. A system for content-based storage and retrieval in an image database. In *Image Storage and Retrieval Systems, IS&T / SPIE Symposium on Electronic Imaging - Science and technology, SPIE'92*, pages 80–92, San Jose, California, 1992.

[HUL89] R. HULL. Four views of complex objects: A sophisticate's introduction. In *Nested relations and complex objects in databases*, pages 87–116. Springer-Verlag, Lecture Notes in Computer Science, No 361, 1989.

[KIM90] W. KIM. Object-oriented databases: definitions and research directions. *IEEE Transactions on Knowledge and Data Engineering*, 2(3):327–341, 1990.

[KKL91] D.A. KEIM, K.C. KIM, and V. LUM. A friendly and intelligent approach to data retrieval in a multimedia dbms. In *Database and Expert Systems Applications, DEXA'91*, pages 102–111, Berlin, Germany, 1991.

[LAH95] Y. LAHLOU. Modeling complex objects in content-based image retrieval. In *Storage and Retrieval for Image and Video Databases III, IS&T / SPIE Symposium on Electronic Imaging - Science and technology, SPIE'95*, pages 104–115, San Jose, California, 1995.

[MLH94] N. MOUADDIB, Y. LAHLOU, and G. HALIN. Emir: A content-based data model. In *Maghrebian Conference on Software Engineering and Artificial Intelligence, MCSEAI'94*, pages 503–512, Rabat, Morocco, 1994.

[RKS88] M.A. ROTH, H.F. KORTH, and A. SILBERSCHATZ. Extended algebra and calculus for nested relational databases. *ACM Transactions on Database Systems*, 13(4):389–417, 1988.

[RS92] F. RABITTI and P. SAVINO. Automatic image indexation to support content-based retrieval. *Information Processing & Management*, 28(5):547–565, 1992.

[SCH90] C. SCHWARZ. Content-based text handling. *Information Processing & Management*, 26(2):219–226, 1990.

[WLK87] D. WOELK, W. LUTHER, and W. KIM. Multimedia applications and database requirements. In *IEEE Office Automation Symposium*, pages 180–189, 1987.

A Proof of the realization rule

Let $o \in \mathcal{O}$ and $c \in \mathcal{C}$, such that $o \leftarrow c$. We have to prove that $[\![o]\!] \leq_\tau [\![c]\!]$.

- If o is a terminal object, then c is a terminal class of the same type, and we have indeed $[\![o]\!] \leq_\tau [\![c]\!]$.
- If o is a simple object, then $s(o) = \langle a_1 : o_1, ..., a_{n'} : o_{n'} \rangle$ and $c = \langle a_1 : c_1, ..., a_n : c_n \rangle$, having:
  - $n' \geq n$,
  - $\forall i \in [1, n], o_i \leftarrow c_i$,

  The proof is done in an inductive fashion, on the component objects of $s(o)$. Assume, as induction hypothesis, that $\forall i \in [1, n], [\![o_i]\!] \leq_\tau [\![c_i]\!]$; we then have:

  $[\![o]\!](= [\![o_1]\!] \times ... \times [\![o_{n'}]\!]) \leq_\tau [\![c]\!](= [\![c_1]\!] \times ... \times [\![c_n]\!])$, coming from the definition of the partial order on tuple types.
- If o is a set object, then $s(o) = \{o_1, ..., o_p\}$ and $c = c'^*$. From the definition of the realization link, we get: $\forall i \in [1, p], o_i \leftarrow c'$.

  The proof is also done in an inductive fashion, on the member objects of $s(o)$. Assuming, as induction hypothesis, that $\forall i \in [1, p], [\![o_i]\!] \leq_\tau [\![c']\!]$, we see that $[\![c']\!]$ is greater than all the $[\![o_i]\!]$'s, in the sense of the partial order on object types; and so we induce:

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

  which yields:

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

□