International Journal of Image and Graphics


# VIEWS OR POINTS OF VIEW ON IMAGES

VINCENT ORIA
*College of Computing Sciences, New Jersey Institute of Technology*
*Newark, NJ, 07102-1982, USA*
*E-mail: Vincent.Oria@njit.edu*

M. TAMER ÖZSU
*School of Computer Science, University of Waterloo*
*Waterloo, Ontario, Canada N2L 3G1*
*E-mail: tozsu@uwaterloo.ca*

Received (received date)
Revised (revised date)

Images like other multimedia data need to be described as it is difficult to grasp their semantics from the raw data. With the emergence of standards like MPEG-7, multimedia data will be more and more produced together some semantic descriptors. But a description of a multimedia data is just an interpretation, a point of view on the data and different interpretations can exist for the same multimedia data. In this paper we explore the use of view techniques to defines and manage different points of view on images. Views have been widely used in relational database management systems to extend modeling capabilities, and to provide logical data independence. Since our image model is defined on an object-oriented model, we first propose a powerful object-oriented mechanism based on the distinction between class and type. The object view is used in the image view definition. The image view mechanism exploits the separation of the physical representation in an image of a real world object from the real object itself to allow different interpretations of an image region. Finally we discuss the implementation of the image view mechanisms on existing object models.

*Keywords*: Image Database; Image Semantics; Views on Objects; Views on Images.

## 1. Introduction

Views are used in relational database management systems to extend modeling capabilities and to provide data independence. Relational views can be seen as formulae defining virtual relations that are not produced until the formulae are applied to real relations (view materialization is an implementation/optimization technique). View mechanisms are useful in other newly emerging application areas of database technology. In this paper, we discuss a view mechanism for one of those areas, image databases. This work is conducted within the context of the DISIMA (DIStributed Image database MAnagement system)[1]. Since DISIMA uses object-oriented technology, we deal with object-oriented views.

Due to the volume and the complexity of image data, image databases are commonly built on top of object or object-relational database systems. Image databases, in particular, can benefit from a view mechanism. Specifically, an image can have several interpretations that a view mechanism can model. The DISIMA system[2] defines a model that is capable of handling an image and all the meta-data associated with it, including syntactic characterization (shape, color and texture) of *salient objects* (region of interest) contained in the image. The level at which the syntactic features are organized and stored is called the physical salient object level. Each physical salient object can then be given a meaning at the logical salient object level. In general, salient object detection and annotation is a semi-automatic or a manual process.

Given the fact that we can manually or automatically extract meta-data information from images, how do we organize this information so that an image can be interpreted with regard to a context? That is, if the context of an image changes, the understanding of the image may change as well. Let us take a picture of the Smith family as an example. The family has four members: Mr. and Mrs. Smith, their young son John, and their daughter Jane. If John is describing this picture to his friends, he will say: "This is me, these are my mother and father and over there is my sister." The description of Mrs. Smith for the same picture will be: "This is me, my husband, my young son John, and my daughter Jane." The example illustrates different descriptions of the same image. They are all valid within the right context. As another example, consider an electronic commerce system with a catalog containing photographs of people modeling clothes and shoes. From the customer's point of view, interesting objects in this catalog are shirts, shorts, dresses, etc. But the company may want to keep track of the models as well as clothes and shoes. Assume that the models come from different modeling agencies. Each of the agencies may be interested in finding only pictures in which their models appear. All these users of the same database (i.e., the catalog) have different interpretations of the content of the same set of images. In the first example, the same regions of interest in the image are given different semantics depending on the interpretation context and the second example the objects of interest are different from one context to another.

Defining an image content with regard to a context helps capture more semantics, enhances image modeling capabilities, and allows the sharing of images among several user groups with different points of view on the image. Hence, the user should not notice any difference in the way derived images and base images are handled. However, in existing object view mechanisms[3,4,5,6], derived objects are not first class citizens. To that end, we first defined an object view mechanism on top of which we built the image views.

An earlier version[7] of this work was published in the IFIP DS-8 proceedings. Section 2 discusses the existing image metadata and models. Section 3 describes the image view mechanism and the object view on top of which it is built, Section 4 presents the image view definition language and discusses the implementation

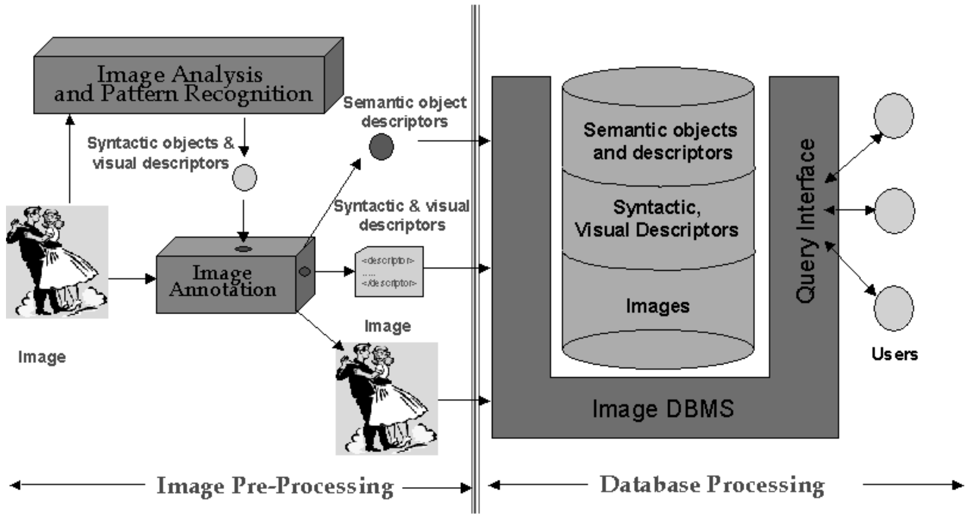



Figure 1: Image Preprocessing and Database.

issues. Section 5 discusses object and image view issues, and Section 6 concludes.

## 2. Image, Annotation, Metadata and Semantics

A common problem in multimedia data is their interpretation and understanding. Raw multimedia data are sequences of bytes that cannot be directly used to answer queries; they have to be preprocessed. The preprocessing represents the multimedia data in a more digest form for a computer. The data obtained from the preprocessing are metadata (data about data) for the raw multimedia data. We are more interested in the content metadata that can be divided into two categories: syntactic metadata and semantic metadata. For an image, the syntactic metadata can be feature vectors representing visual features such as color, texture, and geometric shapes found in the image. The semantic metadata describes the image content in terms of real-world objects, actions and feelings.

Hence an image database can be seen as composed of two distinct parts (Figure 1): The database component in charge of data organization and querying, and the image preprocessing component that extracts features from the images, detects and, sometimes, recognizes objects in the images. The preprocessing is, most of the time, semi-automatic as automatic recognition of objects in an image of arbitrary scene is still a research problem in Computer Vision[8].

### 2.1. *Annotation*

Well-known image processing techniques[9] can be used to segment an image into homogeneous regions (syntactic objects) and extract visual features (color, texture and shape). Identifying the objects and interpreting the image is more challenging and, in most cases, requires human intervention. Annotation is the process that provides interpretations (semantics metadata) to an image. In the IKONA system[10] a relatively small database is manually annotated by associated keywords to the images. The system then finds correlations between the keywords and the image visual features in order to propagate the keywords and annotate a larger image database. The same approach can be used to associate keywords with regions of an image. Semi-automatic annotation techniques based on the WorldNet electronic dictionary[11] have also been proposed[12]. Semantic classes defined as semantic labels and visual descriptors are used to annotate the images. In the DISIMA system[1], the link between physical salient objects (regions of an image) and the corresponding logical salient objects (meanings) are done manually.

## 2.2. *Metadata for Images*

Since it is difficult to directly use raw image data in queries, the acquisition, organization and maintenance of the metadata for images are becoming a key issues in image archiving. Several organizations are defining standard metadata structures for images. Among these standards are the Dublin Core metadata, MPEG-7 and RDF.

### 2.2.1. *The Dublin Core Metadata Initiative*

The Dublin Core (DC) is an international initiative whose goal is to define metadata for digital resources. The initial metadata format published in 1995[13] was extended in 1996 to support images. The extension consisted in the addition of two more elements to the thirteen originally defined and the adaptation of the descriptions of some existing elements. The resulting simple Dublin Core[14] is composed of the following fifteen elements: Title [Name given to the resource by the 'Creator' or Publisher], Author or Creator [person(s) or organization(s) primarily responsible for the intellectual content of the resource], Subject and Keywords [the topic of the resource (keywords, index terms or subject classification)], Description [textual description of the resource], Publisher [the entity responsible for making the resource available], Other Contributor [person(s) or organization(s)not included under 'Creator' who have made significant (but secondary)intellectual contribution to the resource], Date [date the resource was made available in its present form], Resource type [The category of the resource, e.g. 'image'], Format [the data representation of the resource, e.g. 'text/html', 'JPEG'], Resource Identifier [string or number used to uniquely identify the resource, e.g. 'URI', 'URL'], Source [Information on the work from which the resource is derived], Language [languages of the intellectual content of the resource], Relation [relationship of the resource with other resources], Coverage [spatial locations or temporal duration characteristic of the resource], Rights Management [a link to a rights management statement].

### 2.2.2. *The MPEG-7 Metadata Initiative*

MPEG-7, formally known as Multimedia Content Description Interface mainly concentrates on the description of multimedia content. The objective of MPEG-7 is to provide a rich set of organized tools to describe multimedia content and represent information about the content at different levels. MPEG-7 is an ISO standard and addresses image, text, video, audio, speech, and graphics individually along with their combinations.

The core elements of MPEG-7 standards are the Descriptors (D) to define the syntax and semantics of each multimedia feature, Description schemes (DS) to specify the structure and semantics of the relationships between their components. The Description Definition Language (DDL) is used to define the syntax of MPEG-7 and structure of descriptors. It allows creation of new description schemes and descriptors. The DDL[15] consists of (1) XML Schema structural components, (2) XML Scheme data types, and (3) MPEG-7 specific extensions.

### 2.2.3. *The Resource Description Framework (RDF)*

RDF[16] is defined by the World Wide Web Consortium (W3C) in order to provide meaning to resources available on the Web (among which images). Meaning is represented in RDF by sets of triples, each triple containing a subject, a verb and an object of an elementary sentence. In RDF, a document makes assertions that particular objects have properties, with certain values. This structure turns out to be a natural way to describe the cast majority of the data processed by machines. Subject and Object are each identified by a Universal Resource Identifier (URI). The simplest form of a URI is the URL. The verbs are also defined by the URI's. A property is a specific aspect, attribute or relationship, used to describe a resource. Each property has a specific meaning, defines its possible values, the types of resources it can describe, and the relationships with other properties.

## 2.3. *Image Database Models*

The main objective of the standards or proprietary metadata structures is to provide audio-visual and textual content descriptors for individual images. In the presence of a large number of images, it is crucial to find a suitable representation for images, their metadata and the operations to be applied to them in order to answer users queries. This level corresponds to the conceptual level in the ANSI relational database architecture[17] where algebraic optimizations and algorithm selections are performed. Physical optimizations at the physical level (data files and indexes) consist of selecting the indexes and access methods to be used in the query processing. The image models are built on top of existing database models, mainly object-relational and object-oriented.

### 2.3.1. *An Example of Object-Relational Image Data Model*

In [18], an image is stored in a table $T(h : Integer, x_1 : X_1, \ldots, x_n : X_n)$ where $h$ is the image identifier and $x_i$ is an image feature attribute of domain (or type) $X_i$ (note that classical attributes can be added to this minimal schema). The tuple corresponding the image $k$ is indicated by $T[k]$. Each tuple is assigned a score ($\zeta$) which is a real such that $T[k].\zeta$ is a distance between the image $k$ and the current query image. The value of $\zeta$ is assigned by a scoring operator $\Sigma_T(s)$ given a scoring function $s$: $(\Sigma_T(s))[k].\zeta = s(T[k].x_1, \ldots, T[k].x_n)$.

Since many image queries are based on distance measures, a set of distance functions ($d : X \rightarrow X \rightarrow [0, 1]$) are defined for each feature type $X$. Given an element $x : X$ and a distance function $d$ defined on $X$, the scoring function $s$ assigns $d(x)$, a distance from $x$ to every element of $X$. In addition, a set of score combination operators $\Diamond : [0, 1] \times [0, 1] \rightarrow [0, 1]$ are defined.

New selection and join operators on the image table augmented with the scores are defined to allow selection of the $n$ images with lowest scores, the images whose scores are less then a given score value $\rho$ and the images from a table $T_1$ that match images from a Table $T_2$ based on score combination functions.

### 2.3.2. *The DISIMA Model: An Object-Oriented Approach*

The DISIMA model is presented in more detail as it is the base of the view mechanism. In the DISIMA model, as shown in Figure 2, an image is composed of physical salient objects (regions of the image) whose semantics are given by logical salient objects that represent real world objects. The DISIMA model introduces three new types: *Image*, *Physical Salient Objects*, *Logical Salient Objects* and operators to manipulate them.

From the *Image* type, the user can define new types to classify the images. Figure 3 depicts a type hierarchy for an electronic commerce application that represents the catalogs as classes. The general `T_Catalog` type is derived from the root type `T_Image`, the root image type provided by DISIMA. The type `T_Catalog` is specialized by two types: `T_ClothingCatalog`, and `T_ShoesCatalog`.

An example of salient object hierarchy, corresponding to the image hierarchy defined in Figure 3, is given in Figure 4.

A *physical salient object (PSO)* is a region of an image, a geometric object (without any semantics) in a space (defined by an image) with properties like shape, color, and texture. A *logical salient object (LSO)* is the interpretation of a region. It is a meaningful object that is used to give semantics to a physical salient object.

If we denote by $\mathcal{L}$ the set of all logical salient objects and $\mathcal{P}$ the set of all physical salient objects, the content of an image $i$ is defined by a pair $Cont(i) =< \mathcal{P}^i, s >$ where:

- $\mathcal{P}^i \subseteq \mathcal{P}$ is a set of physical salient objects,
- $s : \mathcal{P}^i \longrightarrow \mathcal{L}$ maps each physical salient object to a logical salient object.

An *image* $i$ is defined by a triple $< Rep(i), Cont(i), Desc(i) >$ where:

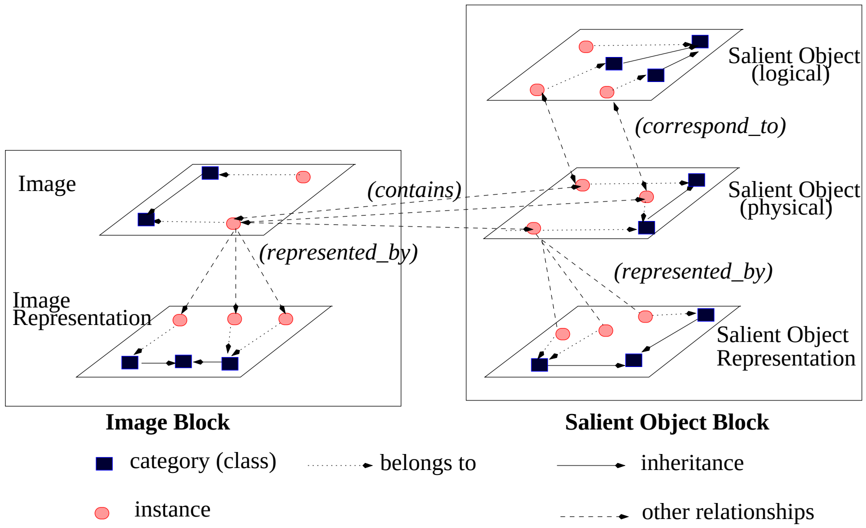



Figure 2: The DISIMA Model Overview.

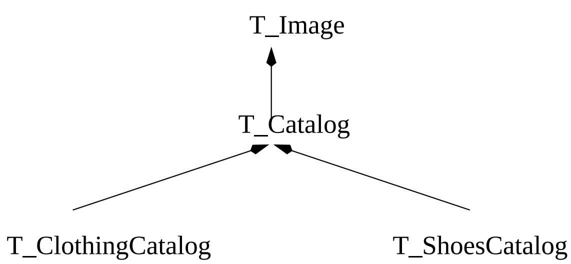



Figure 3: An Example of an Image Hierarchy.

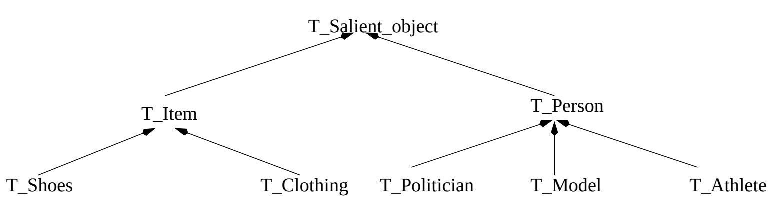



Figure 4: An Example of Logical Salient Object Hierarchy.

- $Rep(i)$ is a set of representations of the raw image in a format such as GIF, JPEG, etc;
- $Cont(i)$ is the content of the image i;
- $Desc(i)$ is a set of descriptive alpha-numeric data associated with $i$.

Color and texture characterizing the whole image are part of $Desc(i)$.

The new operators introduced for objects of the new types are: *contains* or *semantic join* (to check wether a salient object is found in an image), and the *similarity join* that is used to match two images or two salient objects with respect to a predefined similarity metric on some low-level features (color, texture, shape, etc.). In addition, DISIMA supports the *spatial join*[19] on physical salient objects.

## 3. Defining Image Views in DISIMA

The mechanism of image views in DISIMA exploits the distinction between an image and its content represented as physical salient objects, and the distinction between a physical salient object and its meaning represented as a logical salient object. The idea is (1) to be able to define different sets of regions of interest (physical salient objects) from the same image and (2) to be able to assign different logical salient objects to a physical salient object depending on the context.

In the initial DISIMA model, a physical salient object is associated with one and only one logical salient object but to a logical salient object can correspond several physical salient objects. This allows only one context of interpretation for an image.

**Definition 1** *An* image interpretation context *is defined by the minimal logical salient object sub-schema and the logical salient object instances that give meanings to the physical salient objects contained in the image.*

The first extension to the model is to change the relationships between PSOs and LSOs into a many-to-many relationships. This means that a physical salient object can have different logical salient objects each of which can belong to a different interpretation context. A new interpretation can be derived from an existing one by selecting, aggregating or redefining descriptor elements. These derived interpretation are based on object view mechanism. The second extension is to allow an image to have different contents in terms of the physical salient objects. These two extensions form the image view mechanism based on an object view mechanism that built on an object model[20] we defined earlier.

### 3.1. *The Object Model*

We separate the definition of object characteristics (a *type*) from the mechanism for maintaining instances of a particular type (a *class*) for several well known reasons[20]. A *type* defines behaviors (or properties) and encapsulates hidden behavioral implementations (including state) for objects created using the type as template. We use the term behaviors (or properties) to include both public interface functions (methods) and public state (public instance variables). The behaviors

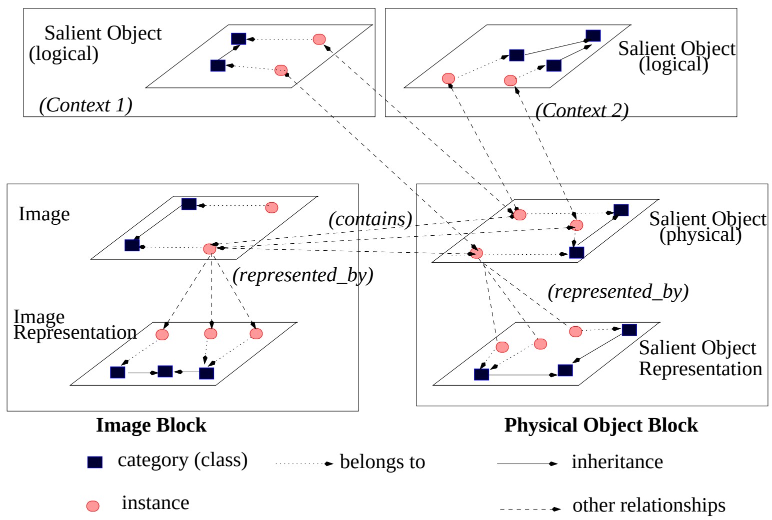



Figure 5: Image Views in DISIMA.

defined by a type describe the *interface* for the objects of that class. A *class* ties together the notion of *type* and *object instances*. The entire group of objects of a particular type, including its subtypes is known as the *extent* of the type and is managed by its class. We refer to this as *deep extent* and introduce *shallow extent* to refer only to those objects created directly from the given type without consideration of its subtypes. For consistency reasons all the type names used in this paper start with $T\_$. The object model that we use in this study differentiates types from classes. Let $\mathcal{C}$ be the set of class names. If $C$ is a class name, $T(C)$ gives the type of $C$ and $\Gamma(C)$ denotes the extent of the class $C$. We denote by $\mathcal{T}$, the graph representing the type hierarchy. The main concepts for the model can be summarized as follows:

- The set of types $T$ with a partial order $\leq_t$ (subtyping). A type `T_X` can define a set of behaviors ($\{B\_alpha_i\}$). It can also bind behaviors to functions.
- The set of classes $C$. Each class `X` has an associated type `T_`(`X`) = `T_X` (type of the objects in the class) and manages a set of objects of type `T_X`: [$\Gamma$(`X`)]
- The set of behaviors $B$.
- The set of functions $F$. A function is defined by a signature and a body.
- The set of *objects* $O$. Each object $o$ belongs to a single class [`X` $= C(o)$] and each object knows its type [`T_X` $= T(o)$].

### 3.1.1. *Derived Classes*

We consider two types of derived classes: simple derived classes (derived from a single class called *the parent class*) and composed derived classes (derived from two or more parent classes). We will use the term *base class* to refer to a non-derived class. In the same way, a base object refers to an object of a base class. The derivation relationship is different from the specialization/generalization one in the sense that the objects and properties introduced are obtained from data previously stored in the database.

**Simple Derived Class:** A simple derived class is a virtual class derived from a single parent class.

**Definition 2** *A* **derived class** $C_d$ *is defined by* $(C, \Phi, \Psi)$ *where:*

- $C$ *is the parent class*
- $\Phi$*, the filter, is a formula that selects the valid objects from* $C$ *for the extent of* $C_d$
- $\Psi$*, the interface function, defines the type of* $C_d$ *by combining the functions* A: Augment *and* H: Hide *such that* $\Psi = A \circ H$*, where* $A$ *maps a set of objects of a particular type to a set of corresponding objects in a type with some additional properties. Similarly* $H$ *hides some properties.*
- $\Gamma(C_d) = \Psi(\Phi(\Gamma(C)))$

As defined, $\Psi$, $A$ and $H$ have to be applied to sets of objects of a certain type to return sets of objects of another type. To avoid introducing news terms, we will extend their applications to types.

If $\alpha, \beta, \gamma, \delta$ are properties defined in `T_C`, $H(\texttt{T\_C}, \{\alpha, \beta\})$ will create a new type (let us call it `T_Restricted_C`) in which only the properties $\gamma, \delta$ are defined. Hence `T_Restricted_C` is a supertype of `T_C`.

$A(\texttt{T\_C}, \{(\mu : f_1), (\nu : f_2)\})$ will create a type (`T_Augmented_C`) with the additional properties $\mu$ and $\nu$, where $f_1$ and $f_2$ are functions that implement them. `T_Augmented_C` is a subtype of `T_C`.

$A(H(\texttt{T\_C}, \{\alpha, \beta\}), \{(\mu : f_1), (\nu : f_2)\})$ defines the type $\texttt{T\_C}_\texttt{d}$ for a class $C_d$ derived from a class $C$ with the properties $\alpha, \beta$ of `T_C` hidden and $\mu, \nu$ as new properties.

In general, the type $\texttt{T\_C}_\texttt{d}$ of a class $C_d$ derived from the class $C$, is a sibling of $T(C)$. However, if no properties are hidden, $T(C_d) \leq_t T(C)$ ($\leq_t$ stands for the subtyping relationship). Alternatively, if no properties are added, $T(C_d) \geq_t T(C)$ ($\geq_t$ stands for the supertyping relationship). The notion of sibling generalizes the notion of subtyping and supertyping. The most general case where some properties are removed and new ones are added is illustrated by Figure 6. In this example, we assume that the following properties are defined for the different types:

- `T_Person`(SIN: int, LastName: string, FirstName: string, Sex: char, DateOfBirth: date)
- `T_Restricted_Person`(SIN: int, LastName: string, FirstName: string, Sex: char)
- `T_Augmented_Restricted_Person`(SIN: int, LastName: string, FirstName: string, Sex: char, Age: int)

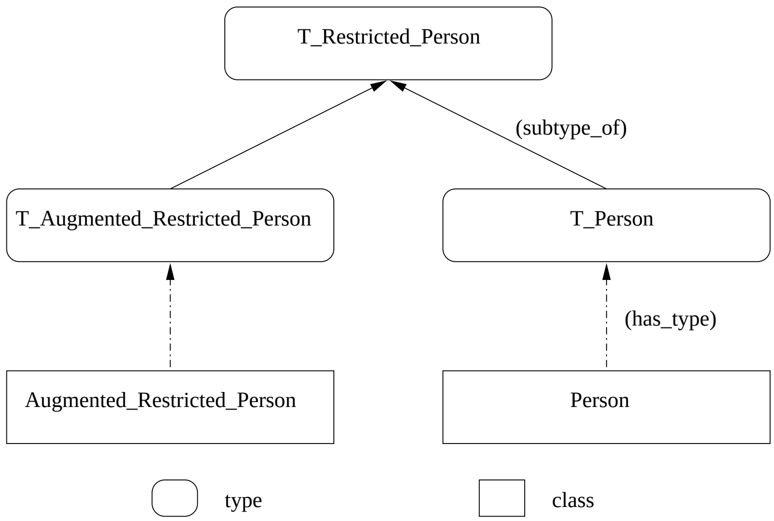



Figure 6: An Example of a Derived Class and its Type

In Figure 6, the extent of `Augmented_Restricted_Person` is a subset of the extent of `Person` with a different interface defined by the type `T_Augmented_` Restricted_Person.

**Composed Derived Class:** Assume that the type `T_Person` has two subtypes `T_Student` and `T_Faculty`. Some of the students teach and some faculty do only research. The Type `T_Student` has the properties *(Year: int)* and *(Teach: boolean)* while the properties *(HiringDate: date)* and *(Teach: boolean)* are defined for `Faculty`. We would like to derive a class `Teacher` of all the persons who teach with the property *(TimeServed: int)* obtained either from *HiringDate* or from *Year* depending on the type of the base object. The class `Teacher` cannot be directly derived from the class `Person` since the useful properties are not defined in `T_Person`. In the following, we propose a way (composed derived class) to solve this problem.

**Definition 3** *Let $(C_1, C_2) \in \mathcal{C}^2$ be a pair of classes. Then:*

- $C_d = C_1 * C_2$ *with a filter* $\Phi$ *and an interface* $\Psi$ *is a composed derived class with* $\Gamma(C_d) = \Psi(\Phi(\Gamma(C_1) \cap \Gamma(C_2))$
- $C_d = C_1 + C_2$ *with a filter* $\Phi$ *and an interface* $\Psi$ *is a composed derived class with* $\Gamma(C_d) = \Psi(\Phi(\Gamma(C_1) \cup \Gamma(C_2))$
- $C_d = C_1 - C_2$ *with a filter* $\Phi$ *and an interface* $\Psi$ *is a composed derived class with* $\Gamma(C_d) = \Psi(\Phi(\Gamma(C_1) - \Gamma(C_2))$

*with* $T(C_d)$ *a sibling of* $Anc(T(C_1), T(C_2))$ *where* $Anc(X, Y)$ *is a function that returns the first common ancestor of* $(X, Y)$ *in the type hierarchy* $\mathcal{T}$.

The semantics of the constructive operations $\{*, +, -\}$ are respectively based on the basic set operations $\cap, \cup$ and $-$. As defined, $\{*, +, -\}$ are binary operations but the

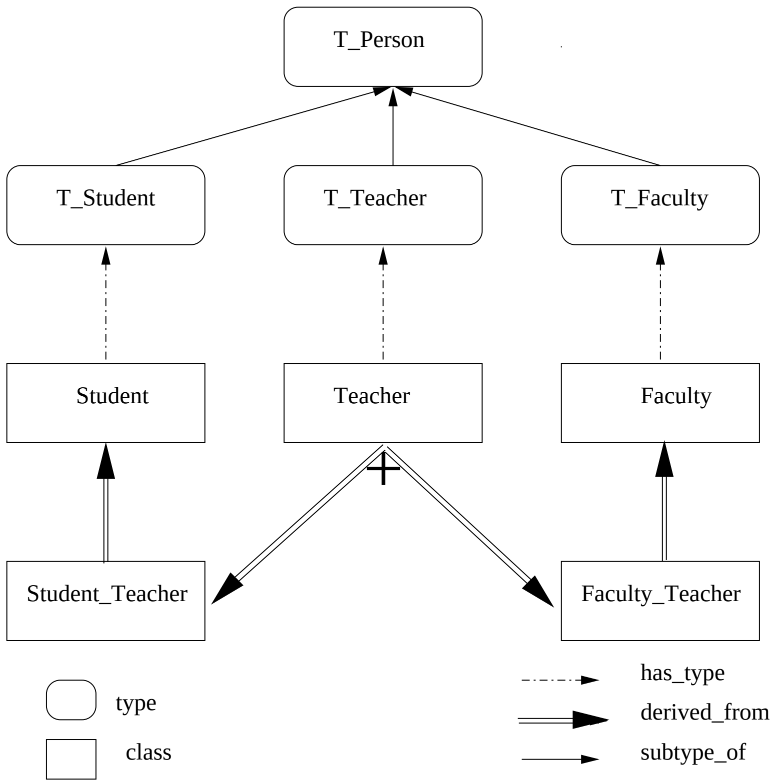



Figure 7: An Example of a Composed Derived Class and its Type.

formulae obtained can be seen as terms and be combined for more complex ones. Note that $C_1$ and $C_2$ can be derived classes as previously defined. The ancestor function $Anc$ works fine when $\mathcal{T}$ is rooted. When this is not the case, a common supertype `T_C` is created for $T(C_1), T(C_2)$. In the worst case, `T_C` will not have any properties in it.

The problem of deriving a class `Teacher` can be solved by defining a simple derived class `Student_Teacher` whose extent is a subset of all the students. In the same way, we derive the class `Faculty_Teacher` from `Faculty`. `Teacher` is then defined as $Teacher = Student\_Teacher + Faculty\_Teacher$. The type `T_Teacher` is a subtype of `T_Person` which is the common ancestor (Figure 7).

### 3.1.2. *Identifying and Updating a Derived Object*

A derived object is always derived from one and only one base object although its properties can be totally different from the properties of the base object. This happens when all the properties of the base class are hidden and new ones are defined for the derived class. Hence, a derived object can be seen as a base object viewed from another angle (the interface function of the derived class). Both the derived object and its corresponding base object can be identified by the OID of the base object (BASE_OID). If we redefine the notion of OID as follows: OID= $< class\_name, BASE\_OID >$ then the base object can be differentiated from the derived one. This OID defines a logical idenfier for any object including the derived ones independently from any view implementation technique. In the case of view materialization with incremental maintenance, an active research area[21,22,23,24], the derived object OID is a key candidate and can be directly used as identifier.

A derived object knows its base object. Therefore, updating a property inherited from the base type can easily be propagated to the base object. Creating new objects for a derived class should first create the objects in the base class with some possible unknown property values.

#### 3.1.3. *Inserting the Type of a Derived Class into the Type System*

The type of a derived class has to be inserted at the right place in the type system. The insertion is simple when the associated type of a derived class is a subtype of the type of the parent class (or an ancestor of the parent classes). This happens when the definition of a derived class uses only the function *augment*. In this case, the new type is simply inserted as a subtype of the type of the parent class. The problem of inserting the new type into the type system is more complex when the function *hide* is used in the definition of a derived class, especially when the type associated to the parent class inherits some properties from another type. For example, all the behaviors inherited by `T_RestrictedPerson` in Figure 6 and Figure 7 are defined in `T_Person`. Therefore `T_RestrictedPerson` should be defined as a supertype of `T_Person`, and `T_Person` should have locally defined only the behaviors not in `T_RestrictedPerson`. The type system is automatically updated when classes are derived.

#### 3.1.4. *Derived Classes and Subclassing*

As indicated earlier, the class derivation relationship is different from the subclass/superclass relationship. However $\mathtt{X_d}$ derived from a class `X` can be defined as a subclass of `X` ($\mathtt{X_d} \leq_c \mathtt{X}$), if the following two conditions are satisfied:

- $\mathtt{T\_X_d} \leq_t \mathtt{T\_X}$
- $\Gamma(\mathtt{X_d}) \subseteq \Gamma(\mathtt{X})$

These two constraints ensure that we keep the same semantics of subclass/superclass relationships as traditional object models. That is, an object of a subclass is also an object of the superclass, and any behavior defined for objects of the superclass can be applied to objects of the subclass.

### 3.2. *Derived Image Classes and Derived Salient Object Classes*

### 3.2.1. *Derived Image Classes*

A derived image class, in addition to defining a new type, selects the the needed physical salient objects from the base image possibly with new semantics.

**Definition 4** *A* **derived image class** *is a class derived from an image class that specifies the valid logical salient objects for images in its extent. If $i_d$ is an image derived from an image $i$, then the set of physical salient objects contained in $i_d$ is a subset of the set contained in $i$. The physical salient objects in $i_d$ are those for which the corresponding logical salient objects belong to one of the valid logical salient objects.*

In addition to redefining the type, a derived image class redefines the content of the images it contains. For example, from the *ClothingCatalog* class defined in Figure 3, we can derive two different catalogs giving different interpretations of the images in the *ClothingCatalog* image class: the customer catalog class (*CustomerCatalog*) and the clothing company catalog (*CompanyCatalog*). The customers are interested in finding clothing from the catalog. Therefore, the valid logical salient object class is *Clothing*. In addition to the clothing, the company may be interested in keeping some information about the models.

A composed derived image class can also be created. For example, from *ClothingCatalog* we can derive the class *FemaleClothingCatalog*. We can also derive *FemaleShoesCatalog* from *ShoesCatalog*. *FemaleClothingCatalog* and *FemaleShoesCatalog* can be combined using the + operator to derive a class *FemaleApparelCatalog*. The common ancestor of *FemaleClothingCatalog*, and *FemaleShoesCatalog* is *Catalog*. Therefore the type of *FemaleApparelCatalog* has to be a sibling of the type of *Catalog* (Figure 8).

### 3.2.2. *Derived Logical Salient Object Classes*

Definition 5 defines the content of an image $i$ as a pair $Cont(i) =< \mathcal{P}^i, s >$ where $\mathcal{P}^i$ represents the physical salient objects and the function $s$ maps each physical salient object to a logical salient object. An image $i_d$ can be derived from $i$ and $Cont(i_d) =< \mathcal{P}^i, s_d >$. Assume we derived a logical salient object class $L_1$ from the logical salient object class $L$ and that all the physical salient objects in $\mathcal{P}^i$ are mapped to objects of $L$. If we note by $f$ the interface function that transforms an object of $L$ to an Object of $L_1$, and we define $s_d = s \circ f$, then $i_d$ is a derived image that contains $L_1$ objects.

For example, the classes `FemaleClothing` and `FemaleShoes` can be respectively derived from *Clothing* and `Shoes` (Figure 4). A composed derived class *FemaleApparel* can be derived from the two previously derived classes and the derived image class `FemaleApparelCatalog` can be defined as images containing female apparels. Of course, `T_FemaleClothing` and `T_FemaleShoes` can respectively be different from `T_Clothing` and `T_Shoes`. `T_FemaleApparel` is then, a sibling of `T_Apparel`.

**Definition 5** *An* **image view** *is defined by:*

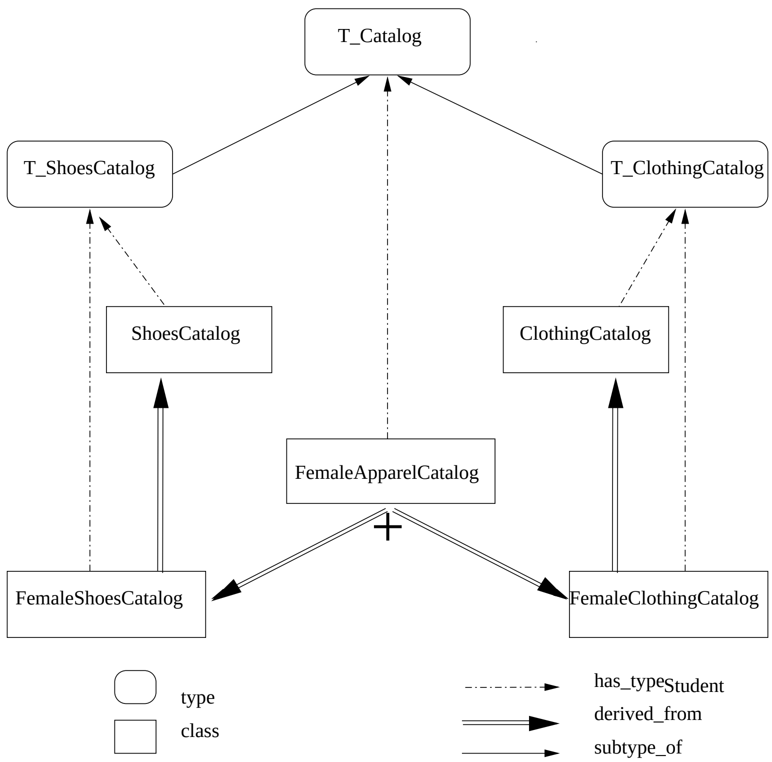



Figure 8: An Example of a Composed Derived Image Class

- *a derived image class*
- *an interpretation context defined as a set of selected logical salient object classes (base or derived).*

## 4. The Image View Definition Language

The view definition language allows the definition of derived image and logical salient classes. Queries in the view definition are expressed in MOQL (Multimedia Object Query Language)[25], the query language defined for DISIMA. MOQL extends the standard object query language, OQL[26] with predicates and functions to capture temporal and spatial relationships. Most of the extensions have been introduced in the *where* clause in the form of new predicates as the new operators introduced by the DISIMA model are joins. The convention used in the language definition is: [ ] for optional, { } (different from **{}** which are part of the language) for 0 to n times, and | for one and only one among different possibilities.

- Create a derived class<br>
  **derive {** <derived class name> **from** <class definition ><br>
  [ **augment** {<virtual behavior name> **as** <query> |<br>
  <function name> ;}]<br>
  [ **hide** < behavior list>]<br>
  {**cast** < behavior—*type* > **into** <new type> }<br>
  [ **content** < interpretation context>]<br>
  **extent** <extent name > [**as** <query>]<br>
  **}**;<br>
  <br>
  <class definition> := <class name> |<br>
  (<class definition > **union** | **intersect** | **minus** <class definition>)
- Delete a derived class<br>
  **delete** <derived class name | image view name>
- Create an Image View<br>
  **create image view** <image view name> **as**<br>
  **{** {<derived image class definitions >};<br>
  {<derived logical salient object class definitions >}<br>
  **}**
- Select an Image View<br>
  **set image view to** <image view class name>

The *derive* command is used to define a derived logical salient object class, as well as derived image classes. The classes that the *derived classes* are derived from one or more base or derived classes. The query in the *extent* clause defines the derived class extent and must return a unique subset of the combination of the parent class extents. The *augment* clause is used to define new properties. A query can invoke an existing property. In this case, the keyword *this* is used to refer to the current object. If $(\alpha : T(C))$ is a property and $C_d$ is a class derived from $C$, then the clause *cast* can be used to cast the type of $\alpha$ into $T(C_d)$. The *content* clause allows to define the logical salient object classes that will form the interpretation context. This clause is used only for derived image classes and is used to select the interpretation context. It is complemented by the *cast* clause. If a logical salient object class mentioned in the view is a derived class, then the salient objects from parent image should be casted to the derived type. Assume a salient object class $S_d$ is derived from class $S$ and an image $i$ (element of the image class $I$) contains a salient object of type $T(S)$. If we derive an image class $I_d$ from $I$ with the clause *content* $S_d$, image $i_d$ derived from $i$ will contain a salient object of type $T(S_d)$ instead of $T(S)$. For example, in the image view `CustomerClothing` that follows, an image of `CustomerCatalog` contains elements of `CustomerClothing`, rather than `Clothing` as salient objects. Derived classes can dynamically be deleted.

An image view defines the interpretation context for one or more image classes. An image view is created using the command *create image view* and contains derived image and salient object classes. The salient objects classes do not have to be derived. They can be base classes introduced to provide complementary interpretations. In this case the image view just selects the the logical salient objects to form the interpretation context. The *Set Image View to* command is used to set the current image interpretation for querying purpose.

### 4.1. *Examples of Image Views*

In the following, we give some examples of image views derived from the catalog database. The corresponding schema expressed in the ODMG object model[26] is given in the Appendix. The schema given in the Appendix can be seen as the most general view, the view of the company owner of the catalog: each image contains models and clothes. The example views correspond to the Customer View, the Female Clothing Catalog View, and the Female Apparel Catalog view.

```
Create Image View CustomerCatalog
{
    derive    {CustomerClothing from Clothing
    augment   inStock as this.inStock();
              avgPriceForType as
              avg(Select c.price
              From Clothes c
              Where c.type = this.type);
    hide      stock, lastOrderDate, lastArrivalDate, nextArrivalDate
    extent    CustomerClothes};
    derive    {CustomerCatalog from ClothingCatalog
    hide      photographer, date, time, place
    cast      accessories into Set<Ref<CustomerCatalogs>>
    cast      Clothing into CustomerClothing
    extent    CustomerCatalogs
    content   CustomerClothing}
}
```

The derived class `CustomerClothes` redefined `Clothes` for the customers' use. Attributes *stock, lastOrderDate, lastArrivalDate, nextArrivalDate* are hidden and the virtual attribute *avgPriceForType* returns the average price for this type of clothing.

The image view `CustomerCatalog` uses the image class `Catalog` renamed as *CustomerCatalog* with *CustomerCatalogs* as its extent name. All the images are available but their content will be limited to objects of `CustomerClothes` which redefines `Clothing`. Attributes *photographer, date, time, place* are hidden. The attribute *accessories* was defined as a set of images from *Catalog*. Its type has to be changed to set of *CustomerCatalogs* to ensure consistency.

```
Create Image View FemaleClothingCatalog
{
    derive    {FemaleClothing from CustomerClothing
    extent    FemaleClothes as
              Select c
              From CustomerClothes c
              Where c.sex = 'female' or c.sex = 'unisex'};
    derive    {FemaleClothingCatalog from ClothingCatalog
    hide      photographer, date, time, place
    cast      accessories into Set<Ref<CustomerCatalogs>>
    cast      CustomerClothing into FemaleClothing
    extent    FemaleClothingCatalogs
    content   FemaleClothing};
}
```

Only images containing female items are selected from the clothing catalog. The salient objects are restricted to female clothing.

```
Create Image View FemaleApparelCatalog
{
    derive    {FemaleShoes from Shoes;
    augment   inStock as this.inStock();
              avgPriceForType as
              avg(Select s.price
              From Shoes s
              Where s.type = this.type);
    hide      stock, lastOrderDate, lastArrivalDate, nextArrivalDate

    extent    FemaleShoesExtent as
              Select s
              From ShoesExtent c
              Where c.sex = 'female' };
    derive    {FemaleShoesCatalog from ShoesCatalog
    hide      photographer, date, time, place
    extent    FemaleShoesCatalogs
    content   FemaleShoes};
```

**derive** {FemaleApparelCatalog **from** FemaleClothingCatalog **union** FemaleShoesCatalog
**extent** FemaleApparelCatalogs};
**}**

The *FemaleApparelCatalog* combines the *FemaleClothingCatalog* and the *FemaleShoesCatalog* into a new derived catalog.

### 4.2. *Implementing Derived Classes in DISIMA*

The distinction between types and classes is not supported by most object-oriented languages in current use. DISIMA is being implemented on top of ObjectStore[27] using C++. DISIMA provides types for image and logical salient objects that can be subtyped by the user. The implementation we describe in this section simulates the idea using C++. We implement all our types as C++ classes. We call these C++ classes *type classes* and their names start with *T_*. For example `T_Person` will be a type class for the class *Person*. Our classes are objects of the C++ class *C_Class*. *C_Class* has a subclass *D_Class* for derived classes. The properties defined for *C_Class* are:

- Name: name of the class
- Type: type class name
- SuperclassList: list of the superclasses
- SubClassList: list of the subclasses
- ShallowExtent (virtual function): The shallow extent of the class
- DependentList: list of classes derived depending on this one

The properties defined for *D_Class* are:

- Base ClassList: list of the classes it is derived from
- Filter: filter function
- ShallowExtent: redefined
- MaterializationFlag: set when the ShallowExtent is up-to-date
- Change: function used to unset the MaterializationFlag

The *DependentList* in the class *C_Class* contains all the classes derived from that class and also all the derived classes for which an augmented property is computed using objects of that class. Since the type of a derived class can be different from the type of its base class we choose to materialize the derived class extent. An object of *C_Class* represents a user's class and the extent (*ShallowExtent*) property returns objects of the type class (*Type*). The *SubClassList* can be used to recursively compute the deep extent. To simplify the materialization process, we only store one level of base class. That is, the *Base ClassList* of a derived class contains only non-derived classes. A derived class extent is materialized the first time the class is referred to and the materialization flag is set. Each time new objects are created, modified or deleted in a base class, a change message is sent to each of the classes in the *DependentList* to unset the materialization flag. If the materialization flag is unset when a derived class is accessed, the derived class extent is recomputed and the materialization flag is reset.

When the augmented properties of a derived object are computed from the single base object without any aggregate, the management algorithm for incremental view maintenance can easily be implemented as follows. An object of a derived class contains the OID of the base object it is derived from. The *Change* method passes the OID of the changed base object (new, deleted or updated) to the derived class object where it is kept in the *ChangeList* of the derived class object. The *ChangeList* can then be visited to update or create the derived objects for modified or new base objects and to delete derived objects corresponding to deleted base objects.

## 5. Object Models and Views

### 5.1. *Object Models*

In most object systems, as indicated earlier, the terms *type* and *class* are used interchangeably to refer to one or more of the following notions: (1) real-world entity, (2) programmatic interface, (3) implementation of the programmatic interface, (4) internal machine representation, (5) factory for creation of instances, (6) maintainer of the extent (the set of all the instances of a class). For example, in GemStone[28], *class* refers to all of the above notions. In ObjectStore[27], *type* refers to (2) and (4), while *class* refers to (1), (3), and (5). Alternately, the *type* of an object of a particular *class* is implicitly defined by that *class*, so it may also be argued that *class* in ObjectStore actually refers to all notions except (6). In $O_2$[29], *type* refers to (2) and (4), while *class* is a special kind of *type* that refers to (1) through (6). In Iris[30], *type* refers to (1), (4), (5), and (6); (2) and (3) are defined with respect to the *type* but are not a part of the *type*. In VODAK[31] (which is built on top of ObjectStore but has a different data model), *type* refers to (2), (3), and (4), while *class* refers to (1) and (5). *Type* in VODAK has two components: the *interface*, referring to (2), and the *implementation*, referring to (3). Both parts of the VODAK *type* can define the representation. In relational database systems, *type* refers to (2) and (4). XML[32] and DAML+OIL[33], through XML Schema, offer a variety of types that refer to (2) and (4) But XML has no explicit constructs to define classes. RDF[16], has only literal (strings) as type (2) and (4) and uses the $< rdfs : class >$ to define classes (5).

### 5.2. *Views and Objects*

Several object view mechanisms have been proposed since the early 1990s[3,4,5,6,34]. In general, the main problems with these views[6] are (i) expressive power (restrictions on queries defining views); (ii) reusability and modeling accuracy (insertion of the views into the generalization hierarchy); and (iii) consistency (stability in OID generation).

Problems (i) and (ii) are somewhat related. For example, using the view mechanism in [5], if the user wants the view class to be linked to the generalization hierarchy, the query that generates the view class has to be restricted. In addition, problem (ii) raises a typing problem (how is the type of the virtual class related to the type hierarchy?) and a classification problem (how is the extent of a virtual class related to the existing ones?). Finding an answer to these two questions in an environment where the only relationship is the *is-a* relationship can lead to contradictions. The distinction between the derivation hierarchy and the generalization hierarchy in our proposal, based on the distinction between type and class, provides an elegant solution to problems (i) and (ii). For example, the fact that the associated type of a class is not tied to the class allows a view based on a projection of its parent type behaviors to be created with an associated type defined as a supertype of the parent type, while the extent is a subset of the extent of the parent class extent. The type of a derived class is computed from the view definition and inserted into the type system. The insertion of the new type can lead to local reorganization of the type hierarchy without any effect on existing classes.

In addition, the object view mechanism proposed in this paper allows us to derive classes from several existing ones. Problem (iii) is solved by the fact that a derived object is seen as a base object with a different interface function. A derived object and its base class share the same OID but are uniquely identified by the pair $< class\_name, OID >$, which is invariant even if the derived object is recomputed. These problems needed to be solved in order to provide clean image views as for the user there should not be any difference between an image view (a derived context) and a base context. The two levels of salient objects ensure the semantic independence and multi-representation of salient objects.

## 6. Conclusion

Because of their nature, images are often represented using object or object-relational models. Hence, an image view mechanism should should be an extension of object models. The aim of an image view mechanism is to provide a way to define different context of interpretations for the same image without having to duplicate the image data.

The image view is based on an object model. The distinction between type and class in the model used in our view mechanism allows the manipulation of both the type and the extent in a view definition. The distinction also facilitates the integration of the type of a virtual class into the type hierarchy. A virtual class can be derived from one or several classes without any constraint on the query defining it. The DISIMA model separates the objects contained in an image (*physical salient objects*) from their semantics (*logical salient objects*). Using our object view mechanism, we proposed an image view mechanism that allows us to give different semantics to the same image. For example, a derived image class can be defined by deriving new logical salient object classes that give new semantics to the objects contained in an image or by hiding some of the objects using a derived image class.

The main contributions of this paper are the proposal of a powerful object-oriented view mechanism based on the distinction between class and type, a proposal of an image view mechanism based on image semantics and the image view implementation using a language that does not intrinsically support the distinction between class and type.

**Acknowledgements**

The authors thank Tuan An Ta and Jean-Marc Saglio for fruitful discussions on object models and views.

This research is supported by a strategic grant from the Natural Science and Engineering Research Council (NSERC) of Canada and the Institute for Robotics and Intelligent System.

The work was conducted while the authors were affiliated to University of Alberta.

**References**

1. The DISIMA Group. The DISIMA online demonstration. Accessible at http://db.uwaterloo.ca/ tozsu/projects/multimedia.
2. V. Oria, M. T. Özsu, X. Li, L. Liu, J. Li, Y. Niu, and P. J. Iglinski. Modeling images for content-based queries: The DISIMA approach. In *Proceedings of 2nd International Conference of Visual Information Systems*, pages 339—346, San Diego, California, December 1997.
3. E. Bertino. A view mechanism for object-oriented databases. In *Proceedings of International Conference on Extending Data Base Technology*, pages 136—151, Viena, Austria, March 1991.
4. S. Abiteboul and A. Bonner. Objects and views. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 238—247, Denver, Colorado, May 1991.
5. Casio Santos, Claude Delobel, and Serge Abiteboul. Virtual schema and bases. In *Proceedings of International Conference of extending Data Base Technology*, pages 81—94, Cambridge,UK, March 1994.
6. X. Ye, C. Parent, and S. Spaccapietra. Derived objects and classes in DOOD system. In *4th International Conference on Deductive and Object-Oriented Databases, DOOD 95*, pages 539—556, Singapore, December 1995.
7. V. Oria, M. T. Özsu, D. Szafron, and P. J. Iglinski. Defining views in an image database system. In *Proceedings of the 8th IFIP 2.6 Working Conference on Database Semantics (DS-8) "Semantic Issues in Multimedia Systems"*, Rotorua, New Zealand, January 1999.
8. A. W. M. Smeulders, M. L. Kersten, and T. Gevers. Crossing the divide between computer vision and data bases in search of image databases. In *Proceedings of 4th IFIP 2.6 Working Conference on Visual Database Systems - VDB 4*, pages 223—239, L'Aquila, Italy, May 1998.
9. A. Del Bimbo. *Visual Information Retrieval*. Morgan Kaufmann Publishers, 1999.
10. N. Boujemaa, J. Fauqueur, M. Ferecatu, F. Fleret, V. Gouet, B. Le Saux, and H. Sahbi. Interactive specific and generic image retrieval, MMCBIR. http://www-roq.inria.fr/cgi-

bin/imedia/ikona, 2001.

11. G. A. Miller. Worldnet: A lexical database for english. *Communication of ACM*, 38(11):39—41, May 1995.
12. J. Fan, X. Zhu, M. S. Hacid, and A. K. Elmagarmid. Model-based video classification toward hierarchical representation, indexing and accessing. *Journal of Multimedia Tools and Applications, Special Issue on Multimedia and Internet*, 17(1):97—120, 2002.
13. S. Weibel, J. Godby, E. Miller, and R. Daniel. OCLC/NCSA metadata workshop report. Online Computer Library Center, 1995.
14. S. L. Weibel and C. Lagoze. *International Journal on Digital Libraries*.
15. J. Hunter and F. Nack. An overview of MPEG-7 DDL proposals. *Signal Processing: Image Communication Journal*, 16:271—293, 2000.
16. O. Lassila and R. R. Swick. Resource description framework (RDF) model and syntax specification. W3C recommendation. Available at http://www.w3.org/TR/REC-rdf-syntax/, February 1999.
17. R. Ramakrishnan and J. Gehrke. *Database Management Systems, 2nd edition*. McGraw Hill, 2000.
18. S. Santini and A. Gupta. An extensible feature management engine for image retrieval. In *Proceedings of SPIE Vol. 4676, Storage and Retrieval for Media Databases*, San Jose, California.
19. O. Günther. Efficient computation of spatial joins. In *Proc. IEEE 9th Int. Conference on Data Engineering*, 1993.
20. Y. Leontiev, M. T. Özsu, and D. Szafron. On separation between interface, implementation and representation in object DBMSs. In *Proceedings of Technology of Object-Oriented Languages and Systems 26th International Conference (TOOLS USA98)*, pages 155—167, Santa Barbara, August 1998.
21. D. Agrawal, A. El Abbadi, A. Singh, and T. Yurek. Efficient view maintenance at data warehouse. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 417—427, Tucson, Arizona, May 1997.
22. L. S. Colby, A. Kawaguchi, D. F. Lieuwen, I. S. Munick, and K. A. Ross. Supporting multiple view maintenance policies. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 405—416, Tucson, Arizona, 1997.
23. K. A. Ross, D. Srivastava, and S. Sudarshan. Materialized view maintenance and integrity constraint checking: Trading space for time. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 507—518, Montreal, Canada, June 1996.
24. B. Adelberg, H. Garcia-Molina, and J. Widom. The STRIP rule system for efficiently maintaining derived data. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 147—158, Tucson, Arizona, May 1997.
25. J. Z. Li, M. T. Özsu, D. Szafron, and V. Oria. MOQL: A multimedia object query language. In *Proceedings of the 3rd International Workshop on Multimedia Information Systems*, pages 19—28, Como, Italy, September 1997.
26. R. G. G. Cattell, D. Barry, D. Bartels, M. Berler, J. Eastman, S. Gamerman, D. Jordan, A. Springer, H. Strickland, and D. Wade, editors. *The Object Database Standard: ODMG 2.0*. Morgan Kaufmann, San Francisco, CA, 1997.
27. C. Lamb, G. Landis, J. Orenstein, and D. Weinreb. The objectstore database system. *Communications of ACM*, 34(10):19—20, 1991.
28. R. Bretl, D. Maier, A. Otis, J. Penney, B. Schuchardt, J. Stein, E. H. Williams, and M. Williams. The GemStone data management system. In W. Kim and F. H. Lochovsky, editors, *Object-Oriented Concepts, Databases and Applications*. Morgan

Kaufmann, 1989.

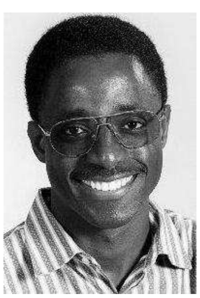
29. C. Lcluse, P. Richard, and Fernando Vlez. *Building an Object-Oriented Database System, The Story of O2*, chapter O2, an Object-Oriented Data Model, pages 77–97. Morgan Kaufmann, 1992.
30. D. Fishman. Overview of the Iris DBMS. In W. Kim and F. H. Lochovsky, editors, *Object-Oriented Concepts, Databases and Applications*, pages 219—250. Morgan Kaufmann, 1989.
31. VODAK V4.0 user manual. GMD Technical Report No. 910, 1995.
32. T. Bray, J. Paoli, and C. M. McQueen. The extensible markup lamguage (xml) specification. W3C recmmendation. Available at http://www.w3.org/TR/REC-xml-19980210, 1998.
33. D. Connoly, F. v. Harmelen, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, and L. A. Stein. DAML+OIL (march 2001) reference description. W3C note. Available at http://www.w3.org/TR/2001/NOTE-daml+oil-reference-20011218, December 2001.
34. H. A. Kuno and E. A. Rundensteiner. The MultiView OODB view system: Design and implementation. *Journal of Theory and Practice of Object Systems (TAPOS)*, 2(3):202—225, 1996.

## Appendix A

```
class Image{
    Set<Ref<Representation>> representations;
    Set<Ref<PhysicalSalientObject>> physicalSalientObjects
            inverse image;
    // Methods
    display(); }
class Catalog: Image{
    Person photographer; Date date; Time time; String place; }
class LogicalSalientObject{
    Set<Ref<PhysicalSalientObject>> physicalSalientObjects
            inverse logicalSalientObject;
    //Methods
    Region region(Image m); // salient object's region in image m
    Color color(Image m); // salient object's color in image m
    Texture texture(Image m); // salient object's texture in image m }
class Person: LogicalSalientObject{
    String name; String occupation; Address address; }
class Model: Person{
    String : agency; }
class Apparel: LogicalSalientObject{
    String name; String type; Real price; Set<Real> size;
    Manufacturer manufacturer; Integer stock; String colors;
    Date lastOrderDate; Date lastArrivalDate; Date nextArrivalDate;
    //Methods
    Boolean inStock(); // true if the the clothing is in stock }
```

```
class Clothing: Apparel {
    Set<Ref<Catalog>> accessories;// images of items that match with the cloth }
class Shoes: Apparel {
    String sole;String upper; }
class PhysicalSalientObject{
    Ref<LogicalSalientObject> logicalSalientObject
            inverse physicalSalientObjects;
    Ref<Image> image
            inverse physicalSalientObjects;
    Region region; Color color; Texture texture }
Set<Ref<LogicalSalientObject>> LogicalSalientObjects; //all salient objects
Set<Ref<Person>> Persons; //salient objects of type Person
Set<Ref<Model>> Models; //salient objects of type Model
Set<Ref<Clothing>> Clothes; //salient objects of type Clothing
Set<Ref<Shoes>> ShoesExtent; //salient objects of type shoes
Set<Ref<Image>> Images; //all images
```

## Photo and Bibliography



**Vincent Oria** received his Ph.D. in computer science from Ecole Nationale Supérieure des Télécommunications (ENST), Paris, France, in 1994.

From 1996 to 1999, he was he was a post-doctoral fellow in the Department of Computing Science of the University of Alberta, Edmonton, Canada. Since January 2000, he is an assistant professor of computer science at New Jersey Institute of Technology.

Dr. Oria's research interests involve multimedia data management, spatial data management, and e-learning systems. He has been a visiting professor or researcher at National Institute of Informatics (Tokyo, Japan), GMD-IPSI (Darmstadt, Germany), ENST (Paris, France) and University of Paris-IX Dauphine (Paris, France). He has served on a number of conference program committees.

Dr. Oria is a member of the Association for Computing Machinery.

**M. Tamer Özsu** received his Ph.D. in computer and information science from The Ohio State University, Columbus, USA, in 1983.

From 1984 until 2000, he was with the Department of Computing Science of the University of Alberta, Edmonton, Canada. Since 2000, he is a professor of computer science and a faculty research fellow at the School of Computer Science, University of Waterloo, Canada. Dr. Özsu's research interests are in multimedia data management, distributed data management, and structured document management. He has authored/co-authored/edited about seven books. He is the Coordinating Editor-in-Chief of The VLDB Journal (VLDB Foundation), and serve on the editorial boards of Distributed and Parallel Databases Journal (Kluwer Publishers), Information Technology and Management (Kluwer Publishers), World Wide Web Journal (Kluwer Publishers), and Springer Book Series on Advanced Information & Knowledge Processing.

Dr. Özsu is the current Chair of ACM Special Interest Group on Management of Data (SIGMOD), a past trustee of the VLDB Endowment (1996-2002), and a founding trustee of the International Federation of Cooperative Information Systems. He chaired the Computer and Information Science Grant Selection Committee of the Natural Sciences and Engineering Research Council of Canada during 1993-94 and served as the member of the same Committee from 1991 to 1993. He held the University of Alberta McCalla Research Professorship for 1993-94 and was Acting Chair of the Department of Computing Science during 1994-95.

Dr. Özsu is a member of the Association for Computing Machinery and IEEE Computer Society.