A Semantic Data Model for Embedded Image Information*

James Griffioen
Rajiv Mehrotra†
Rajendra Yavatkar

Department of Computer Science
University of Kentucky
Lexington, KY 40506-0027
{griff,rajiv,raj}@ms.uky.edu

August 31, 1994

Abstract

Historically, database systems have provided convenient methods for obtaining specific information from a large repository of data. All the information was readily available and the mapping between the data and its semantics was straightforward. However, the increasing availability of multimedia data sources introduces new data forms for which the mapping between data and its semantics is not clear. In particular non-alphanumeric data such as images, videos, graphs, charts, etc. contain large amounts of information that are difficult to quantify in a complete and concise fashion. Moreover, the semantic information contained in such data is often application specific.

This paper addresses the problem of embedded semantic information. The basic idea is the integration of a data processing component into data modeling systems to allow additional (possibly application dependent) information to be extracted. This paper proposes a new object-oriented data modeling approach, shows how such a design can be applied to a pictorial information system, and describes a prototype system with an example application.

Index terms: Image Information Modeling, Image Data Management, Image Databases, Object-Oriented Modeling, Multimedia Information Management.

---

*Author names appear in alphabetical order. This work was supported in part by the NASA Langley Research Center under grant NAG-1-1276 and by a NSF Research Initiation award no. NCR-9111323.

†Department of Computer Science, Center for Robotics and Manufacturing Systems, University of Kentucky, Lexington, KY 40506.

1 Introduction

New applications requiring efficient management of visual information (such as images, videos, graphs, charts, etc.) are rapidly evolving as a result of the recent widespread availability of hardware and software support for visual applications. Examples of such applications include weather information management, medical information management, environmental pollution information systems, space exploration, manufacturing information management, genome research, training and education systems. These applications require representation, storage, manipulation, and management of visual information. In fact, stored images (or visual data) are the central source of information for many such applications. In many cases, the stored images are themselves the entities or objects to be represented and managed by the system. Several challenging issues need to be handled in the design of a visual information management systems. Some of these issues are: *data modeling, data compression, content-based visual information retrieval, and integrated query processing.* This paper is concerned only with the data modeling aspect of visual information management. Specifically, general requirements concerning pictorial (image) information modeling are discussed and an object-oriented system that meets all the requirements is proposed.

Traditional information (data) modeling and management techniques are not suitable for the representation and management of pictorial information [?]. Pictorial information is not modeled or managed in the raw data form in which it typically occurs. Instead, it must first be processed through a series of steps to capture and extract the useful information content. However, the semantics of an image and the useful information contained in an image depend largely on the application using the image. For example, consider a satellite image or an aerial photograph. A defense application may identify various regions (roads, bridges, specific structures) in the photograph as target sites and associate target specific identification information with each region. However, a geologist might concentrate on identifying and classifying land and water masses according to her specific interest. In other words, each application defines its own view or interpretation of what is important and meaningful in an image. Moreover, defining a "view" typically involves applying a view-dependent multistage transformation to extract the desired embedded information.

Each application domain must apply its own set of transformations to detect the meaningful components of an image. We refer to the components of an image as *image objects.* Their descriptions contain no semantic information other than basic image feature identification (e.g., a boundary segment, a rectangular region, etc.). However, given an application domain (e.g., medical imaging, cartography, or meteorology), additional domain specific semantic information can be associated with an image object. That is, each of the image objects that comprise an image corresponds to some object or semantic category in that application domain. We call the domain-dependent categories *domain objects*. Each application domain typically has a reasonably well-defined set of domain objects (e.g., a ventrical or a tumor in medical imaging) that provide useful domain-dependent information about the image. The *image object* $\leftrightarrow$ *domain object* correspondence is the most important pictorial information in most of the applications.

Modeling and representation of application-dependent image data semantics, the associated data-to-semantics transformation processes, modification and evolution of views, and management of the information generated at each stage of the transformation process pose major challenges to multimedia information modeling systems. This paper discusses these key issues in non-textual information management and proposes a new object-oriented image information modeling approach. We also describe a prototype system based on our design that illustrates the utility of our approach with a concrete example. A video tape and/or a demonstration of the prototype will be presented at the conference to complement the paper.

2 Non-Alphanumeric Data Modeling Requirements

We identify the following requirements for modeling multimedia data (in particular, image data).

Dynamic Data Semantics: The semantics associated with the data in a domain (or image) object typically changes often over its lifetime. Therefore, it is important to dynamically change the set of functions (operations) associated with an object after it is instantiated. That is, the set of functions associated with an object should depend on the semantics of the data represented by the object.

Abstract Function Types: Given an image, one usually has a wide range of functions available that can perform a particular image processing operation. An example of such an operation is edge detection. Several edge detection techniques exist and new methods continue to emerge. Each technique implements a different/new algorithm and may be appropriate under different circumstances. Thus, given an image object, it is advantageous not to bind the data to a particular function, but rather to an abstract function class and dynamically select an appropriate function from the abstract class at runtime.

Inheritance: Given a raw image, two or more users (or applications) might process the same image and obtain different semantic data to be used for different purposes. For example, a satellite image might be of interest to both a geologist and a miltary analyst. Initial processing of the raw image (such as noise removal, enahncements) might be common to both applications before domain-specific processing takes over. In such situations, it is important to group common operations and objects together in a class and then let individual applications inherit the image attributes in a subclass avoiding duplication of efforts.

Composition: Composition refers to the merging of two or more distinct objects into a new object. For example, two independent pictures of the same scene may be merged together to produce additional information about the scene (e.g., the depth of objects in the scene). Alternatively, a single facial image might be processed to extract the individual facial features, modify some of the features, and then merge the modified facial features back together to create an object that can be easily searched for certain combinations of facial feature characteristics. In other words, the system must provide the ability to combine processed objects together into compound objects with new functionality that did not exist when the objects were independent.

History mechanism: As discussed earlier, an image typically goes through a series of transformations that extract information from the image or compute new information based on the image. Thus, the state of an object must contain not only the current data and associated semantics, but also the sequence of operations that were applied to the original object to bring it to the current state. Such a history is especially useful in an interactive programming system where a user may wish to test a variety of alternatives for processing the visual information. Note that a history mechanism also includes the notion of saving the results of past processing so that it can be reused by future users/applications. Alternatively, the original user may wish to backtrack to a previous version of the data and resume processing from that point onward.

3 The MOODS Data Model

MOODS supports three new abstractions that distinguish it from ordinary object oriented data modeling: *dynamic data semantics*, *function groups*, and *composition*.

3.1 Semantic Data

Data semantics represent a significant conceptual deviation from the standard object oriented model. In MOODS, each object consists of three components: data structures, methods (functions), and a *data* *semantic* specification. The data semantics portion of the object describes the current semantic meaning of the data currently stored in the data structure component of the object. Over the lifetime of an object, the semantic meaning of the object will vary depending on the current contents of the data structures. The data semantic component allows us to record the current semantics of the data and to adjust (change the binding of) the set of methods associated with the data. This ability allows us to correctly model the data based on the semantic meaning of the data rather than the structure of the data as is the case in an object oriented model.

As mentioned in the introduction, multimedia data such as image data often undergoes a sequence of processing stages before reaching its desired form. Each stage has its own semantic meaning and applicable functions. To model these semantic changes, MOODS allows the methods associated with an object to change both the data and the semantic meaning of the data. Note that changing the semantic meaning of the data effectively changes the class to which the object instance belongs. In MOODS, all methods or functions associated with a particular class have the property that when invoked, they transform an instance of one class into an instance of another class (possibly the same class). These transformations between semantic classes closely model the sequence of processing stages a data item traverses over its lifetime.

MOODS also deviates from the standard object oriented model in terms of the way in which methods operate on the data within an object. MOODS supports *immutable objects*. When a user invokes a method on an object, the method does not modify the current object but instead produces a completely new object. Saving past versions of nontextual data is useful for several reasons. As mentioned in section **??**, a desirable property of nontextual data modeling is the ability to backtrack to an earlier version of the data to restart processing at that point. Backtracking is particularly useful in an interactive data modeling system. Unfortunately, in practice, few methods have an inverse operation requiring us to store previous versions of the object. Also, past versions of data may contain information in an unprocessed state that may be of value to a researcher (e.g., a radiologist may need the original x-ray to confirm conclusions reached from a processed version of the x-ray).

3.2 Abstract Operations and Flexible Binding

Another unique feature of MOODS is that it allows the operations on an object to be defined in terms of abstract methods called *function groups*. Each function group represents an abstract or logical function and actually consists of a set of related functions. Function groups are globally defined and can be incorporated into any class. MOODS objects are defined in terms of function groups rather than specific functions. That is, a class definition specifies the list of abstract operations that apply to the class, not particular functions.

Typically a function group contains a set of semantically or functionally equivalent methods. The set of methods within a function group need not accept the same inputs nor produce the same outputs, however, they should all provide the same or similar logical operation. For example, consider an edge detector function group that represents the abstract operation *edge detection*. The group actually consists of a suite of edge detection methods based on different edge detection algorithms. Even though each one of the edge detection routines perform edge detection, they may differ in terms of their accuracy in detecting edges, execution times, or space requirements. As new edge detector methods become available, they are dynamically added to the edge detector function group and made available to existing and new objects.

Use of function groups raises an important question about resolving ambiguities when more than one function can be selected at run time. To uniquely identify a particular function from a function group, the system needs four pieces of information: function name, types of any implicit inputs, types of implicit outputs, and the data types of the explicit input and output parameters. In many cases, the current semantics of the data provides sufficient information for the system to automatically select a specific function at runtime [?]. However, if an ambiguity arises, the user must select a specific function from the function group. If the system is interactive, this selection may occur at runtime via prompting. Alternatively, a specific default function can be specified at compile time, load time, or runtime.

3.3 Composition

As mentioned in the requirements section, an information modeling system must have the ability to create new objects by combining two or more existing objects together. Moods provides a mechanism by which a user can dynamically create a new object from existing objects. The new object's type and semantic meaning may correspond to a predefined class or may be a completely new, dynamically create, class.

Logically, the Composition operation can be broken into three steps: merge the data from the objects being composed, merge the functions from the objects being composed, and add new abstract functions that operate on the merged data. Composition is similar in many respects to the notion of inheritance in standard object-oriented programming. However, unlike inheritance, composition is the runtime combination of the current data found in the objects being combined.

3.4 Casting

Another necessary feature of a semantic based model is the ability to *cast* an object into a different semantic class. This is particularly true of visual processing which must undergo iterative refinement until a suitable image is produced. Once the image has reached the desired form, the object must be *cast* to the semantic class that it now represents.

4 An Evolving Image Database System

To evaluate our design and new semantic data model, we have developed a new language called *MOODS* which contains support for function groups, semantic data, and a visual (X-Windows Tcl/Tk) runtime interface. We have used MOODS to implement the image processing component of a simple image database system. The following sections describe both the MOODS prototype and an example application.

4.1 The MOODS Prototype

The MOODS language is an extension of C++ with support for semantic data groups, function groups, composition, casting, and a visual runtime interface for processing semantic data. The MOODS compiler is implemented as a preprocessor whose output is then passed through the standard GNU C++ compiler.

Creating a new application typically requires only creating a set of specifications (SPEC_FILE) that define the data and semantic processing stages the data may traverse over its lifetime. The MOODS preprocessor takes as input a SPEC_FILE and a library of function groups that contains definitions and implementations of available function groups. The resulting C++ code is compiled and then linked into a MOODS executable.

MOODS includes a Tcl/Tk (X-window) user interface that helps users push semantic data through the various processing stages at runtime. The user interface displays the stages the data has already traversed and allows the user to interactively direct further processing. The user selects the next processing step using a pop-up menu with the resulting processing stage being displayed on the screen. Backtracking can be achieved easily by clicking on a previous stage to invoke its pop-up menu of applicable functions. Our experience shows that users are rarely satisfied with their first choice of parameters at any give processing stage. Consequently, backtracking has proven to be an crucial capability. Backtracking allows a user to interactively and iteratively refine an image until the desired information is visible/available.

4.2 An Example Application

Consider an image database with integrated image processing capabilities. In such a database, raw images are input to the database and then processed to some degree to obtain information about the image. Although some initial application-independent processing may be applied to the image, each application must apply application dependent processing methods to obtain the information deemed important by the application. We will illustrate this crucial point further with an elaborate example of an image database system that stores digitized music manuscripts. Raw input images can be processed in such a database using conventional image processing steps. However, semantic data such as *notes*, *stems*, and *tails* can only be successfully discerned and recovered from raw images by applying a series of processing steps that take into account the specific structure of music notes.

We have implemented the image processing component of an image database system to demonstrate the utility of the MOODS data model. Assume images are entered into the database system in a raw, unprocessed, format and assume that a variety of image processing functions (and function groups) are available to process an image. For the purposes of our example, the following function groups would prove useful:

Enhance: consists of three enhancement functions, *neighborhood averaging*, *median filtering*, and *histogram modification* to clean up the image for further processing.

Erode: consists of three simple functions that erode an image: *disk* erosion and *horizontal* and *vertical line* erosion.

Dilate: consists of three simple functions that dilate an image: *disk* dilation and *horizontal* and *vertical line* dilation.

Difference: finds the difference between two images.

Union: merges two images into a single image.

Domain Dependent Labeling: this set of functions process the results of previous stages to label the image objects detected (e.g., notes, staff, etc.).

Domain Dependent Analysis: this function group consists of functions that draw conclusions, make observations, make a diagnosis, construct a hypothesis, etc. (e.g., the music is written in a minor key). Some methods may be automatic, others will be interactive and require user analysis.

As the data is processed and moves from stage to stage, the semantics of the data also change. That is, the processing sequence takes the data through a series of semantic states. The system represents each stage of the transformation sequence with a unique semantic class.

Assume we want to add image-object to domain-object processing capabilities associated with the music recognition domain to our image database. To achieve this, we must define a series of semantic states, where each semantic class only allows some subset of the functions taken from the global function group space. For example, Figure **??** illustrates the semantic stages a music image may traverse.

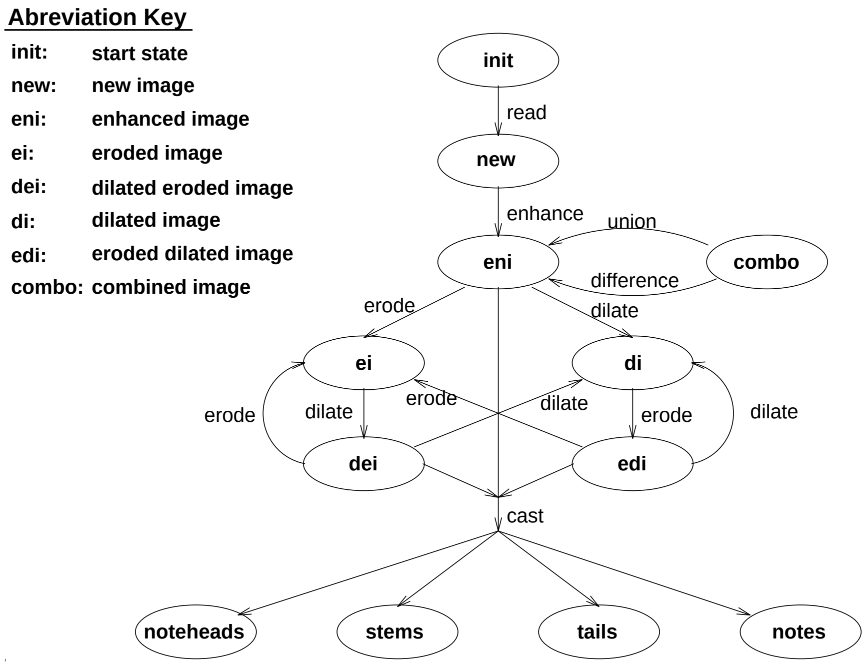

Figure 1: The semantic stages a music manuscript may traverse over its lifetime. Note that erosion/dilation must be applied in pairs (e.g., an eroded image can only be dialated). Also, the semantic states *notes, stem* and *tails* are only reached by "casting" when the desire dilated eroded image or eroded dialated image is achieved.

Each oval represents a different semantic class with semantic-specific functions. The lines between ovals represent the application of a particular function selected from a function group.

Note that the structure and format of the data in all the states shown is identical. However, their semantics differ and thus the functions applicable to each class also differ. Although additional semantic classes are not shown, the data might be processed further by applying domain dependent functions to analyze the data and draw various conclusions.

To create the **Music_Notes** processing component of the database, the user would simply code the semantic processing phase shown in Figure ?? using the MOODS language. Because the global set of function groups already contains the necessary image processing functions, the user only needs to specify the semantic processing phases. If additional function groups or functions within a function group are needed, the user must also code these in MOODS. The specification is then compiled into an executable module that can be applied to the images stored in the database.

To illustrate various MOODS features such as combination, iterative refinement, backtracking, and history/pathsaving, consider the portion of music shown in Figure ??(a). Given this image, a user might apply the following series of processing steps (which are illustrated in figure ??) at runtime to extract

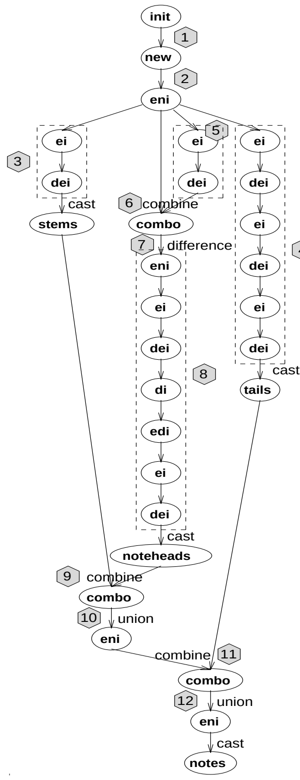

Figure 2: An example of composition.

the notes and their components (note heads, stems, and tails):

1. Read in the original image.

2. Apply a noise removal function to generate an enhanced image.

3. Apply a series of erosion/dilation functions followed by a cast to extract the note stems.

4. Apply a series of erosion/dilation functions followed by a cast to extract the note tails.

5. Apply a series of erosion/dilation functions to extract the staff and the note tails.

6. Create a combined object containing the enhanced image and the image of staff and note tails.

7. Calculate the difference between the two images, creating a new image without the staff and tails.

8. Apply a series of erosion/dilation functions to isolate the noteheads.

9. Create a combined object containing noteheads and stems.

10. Union the noteheads and stems into a single image of notes with stems.

11. Create a combined object containing noteheads/stems and tails.

12. Union the noteheads/stems with the tails to obtain a single image containing complete notes.

This example runtime processing series exercise several features of the MOODS data model. Steps 3, 4, and 5 require MOODS' ability to backtrack to a previous state in order to create additional paths that are eventually combined, via MOODS combination, into a single object. In most cases, the user would repeatedly backtrack, refining the erosion/dilation parameters until the appropriate settings are determined. The long series of erosions/dilations depicted in step **??** illustrates MOODS iterative refinement capabilities needed to extract the note tails from the image. Combination then allows the user to apply multicomponent function groups such as difference or union to achieve the desired results. Finally, MOODS pathsaving/history mechanism allows the user to save this transformation process and reexecute it on the entire manuscript from which this image was taken or on a similar manuscript.

Figures **??** and **??** depict the images at various stages of the processing sequence. The image database, now equipped with image processing functions and semantic information, provides the support necessary for users to extract embedded information but yet stay within the semantic operations of the object. Moreover, future processing is substantially simplified because past transformations can be saved and reused.

We plan to demonstrate the working system or show a video tape of the demonstration at the conference.

References

[1] S. K. Chang, J. Reuss, and B. H. McCormik, An Integrated Relational Database System for Pictures, *Proc. IEEE Workshop on Picture Data Description and Management*, Chicago, IL, April 1977, pp 49-60.

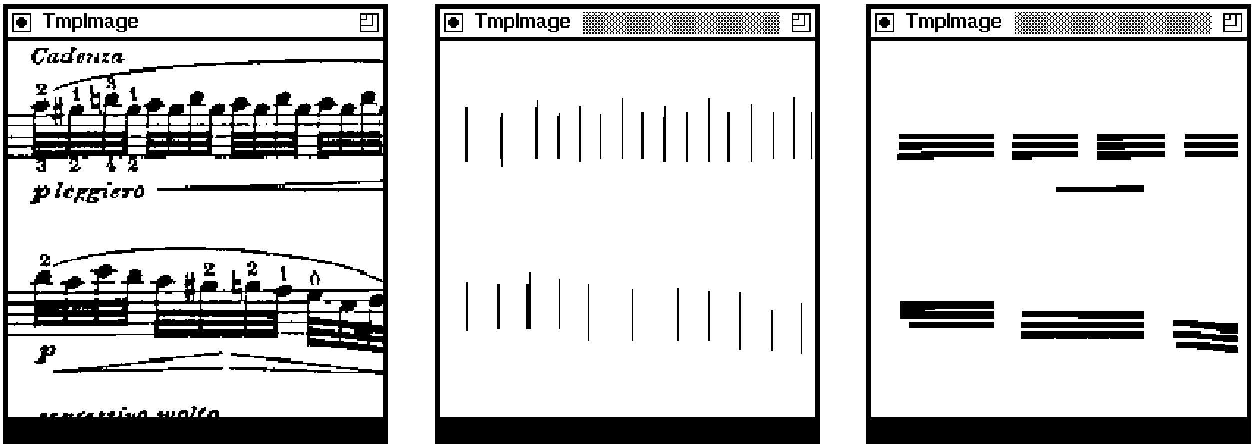

Figure 3: (left to right) (a) The original image before processing, (b) The stems extract by processing step 3, (c) The tails extract by processing step 4.

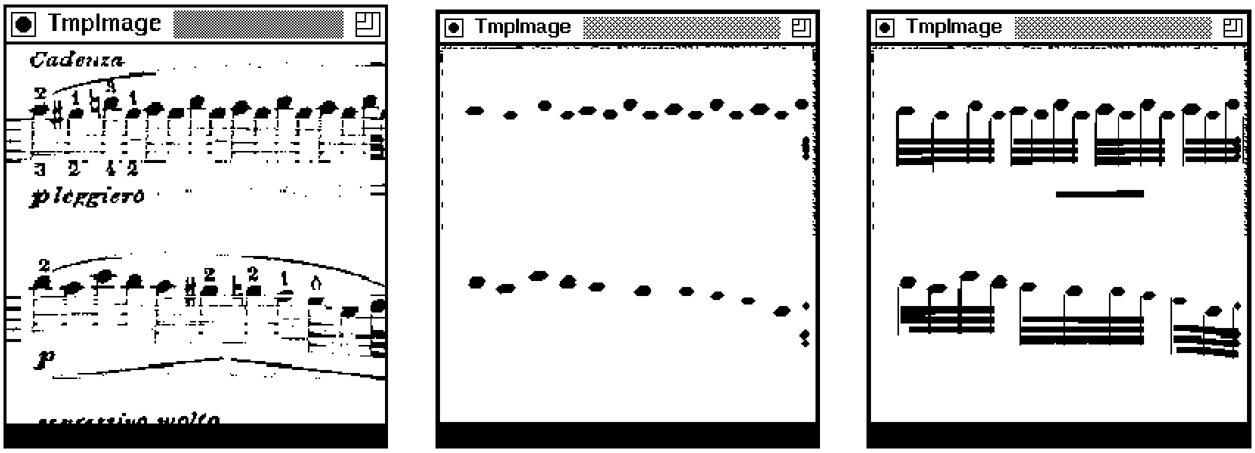

Figure 4: (left to right) (a) The original image minus the note tails after step 7, (b) The noteheads produced by step 8, (c) The final image after the union in step 12.

[2] N. S. Chang and K. S. Fu, A Relational Database System for Images, in *Pictorial Information Systems*, S. K. Chang and K. S. Fu, Editors, North-Holland Publishing Company, Amsterdam, 1980, pp. 405-414.

[3] J. Griffioen, R. Mehrotra, and R. Yavatkar, An Object-Oriented Model for Image Information Representation, *Technical Report No. 232-93*, Department of Computer Science, University of Kentucku, April 1993

[4] W. I. Grosky and R. Mehrotra, Image Database Management, *Advance in Computers*, Vol. 34, M. C. Yovits, Editor, Academic Press, 1992, pp. 237-291.

[5] A. Gupta, T. E. Weymouth, and R. Jain, An Extended Object-Oriented Data Model for Large Image Data Bases, *Proc. 2nd Symposium on Design and Implementation of Large Spatial Databases*, Zurich, Switzerland, August 1991.

[6] A. Gupta, T. E. Weymouth, and R. Jain, Semantic Queries with Pictures: The VIMSYS Model, *Proc. 7th International Conference on Very Large Databases*, Barcelona, Spain, August 1991, pp. 69-79.

[7] H. V. Jagadish and L. O'Gorman, An Object Model for Image Recognition, *IEEE Computer*, vol. 22, No. 12, 1989, pp. 33-41.

[8] T. Kunii, S. Weyl, and J. M. Tenenbaum, A Relational Database Schema for Describing Complex Pictures with Color and Texture, *Proc. 2nd International Joint Conference on Pattern Recognition*, Copenhagen, Denmark, August 1984, pp. 310-316.

[9] W. Kim, Object-Oriented Databases: Definition and Research Directions, *IEEE Transactions on Knowledge and Data Engineering*, vol. 2, No. 3, September 1990, pp. 327-341.

[10] R. Mehrotra and W. I. Grosky, REMINDS: A Relational Model-Based Integrated Image and Text Database Management System, *Proc. Workshop on Computer Architecture for Pattern Analysis and Image Database Management*, Miami Beach, FL, November 1985, pp. 348-354.

[11] B. Stroustrup, What is "Object-Oriented Programming"?, *IEEE Software*, May 1988.