An Object-Oriented Model for Image Information Representation*

James Griffioen
Rajiv Mehrotra†
Rajendra Yavatkar

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

December 14, 1993

Abstract

Efficient management of non-alphanumeric data (such as images, videos, graphs, charts, etc.) is essential to a large number of applications. These applications usually require representation, storage, and management of visual information. This paper discusses the issues in pictorial information modeling, examines the suitability of traditional data modeling approaches, and then proposes a new object-oriented data modeling approach, called *Modeling Object-Oriented Data Semantics (MOODS)*.

A key idea in MOODS is to integrate the image processing component of an image information management system with the data modeling technique. As the image processing techniques take a raw image through a series of transformations, data semantics associated with the image continue to evolve. In addition, a user may provide additional assistance in defining semantics at various stages. We argue that the data modeling technique must capture the sequence of transformations and allow a variety of semantics to be associated with the same data to accommodate different application domains. Additional salient features of MOODS include a notion of abstract function classes and dynamic binding of functions to data.

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

Efficient management of non-alphanumeric data (such as images, videos, graphs, charts, audios, etc.) is essential to a large number of 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. Almost all these applications require representation, storage, and management of visual information. In fact, stored images 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. Data representation or modeling is a key component of any information management system. This paper deals with the problem of pictorial (image) information modeling.

Traditional information or data modeling and management techniques are not suitable for the representation and management of pictorial information [4]. Pictorial information is not modeled or managed in the raw form in which it typically occurs. Instead, it must first be processed through a series of steps to capture the useful information content. Also, the data model must model the pictorial information at every stage of this processing as described below.

To utilize the information present in an image, the image must first be processed (automatically or manually) to obtain its meaningful components and the information it conveys (e.g., regions of uniform intensity, curve segments, regions of uniform texture, etc.). To effectively manage these components, the information is typically converted into a suitable representation that can be easily managed, queried, combined, processed further, or displayed. We refer to the resulting components of the image as *image objects*. They 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 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., ventrical or 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.

We must resolve the following key issues in pictorial information representation and data modeling to adequately manage visual information.

1. In general, several different techniques exist for the same generic image processing task or machine vision task. Usually, image objects and associated descriptions (say boundaries of detected regions) extracted from an image using different techniques differ. Therefore, several different feature-based representation of an image or an image object are possible depending upon techniques employed for its extraction from the image. The information modeling technique should provide a way for the user to associate a group of operations that each performs the same generic task (e.g., boundary detection) with a data type. The desired image object can be extracted from an image (or an image object definition can be developed) by selecting a particular technique from an associated generic task group. In other words, the image processing component of an image data management system should be fully intergrated into the data modeling technique. Also, with the advance in the areas of image processing and machine vision, new techniques can be added to a generic task group or existing technique can be modified or deleted. The modeling techniques should be able to accommodate such changes with ease.

2. As mentioned earlier, for each application domain, the correspondence between the image objects of every stored image and the domain objects must be correctly established. In a large image information management system, this correspondence needs to be established either automatically or semiautomatically (via interaction with the user through an interface) using image processing and machine vision techniques. The overall process of labeling an image and its components in terms of domain-based semantic categories (or domain objects) generally involves a sequence of processing tasks (see Section 2). These processing tasks change the data content. The data modeling technique should be able to represent the entire transformation process as well as the data contents at all stages.

3. The semantic information (or interpretation) associated with an image and its components generally varies considerably from domain to domain. In other words, the set of valid domain objects that can be used to label image components are domain-dependent. Since, in a database system, stored images can be shared among different applications (i.e., multiple domains are possible), the image information modeling technique should be able to represent multiple *image object* $\leftrightarrow$ *domain object* correspondences and the associated mappings.

4. In some application domains, the definition and representation of domain objects are not very

precise and as a result, the same image object can be assigned different domain objects by different users. For example, in medical imaging, the semantic categories (e.g., *large ventricles* or *tortuous*) used by radiologists cannot be precisely defined due to imprecisions in the specifications of the prototypical member of a category as well as the fuzziness of the category boundaries. In other words, for a given domain object, the recognition methods or definitions used by different user may vary in some applications. Each such variation defines a new *user view* of the image data within the same domain. Thus, even for an application domain specific image database system (i.e., only one set of valid domain objects), multiple *image object* $\leftrightarrow$ *domain object* correspondences and mappings are possible.

5. In several application domains, the associated semantic categories (i.e., the set of valid domain objects) or the definitions of domain objects (i.e., *image object* $\leftrightarrow$ *domain object* mappings or user views) evolve or significantly change with time (i.e., during the life time of the system). Such situations quite commonly occur in the areas of space exploration, genome research, medical information management, etc.. In these cases, new image object-to-domain object correspondences may have to be added or the existing correspondences may have to be modified or deleted. The image modeling technique should be able to represent and handle such changes.

Hence, an ideal image information modeling system should be flexible enough to accommodate differing semantic views of the same image and dynamic enough to handle the advances in the areas of image processing as well as the application domains and the resulting view changes and evolutions.

Most of the existing image information modeling techniques are based on the conventional or extended relation data model [8, 1, 2, 10]. All these systems except [10] do not have any provisions to handle the issues mentioned above. Mehrotra and Grosky [10] proposed a relational model called *REMINDS* which allows the hierarchical representation of a generic image object and the associated semantic object. *REMINDS* also provided a mechanism to represent the image interpretation process that maps a given image and its components to the associated semantic categories. However, *REMINDS* has no straight forward way of modeling multiple interpretations of the same image or view evolution. Recently, the advances in object-oriented technologies have been utilized to develop some image information modeling techniques [7, 5, 6]. In [7], Jagadish and Gorman propose a notion of physical and logical hierarchy for image information modeling. The physical hierarchy describes the hierarchical structure of an image, and the logical hierarchy describes the hierarchical structure of the semantic view of that image. There is no provision for modeling and management of multiple logical hierarchies, changes in the hierarchies, or image interpretation operations. VIMSYS [5, 6] is probably the most comprehensive pictorial data model developed so far. It represents image information using a four-layer model. Object-oriented modeling is used in each of the layers. The four layers are:

1. image representation and relation layer for representing multiple representation of each image object,
2. image object and relation layer to represent the image features and their relationships,
3. semantic object and relation layer to represent the correspondences among the image objects and semantic objects, and
4. semantic event and relation layer for representing so called temporal features (i.e., a set of related features over an image sequence). There is no explicit mention of how the mapping strategies and associated methods are modeled and managed.

We are currently designing and implementing a new object-oriented data modeling approach, called *Modeling Object-Oriented Data Semantics (MOODS)*. *MOODS* extends the conventional object-oriented modeling approach to include additional features to handle the above mentioned challenges posed by pictorial and other non-traditional information. Though *MOODS* is a highly general modeling approach applicable to a wide range of information (including multimedia), this paper describes only the image information modeling related features.

2 Generic Classes of Image Information

As mentioned in the previous section, a semiautomatic or fully automatic (if possible) process that extracts desirable image-objects from a stored image and associates the extracted image-objects with the correct domain-objects is necessary in a large image database management system. This process generally involves a sequence of processing steps. An example transformation process appears in Figure 4. The *enhancement* processing stage is responsible for improving the overall quality of the input image for human or machine interpretation. Examples of the type of processing used at this stage included noise removal, contrast enhancement, motion artifacts removal, etc. The *feature extraction* stage processes a given image (input or enhanced) to extract its desirable features, e.g., boundary components, uniform intensity regions, etc. A symbolic description of each of the detected features is developed in the *feature identification stage*. Examples of such descriptions include line or curve equations for boundary components, area, perimeter, average intensity, or intensity variance of a region. Image-objects formed by detected features are identified (using feature descriptions and their relationships together with some a priori knowledge) and described in this stage. Finally, each of the image-objects is labeled with the corresponding domain-object names. If needed, domain-objects can be grouped to create higher level semantic categories or to assign a semantic label to the entire image.

In light of the above discussion on different stages of processing involved in establishing the correspondence between image-objects and domain-objects, we can say that the type of information required to be modeled can be broadly classified into the following eight categories:

1. Iconic: This information consists of the images themselves, which are stored in a digitized format.
2. Image Registration Data: This is the information found in the header and trailer files of the images.
3. Enhanced Iconic: This are enhanced or restored images obtained by applying some enhancement or restoration technique to the input images.
4. Feature (Segmented) Images: These are feature maps (e.g., edge maps, uniform intensity regions, histograms, etc.) corresponding to the input or processed images.
5. Image-Object Descriptions: This is symbolic data representing properties of image-objects, their components, and their interrelationships.
6. Image Object-to-Domain Object Correspondences: This information consists of the relationships between various image-objects and the corresponding domain-objects.
7. Higher Level Domain Objects: These are semantic classes which are defined in terms of the domain objects and their relationships (over one or more images).
8. Domain Object Information: This is conventional textual data describing the domain-objects.

In most cases, it is expected that a user-system interaction will be required to extract all these data. An image information modeling system must facilitate the representation of all these types of information. In addition, as mentioned earlier, modeling of the semantic category identification process is also desirable. Associated with each of these types of information (or data) is a valid set of operations. Some of these operations may use input information that belongs to one category and produce information belonging to some other category. These operations can be grouped into generic classes such that operations in a generic class perform the same function but employ different algorithms. For example, several different edge detection techniques can be grouped into a generic class called “edge detectors”. Thus, associated with each class of information is a set of valid operation groups. In general, an application determines the set of valid operation groups that need to be associated with a given information class. Also the domain-dependent interpretation of an image must be developed from these operations by selecting appropriate operation from the operation groups associated with the information at different stages of the mapping process. Therefore, the image information management system should be capable of representing and managing both the operation groups and the set of information class/operation group relationships.

In the following, we present our goals for modeling pictorial data and we present a new object oriented approach for modeling pictorial data that address these issues.

3 Modeling Data Semantics

Our current research focuses on the design and implementation of modeling and information representation techniques that can capture both the domain-specific and domain-independent semantics associated with visual information. In particular, we are interested in data models that meet the needs of visual information processing as discussed in first two sections. The following section describes the requirements of such a data model, discuss how conventional object-oriented modeling approaches fail to meet the requirements, and provide an overview of our new model, MOODS.

3.1 System Requirements

We identified four basic requirements for modeling image data. Although our goals target systems that model pictorial data, the goals we outline apply to most types of data.

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

Abstract Function Types: Given an image, one usually has a wide range of functions available to 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.

Thus, like abstract data types, *abstract functions* simple define a logical operation and postpone the binding of functions to data until runtime as much as possible. In many cases, binding can be done automatically at runtime based on the current semantics of the data. As a result, the abstract function completely hides the implementation of the function from the user much like an abstract data type hides its implementation.

Built-in History: 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 (in the form of a stream of functions) is especially useful in an interactive programming system where a user may wish to test a variety of alternatives for processing the visual information. Moreover, such historical information may also be useful when new image processing techniques become available by replacing old techniques in the function stream with new techniques.

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 traffic scene might be of interest to a transportation engineer (interested in pedestrian patterns) and also to a law enforcement officer (interested in specific vehicle). Initial processing of the raw image 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. Moreover, some applications may wish to combine the information from two different classes for creating a domain specific object. Thus, inheritance (both single and multiple) is a desirable attribute in a data model.

3.2 Conventional Object Oriented Approach

Object oriented approach[11] is attractive for data modeling for a variety reasons and Kim [9] provides an excellent description of object-oriented data models. First, object oriented system provide modularity with grouping of data and functions together into an abstract data type. Second, object oriented systems allow creation of an hierarchy (or a lattice) of objects through single or multiple inheritance.

However, the conventional object oriented systems do not currently support the notion of abstract function classes and do not dynamically associate semantics with an object. Moreover, as described earlier, interactive visual information processing/management may benefit from a built-in history mechanism to captures the series of transformations an image data undergoes over its lifetime. Object-oriented systems do not provide such a mechanism.

4 The MOODs Data Model

The previous section outlined our requirements for the processing, management, and storage of non-alphanumeric data. To satisfy these requirements we have defined a new data model which we call *MOODS* (Modeling Object Oriented Data Semantics). MOODS has its roots in the object oriented model yet differs substantially from the standard object oriented paradigm in several ways. In particular, MOODS supports two new abstractions that distinguish it from ordinary object oriented data modeling: *dynamic data semantics*, and *function groups*.

4.1 Semantic Data

Data semantics represents 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 section 2, 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. The system automatically replaces the old object with the new object and adds the old object to a list of old objects representing the stages the data has traversed. Saving past versions of nontextual data is useful for several reasons. As mentioned in section 3, 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). Moreover, because methods in MOODS transform objects of one class into objects of another class, creating a completely new object and leaving the old object unchanged is an appropriate abstraction, especially when the initial class and the resultant class have substantially different structures.

4.2 Function Groups

The second unique feature of MOODS is the ability to define the operations allowed on an object in terms of *function groups*. Each function group represents an abstract or logical function. Figure 1 illustrates the structure of a class definition for two different classes, each using different function groups. A function group is simply a set of related functions. Function groups are globally defined and can be incorporated into any class. Typically a function group will contain 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 a function group consisting of a set of edge detector methods. Some edge detection routines produce identical results but use different algorithms to achieve their results (e.g., one algorithm consumes a lot of memory but runs very quickly, while another algorithm runs very slowly but consumes small amounts of memory). The group may also contain other edge detection routines that do a better or worse job of edge detection, but are nevertheless edge detection methods. 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.

Objects in MOODS are defined in terms of function groups rather than specific functions. That

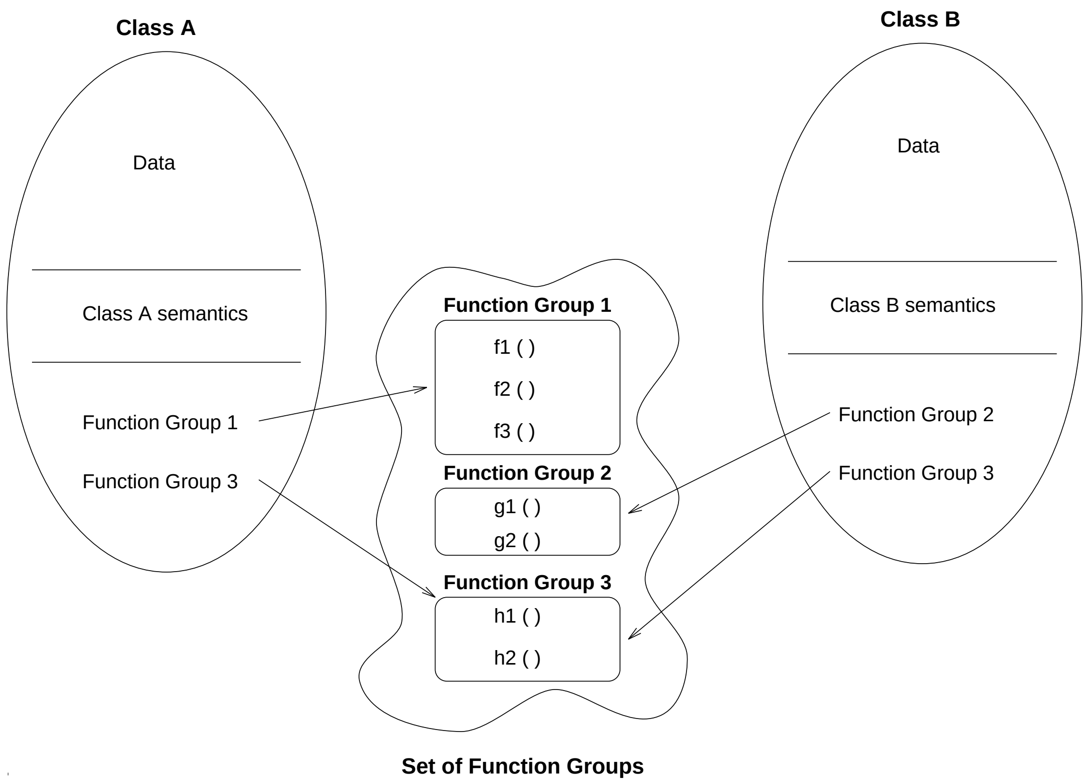

Figure 1: Two class definitions in MOODS. Each class includes function groups selected from a global domain of function groups.

is, a class definition specifies the lists of abstract operations (functions groups) that apply to the class. Binding abstract operations to objects allows us to postpone binding specific functions until the logical operation is required at runtime. As new methods become available they can be dynamically added to a function group effectively changing the set of operations available within a class. In fact, delayed binding provides the ability to alter the set of functions available to instantiated objects. Dynamically changing the functions associated with instances of objects is particularly important given the built-in history mechanism of MOODS via immutable objects.

Another advantage to function groups is the generic use of logical (abstract) operations. Many times a logical operation applies to a wide variety of objects (classes). Function groups allow two or more distinctly different objects to share a logical operation. As a result, enhancements and improvements made to an abstract operation immediately become available to a wide range of object classes.

In many cases, the current semantics of the data provides sufficient information for the system to automatically select a specific function at runtime. If 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 function from a function group can be specified at compile time, load time, or never (in which case a default function is used).

To uniquely identify a particular function from a function group the system needs four pieces of information:

Function Name: a unique name for the function.

Object Data Format: the type of data local to the object in which the function is included (implicit inputs).

Resulting Object Data Format: the type of data local to the resulting object (implicit outputs).

Input/Output Parameter Format: the data types of the explicit input and output parameters.

Although the above information uniquely identifies a function, a subset of this information is typically sufficient to uniquely identify a particular function. The following section describes the process of selecting a particular function from a function group given some subset of the above information.

5 Identifying Functions in a Function Group

Since a class definition in MOODS specifies function groups rather than functions, the immediate question is: How does one invoke an object's methods? For the purposes of this discussion we will ignore passing parameters to functions.

Imagine for a moment that we have a function group containing a set of edge detection functions (e.g., e_1, e_2, ... , e_n). Each function uses a slightly different algorithm to locate the edges in an image. The class definition IMAGECLASS says that it wants to include the set of edge detection functions (assume the set is called E). Given an object called *image* of type IMAGECLASS, we can invoke a particular function using the "full name" of the function. For example, to invoke function e_1 we might write

$$\text{image.E.}e_1()$$

However, we do not require applications to specify the "full name" of a function. Instead, we allow applications to specify function names in an ambiguous fashion. This is particularly useful in an interactive setting. For example, an ambiguous call to an edge detector function might be:

$$\text{image.E}()$$

The systems uses one of four methods to handle ambiguous function invocations: *parameter matching*, *defaults*, *prompting*, and *aborting*. The system tries each of these methods in sequence. First the system tries to resolve the ambiguity using parameter matching. Since each e_x has its own implicit and explicit parameter list, the system may be able to uniquely identify the correct function based on the parameters being passed to the function. However, it may be the case that two functions (e.g., e_i and e_j) take the same types of parameters and the system cannot resolve the ambiguity. To solve this ambiguity, the system allows the application writer to specify a "default" function. If one (and only one) of the ambiguous functions has been labeled as the default function, the ambiguity is resolved. If no default has been specified, or if more than one default has been specified the system either

- prompts the user, asking which function to use, or
- terminates the application.

6 Data References in Generic Functions

The ability to include a function group in multiple objects raises an interesting question: *How does a function identify the data on which it works*? In other words, how does the function access the object's data when each object has its own class-specific data structures.

In order for classes to share functions we must provide a method for writing functions independent of the classes in which they will eventually be included. However, because the classes do not necessarily exist at the time the function is written, the function cannot know the data it must operate on.

MOODS solves this problem with late binding. We bind a classes' data to the data referenced by a function when the function is included in the class (i.e., when the class is written). This binding is done in a way similar to parameter passing which we call *implicit input* labeling [3].

However, late binding only solves half of the data reference problem. Recall from section 4.1, that invoking a function typically changes the class to which an object belongs. In fact, a function invocation really produces an entirely new object (see Figure 2). As a result of the built-in history mechanism and

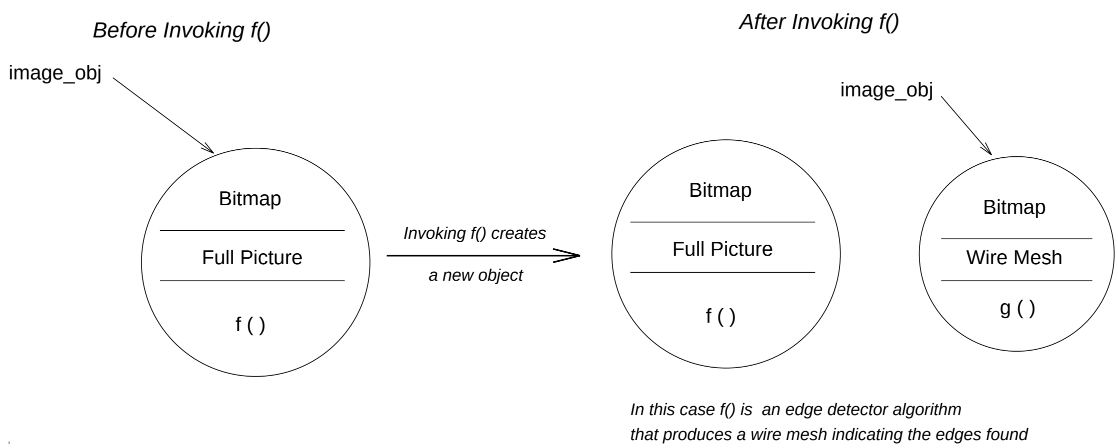

Figure 2: Invoking a function results in a new object possibly belonging to a different class than the original object. Before invoking f(), the variable (object instance) image_obj refers to the object on the left. After invoking f(), image_obj refers to the image on the right.

the use of immutable objects, functions must access (read) data from the current object and modify (write) data in the new object. Consequently, we must also specify the binding between data references in the function and the data in the new object. Figure 3 illustrates this idea. For this purpose, we also use a scheme similar to parameter passing called *implicit output* labeling [3].

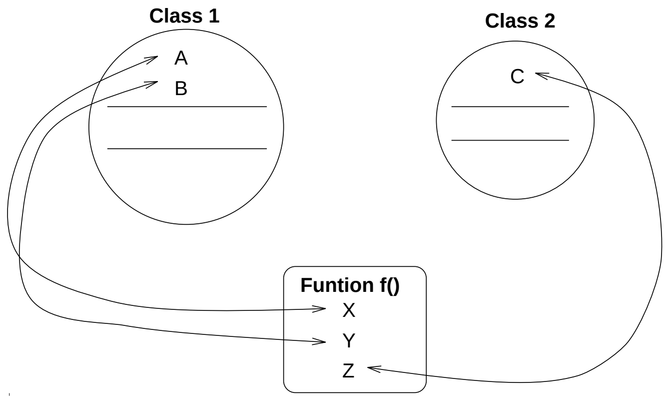

Figure 3: Binding a function to the data in the object to which it belongs and to the data in the object it will create. In the picture, class 1 contains a function f() that (when executed) creates a new object of type class 2. A,B represent private data items in class 1, while C represents a private data item in class 2. A is bound to X, B is bound to Y, and C is bound to Z.

7 An Example System

In this section we present a simple example of the MOODS model-based image information management. Consider an image information system. Raw images are input to the system and then processed to obtain information about the image. A variety of functions are available at each stage of the processing sequence. In particular, the following generic function groups are available for processing and displaying the various information classes.

Image Enhancement: Three simple functions are available which can be used to filter a raw image.

1. Neighborhood Averaging: processes the image using the desired size-averaging filter to produce a corresponding averaged image.
2. Median Filtering: apply the specified median filter to an image to produce a corresponding filtered image.
3. Histogram Modification - modify an image to obtain a new image whose histogram matches the specified histogram.

Region Segmentation: This function group contains functions for finding and marking the various regions within an image.

1. Binary Thresholding: Given an input intensity image, the group of functions produce a binary image by thresholding the image using the specified threshold value.

2. Connected Region Detection: produces the labeled connected region image using the specified threshold values for region growing and merging criteria.

Boundary Detection: This function groups contains various edge detection methods.

1. Sobel Edge Detection: use the *Sobel edge detection* method along with some thresholds for postprocessing to detect thinned and linked boundary segments.
2. Canny Edge Detection: use the *Canny edge detection* method along with some thresholds to detect linked edge points.

Image Display: This function group displays an image on the workstation monitor in the desired format.

1. X Display Image: display an image in an X-window.

Primitive Shape Detection: The functions in this group take boundary maps or region segmented images and apply primitive shape detection methods.

1. Straight Line Detection: detect all straight line segments present in the input boundary image and estimates their equations and positions.
2. Elliptical Boundary Detection: detect all the ellipses present in the input boundary map and estimates their positions and equations.
3. Circular Region Detection: detect all the circular regions present in a region segmented image.
4. Square Region Detection: detect all the square regions present in a regions segmented image.
5. Polygonal Fit: produce the polygonal approximation of contours (detected boundary segments) in a boundary image.

Domain Dependent Labeling This set of functions process the results of the primitive shape detection stage to label detected primitive shapes in terms of domain specific labels. We will not list the specific routines, however, there are three basic classes of techniques: a) boundary map-based Labeling, b) region map-based Labeling, and c) Labeling the composition of region and boundary maps.

Domain Dependent Analysis: This function group consists of functions that draw conclusions, make observations, generate a result, make a diagnosis, construct a hypothesis, etc. Some methods may be automatic, others will be interactive and require user analysis.

Processing the data causes the semantics of the data (image) to 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. To model the changing semantics of the data, the system associates each semantic class with the set of functions that are relevant to the data's current semantic meaning.

Figure 4 illustrates the semantic classes that comprise the image information system. Each circle

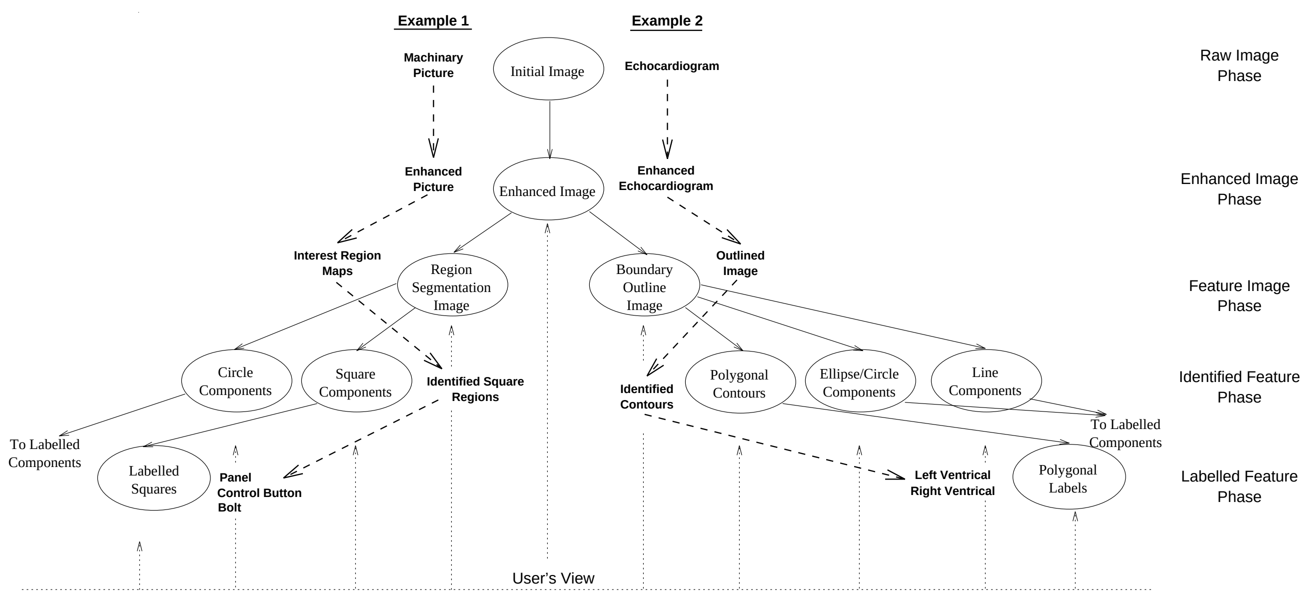

Figure 4: A MOODS image information processing system. Two example application domains, *Industrial Images* and *Medical Imaging*, are shown in bold face. Each user view will be a subset of the dotted lines.

represents a different semantic class with semantic-specific functions. The lines between circle represent the application of a particular function selected from a function group. Note that the structure and format of the data in the two topmost classes, *Initial Image* and *Enhanced Image*, are identical. However, their semantics differ and thus the functions applicable to each class also differ. The specific function groups available to each class are listed in appendix A.

The figure shows two example image processing scenarios which represent two completely different application domains: *Medical Imaging* and *Industrial Image Processing*. The two scenarios demonstrate the flexibility of the system to adapt the processing to each application domain. In the industrial image example, a picture of a piece of machinery is processed to the point where square regions are identified and then labeled as panels, control buttons, or bolts. The medical imaging example shows how an echocardiogram propagates through a series of semantic stages until the left and right ventrical of a heart can be identified. Although additional semantic classes are not shown, the data might be processed further by applying domain dependent functions to analyze the data and construct a diagnosis of the heart's condition/state (e.g., heart attack).

At various steps of the processing sequence, new information is computed and stored in the output class. However, because objects are immutable, the original information is still available to the user in the previous class. In the figure, as we move from top to bottom, a semantic class may have several descendents where each descendent represents a new semantic class that has additional information available. Due to the immutable nature of objects, as a new node fans out to produce one or more new semantic classes, the old nodes (and the information available in them) remain visible to the user. The complete set of information available to users is shown by the dotted lines pointing to potentially visible objects. Not all users need to see the same information. Thus a *user's view* is typically some subset of the potentially visible objects. This allows user-defined views of the information as well as domain-specific views of the information. In the case of the medical imaging example, the user would only see a few of the objects on the righthand side of the tree.

8 Summary

Management and modeling of non-alphanumeric data (such as images, videos, graphs, charts, audios, etc.) has emerged as an important research issue and will continue to play a major role in large information databases as such media forms become commonplace. Modeling and manipulation of these new emerging data forms will be essential to many applications.

In this paper, we have proposed a new data model for manipulating non-alphanumeric data. In particular, we have shown how such a model could be applied to visual data such as image data. The model allows users to direct an image through a series of transformations which change the semantic meaning of the image. At each step in the transformation, the model defines a new set of applicable functions based on the current semantic meaning of the image. We have also shown how abstract operations can be applied dynamically to existing and future data. This ability allows the model to incorporate new techniques and algorithms as they become available and, thus, effectively revitalizes existing data by providing new views of the information inherent in the data. Currently, we are also building a prototype information management system that will allow us to test our ideas in the domain of medical imaging and to refine the model further.

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.

[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, MOODS: A New Object Oriented Model for Data Semantics in Multimedia Information Systems, *Technical Report*, Department of Computer Science, University of Kentucku, April 1993 (in preparation).

[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.

9 Appendix A

```
/***** Data Groups *****/

DG Image {
    unsigned char *pict (image);
    int rows (numRows);
    int cols (numCols);
};

typedef struct {
    float   slope, yIntercept, lineLen;
    int     xCenter, yCenter;
} LineType;


DG LineInfo {
    LineType *lineInfo (lineInfoArray);
    int n (numLines);
    DG Image oldImage;
};

typedef struct {
    int     xCenter, yCenter,
         majorAxis[4], minorAxis[4];
} EllipseType;

DG EllipseInfo {
    EllipseType *ellipseInfo (ellipseInfoArray);
    int n (numEllipses);
    DG Image oldImage;
};

DG CircleInfo {
    int *labels (circleLabelArray);
    int *area (circleAreaArray);
    DG Image oldImage;
};

DG SquareInfo {
    int *labels (squareLabelArray);
    int *area (squareAreaArray);
    DG Image oldImage;
};

DG PolygonInfo {
    int ***contour (polyContourPts);
    DG Image oldImage;
};

DG DomainLabels {
    char **labels (domainLabels);
    int n (numLabels);
};


/***** Function Groups *****/

FG ImageEnhancement {
    NeighborhoodAveraging(image:image:unsigned char *, numRows:r:int,
                       numCols:c:int; image:out:unsigned char *;
                          neighborhoodRows:int, neighborhoodCols:int; );
    MedianFiltering(image:image:unsigned char *, numRows:r:int, numCols:c:int;
              image:out:unsigned char *;
              neighborhoodRows:int, neighborhoodCols:int; );
    HistogramModification(image:image:unsigned char *, numRows:r:int,
```

```
                 numCols:c:int;
                image:out:unsigned char *; ; );
};

FG RegionSegmentation {
    BinaryThresholding(image:image:unsigned char *, numRows:r:int,
                 numCols:c:int;
                 image:out:unsigned char *;
                 threshold:int; );
    ConnectRegionDetection(image:image:unsigned char *, numRows:r:int,
                     numCols:c:int;
                  image:out:unsigned char *;
                  shadeRegionThreshold:int,
                  intensityThreshold:int; );
};

FG BoundaryDetection {
    SobelEdgeDetection(image:image:unsigned char *, numRows:r:int,
                 numCols:c:int;
                 image:out:unsigned char *;
                 gradientThreshold:int; );
    CannyEdgeDetection(image:image:unsigned char *, numRows:r:int,
                 numCols:c:int;
                 image:out:unsigned char *;
                 highThreshold:int, lowThreshold:int; );
};

FG Display {
    Display(image:image:unsigned char *, numRows:r:int, numCols:c:int;
         ;
         xPos:int, yPos:int, posOrNeg:bool; );
};

FG ShapeDetection {
    StraightLine(image:image:unsigned char *, numRows:r:int, numCols:c:int;
           lineInfoArray:lines:LineType *, numLines:n:int; ; );
    EllipseBoundary(image:image:unsigned char *, numRows:r:int, numCols:c:int;
                ellipseInfoArray:ellipses:EllipseType *, numEllipses:int;
              ; );
    CircleRegion(image:image:unsigned char *, numRows:r:int,
           numCols:c:int;
           circleLabelArray:labels:int *, circleAreaArray:area:int *;
           ; );
    SquareRegion(image:image:unsigned char *, numRows:r:int,
              numCols:c:int;
           squareLabelArray:labels:int *,
              squareAreaArray:area:int *;
           ; );
    PolygonalFit(image:image:unsigned char *, numRows:r:int,
           numCols:c:int;
           polyContourPts:pts:int ***; ; );
};

FG DomainLabelling {
    BinaryImageLabelling(image:image:unsigned char *, numRows:r:int,
                numCols:c:int;
                domainLabels:labels:char **, numLabels:n:int;
                ; );
    ConnectedImageLabelling(image:image:unsigned char *, numRows:r:int,
                   numCols:c:int;
                   domainLabels:labels:char **, numLabels:n:int;
                   ; );
    BoundaryImageLabelling(image:image:unsigned char *, numRows:r:int,
                  numCols:c:int;
                  domainLabels:labels:char **, numLabels:n:int;
                  ; );
```

```
    LineLabelling(image:image:unsigned char *, numRows:r:int,
            numCols:c:int, lineInfoArray:lines:LineType *,
            numLines:n:int;
            domainLabels:labels:char **, numLabels:n:int;
            ; );
    EllipseLabelling(image:image:unsigned char *, numRows:r:int,
               numCols:c:int, ellipseInfoArray:ellipses:EllipseType *,
               numEllipses:n:int;
               domainLabels:labels:char **, numLabels:n:int;
               ; );
    CircleLabelling(image:image:unsigned char *, numRows:r:int,
              numCols:c:int, circleLabelArray:circles:int *,
              circleAreaArray:area:int *;
              domainLabels:labels:char **, numLabels:n:int;
              ; );

    SquareLabelling(image:image:unsigned char *, numRows:r:int,
              numCols:c:int, squareLabelArray:squares:int *,
              squareAreaArray:area:int *;
              domainLabels:labels:char **, numLabels:n:int;
              ; );
    PolygonLabelling(image:image:unsigned char *, numRows:r:int,
              numCols:c:int, polyContourPts:poly:int ***;
              domainLabels:labels:char **, numLabels:n:int;
              ; );
};


/***** Classes *****/

CLASS RawImage {
    DG Image rawImage;

    FG ImageEnhancement(rawImage; EnhancedImage, enhancedImage);
    FG Display(rawImage; RawImage, rawImage);
};

CLASS EnhancedImage {
    DG Image enhancedImage;

    FG RegionSegmentation(enhancedImage; Binarymage, binaryImage);
    FG RegionSegmentation(enhancedImage; ConnectedImage, connectedImage);
    FG BoundaryDetection(enhancedImage; BoundaryPict, boundaryImage);
    FG Display(enhancedImage; EnhancedImage, enhancedImage);
};

CLASS BinaryImage {
    DG Image binaryImage;

    FG ShapeDetection(binaryImage; CircleImage, image);
    FG ShapeDetection(binaryImage; SquareImage, image);
    FG Display(binaryImage; BinaryImage, binaryImage);
};

CLASS ConnectedImage {
    DG Image connectedImage;

    FG ShapeDetection(connectedImage; CircleImage, image);
    FG ShapeDetection(connectedImage; SquareImage, image);
    FG Display(connectedImage; ConnectedImage, connectedImage);
};

CLASS BoundaryPict {
    DG Image boundaryImage;

    FG ShapeDetection(boundaryImage; LineImage, image);
```

```
    FG ShapeDetection(boundaryImage; EllipseImage, image);
    FG ShapeDetection(boundaryImage; PolygonImage, image);
    FG Display(boundaryImage; BoundaryPict, boundaryImage);
};

CLASS LineImage {
    DG LineInfo image;

    FG DomainLabelling(image; DomainLabels, labels);
};

CLASS EllipseImage {
    DG EllipseInfo image;

    FG DomainLabelling(image; DomainLabels, labels);
};

CLASS CircleImage {
    DG CircleInfo image;

    FG DomainLabelling(image; DomainLabels, labels);
};

CLASS SquareImage {
    DG SquareInfo image;

    FG DomainLabelling(image; DomainLabels, labels);
};

CLASS PolygonImage {
    DG PolygonInfo image;

    FG DomainLabelling(image; DomainLabels, labels);
};

CLASS DomainLabels {
    DG DomainLabels labels;
};
```