Abstract. A multi-layer integrated data model is presented that unifies digitized spatial data and related lexical information. The user level at the top allows customization of the system for specific applications. At the next layer (the conceptual model), generic spatial and lexical objects are classified into object sets organized in a semantic data model. In the third layer (the continuous database model), object sets and relationships among objects are represented as relations but, in contradistinction to the customary relational database convention, relations with an infinite number of tuples are allowed. For spatial data such as images, these infinite relations correspond to a continuous view of the world. Thus, at this layer and above users need not be concerned about sampling and quantization schemes used to store and manipulate images under limitations imposed by finite machines. The infinite relations are translated into a discrete database model (the fourth layer). Abstract operations can be developed for each of these four layers. Of particular interest here are operations at the third and fourth layers where common spatial operations are expressed in relational algebra extended to facilitate the combination of spatial and non-spatial queries. The actual data operations are performed at the level of the computational model (the fifth layer), where efficient, specialized data structures and algorithms can be applied to spatial and lexical objects. Data representation on physical mass-storage media is modeled in the last layer. An example is provided to illustrate the use of both spatial data and lexical data in a single query. A high-level compilation of this sample query provides an example of a suggested framework for optimization.

Keywords. Data models -- multi-layered, integrated; spatial query languages; image database systems.

1. INTRODUCTION

Imagine a system that combines most of the capabilities of MacDraw, MacPaint, DBase III, Lotus 1-2-3, MS-Word, Hypercard, and Framemaker for manipulating images, line drawings, and lexical data. Imagine also that it exhibits a uniform but customizable user interface, runs fast enough for interactive access, and can access vast quantities of heterogeneous data. The system could be configured, for example, for a radiology department with thousands of X-rays and patient records, for a security agency with photographs, fingerprints, wiretap recordings, and criminal records, for a real estate agency with photographs of properties, plat maps, and agent networks, as a database for computer integrated manufacturing, or as a geographic information system.

Development of a system with these capabilities requires a solid, sufficiently-high-level platform on which to build. Following the lead provided by the development of other large information-handling systems, such as multi-tasking operating systems and the ISO Open Systems network model, we are developing in this paper a layered architecture to cope with the inherent complexity of retrieving and manipulating a collection of heterogeneous objects. The model consists of the following layers:

1. user model
2. conceptual model
3. continuous database model
4. discrete database model
5. computational model
6. mass storage model

In the remainder of this section we briefly describe each layer. The next section contains pointers to some of the relevant literature. Section 3 offers a description of the conceptual and database layers, to which we have devoted most of our attention so far. Section 4 gives some details on possible methods of implementing and interfacing these layers, and starting points for optimization. Our illustrative examples are all drawn from the realm of geographical information systems.

As do models for all information-handling systems, our top-level user model contains application-oriented entities such as satellite images of towns, town boundaries, and town characteristics. It also has an application-specific interface which can be a declarative or procedural application language, or a set of graphic metaphors. This layer may itself consist of several sub-layers: an applications programmer may customize the interface for an individual or group with specific interests.

The conceptual model contains generic object sets such as image segments, closed curves, spatial locations (points), and lists or hierarchies of characteristics. Each object set has a corresponding set of operators appropriate for its domain. Objects may also inherit attributes, have properties, and be subject to constraints.
The continuous database model hides within it the inherent complexity of the discrete representation of continuous spatial and temporal objects such as images, curves, and time signals. Hidden aspects include spatial and temporal sampling, amplitude quantization, choice of view point, and framing characteristics. At this level, continuous objects are represented as infinite sets of tuples. For instance, for an image we postulate a relation that contains an infinite number of tuples (one x-y-tuple for every pair of real numbers x and y). Discrete objects in the continuous database model are, of course, represented as finite sets of tuples just as they are in the discrete database model. The operations for the continuous database model include relation-returning functions as well as the conventional relational operators.

The discrete database contains discrete relations manipulated by conventional relational operators. Since there is no unique correspondence between a continuous object such as a curve and discretized versions of the same object, several discrete relations at this level may correspond to an infinite relation in the continuous model.

Because it would be inefficient to implement relational operators directly on images, curves, or regions represented as tuples, the computational model is the level at which algorithms are actually specified for manipulating objects. Images, for example, may be stored and manipulated as arrays or quad trees, while curves may be represented as linked lists of points. The computational model allows the application of efficient algorithms developed in image processing, pattern recognition, computer graphics, and computational geometry.

Finally, the mass storage model represents the physical database consisting of file structures of blocked records of compressed or uncompressed data. This layer contains, as expected, the only device dependent modules except for the input/output mechanisms of the application layer.

The principal advantage of our approach is the direct correspondence between the user view and reality and the integration of spatial objects, auxiliary information about spatial objects, and application information in one vertically-layered database model instead of in several "horizontal" divisions according to data type. Conventional application data and auxiliary information about spatial objects as well as images, regions, and curves can be viewed uniformly as relations, and thus conventional relational operators can be applied uniformly to all database objects. Considering all data as relations, however, imposes a low-level view on the data. To allow for high-level views, we provide both a conceptual view and any number of application-oriented views where objects can be classified into object sets and specialized by adding additional operations and user-oriented conventions. We are thus able to provide both a unified and a high-level view for data processing. This is offset by the burden placed on the system to manage and control a diversity of objects with different properties and by increased requirements for optimization.

2. RELATED RESEARCH

The potential contribution of conventional database techniques to image processing has spurred an increased interest in the development of image database systems. As Tamura and Yokoya state in their excellent survey, however, their signals and their different background in image database design, and it does not yet seem that the significant concepts of image database systems have been established [29]. Since this survey appeared, a few articles have been published that begin to establish a foundation for image database systems [16,24], but much remains to be done.

Our own impression is that much of the work appearing under the heading "image database" describes either image non-database systems or non-image database systems, or deals with some peripheral aspect of database systems. Image non-database systems such as [12,13,28] consist of software designed to manipulate sizable collections of image data without recourse to most of the important database concepts. Non-image database systems such as [11,17,20] use a database system for image descriptors, but do not contain the images themselves. The peripheral aspects addressed include image compression [26], intelligent image semantics [21], graphics (vector) databases [32], internal data structures for two-dimensional entities [19,31], and the retrieval of image attributes or of automatically extracted descriptors [2,3,30].

Interfaces and languages for image processing have been studied. Some are becoming commercially available, for example, ImageLab (Werner Frei Association), PCPI100 (Trowbridge Industries), LIPS (Gould Inc.), A2-ASP (Dipix Inc.), RAIL (Automatix), and P256 (International Robotics Intelligence). Discussion in the research literature of proposed interfaces includes [1,5,7,23,25]. Some of our earlier work including a description of an implementation of some of our initial ideas in the framework of EQUEL and INGRES is discussed in [9].

For some summary comments about image database systems, see [22]. The field appears ready for the application of some unifying concepts that will allow the whole to emerge from the details. We see the whole as a multi-layer, integrated data model that unifies the definition and manipulation of objects and presents a high-level platform on which application-oriented interfaces can be built.

3. CONCEPTUAL DATA MODEL

We call our conceptual data model the LSO data model to emphasize that we are interested in both lexical and spatial data and that we view both types of data uniformly as Objects. A database for our LSO model is a collection of objects grouped into object sets, a collection of relationships among the objects grouped into relationship sets, and a collection of applicable operations associated with each object set.

Our LSO conceptual data model is based on the Entity-Relationship (ER) model [4] and its extensions [8,10,14]. Our LSO model differs from these because it treats attributes as a point of view rather than a basic building block and because it allows a rich set of object-oriented operations to be associated with each object set.

Figure 1 shows an LSO-model diagram, which we use throughout the paper to illustrate various concepts. In Figure 1, non-lexical object sets are represented by object-set names enclosed in rectangular boxes, and lexical object sets are represented by object-set names alone. For example, the non-lexical object set Image and the lexical object set Population and Intensity - Value represent respectively the set of population and intensity values.

Basic relationship sets in an LSO model are represented by a variety of arcs and hyper-arcs. Relationships between pairs of object sets may be: one-one,
Figure 1. Conceptual-Model Diagram.

represented by arrow-heads at both ends of a connecting arc as for Image and Image_ID; one-many, represented by an arrow pointing functionally from the domain object set to the codomain object set as for Town and Avg_Income; or many-many, represented by the absence of arrow heads. These relationship-set types may also be named by placing a label in a diamond on the connecting arc as is illustrated by the Obtained_By label which names the many-one relationship from Image to Satellite. N-ary relationship sets are also possible, but are not illustrated in our example.

A generalization/specialization relationship between a pair of object sets is represented by placing a subset symbol (C) on an arc with the open end toward the generalization side. For example, every Town_Area is a Region and thus Town_Area is a specialization of Region.

An aggregation/decomposition relationship is denoted by enclosing component object sets in a named rectangular box. When an aggregation relationship is established, a new higher-level object set is also defined. Point, for example, is an aggregation of X values and Y values, and Image Point is an aggregation of the object sets Intensity_Value and Point.

Power set associations are represented by an arc with a star (*) on the power-set side. Thus, for example, each object in the Image object set is an element of the power set of image points and is thus conceptually a set of point/intensity-value pairs.

Object sets are described by object-set descriptors, defined as follows.

Object-Set: <name>
Domain: { <string list> } ISA <object-set name>.
Properties: <constraint/property list>
Operations: <function-definition list>
End

Each object set has a unique name that identifies it. The domain of an object set can be defined in four ways: (1) by giving the elements of the domain as a set of lexical strings, (2) by stating that it "ISA" some other object set (which may either be built-in such as Real and Integer or defined in the LSO model), (3) by listing its subcomponents when it is an aggregation, or (4) by designating an object set as the base set for an object set containing elements of a power set. Additional properties of the object set may be given to further constrain it. Finally, applicable operations may be defined for the object set.
A sample object-set descriptor for Image follows.

Object-Set: Image
Domain: <Image_Point>
Properties:
1. FUNCTION: Point \(\rightarrow\) Intensity_Value
2. CONTINUOUS: Point
3. CONTINUOUS: Intensity_Value
4. \(0 \leq \text{Intensity_Value} \leq 1\)

Operations:
Normalization_Constant_Addition
(Image, Intensity_Value) \(\rightarrow\) Image
Normalization_Algebraic_Addition
(Image, Image) \(\rightarrow\) Image

End

An image is defined as a power set of image points. It is further constrained to be a continuous function from a plane of points to a set of intensity values between zero and one. Two of the many possible operations on images are shown.

Relationship-set descriptors for generalization, aggregation, and power-set relationships are subsumed by object-set descriptors. One-one, one-many, and many-many relationship sets may be described individually or they may be collected together in groups that are normalized in the relational database sense. A (grouped) relationship-set descriptor is defined as follows.

Relationship-Set: <name>
Scheme: <attribute/domain-name list>
Key: <list of attribute or domain names>
End

Each relationship set has a unique name that identifies it. The scheme for the relationship set is a list of object-set names, optionally prefixed with an attribute name which usually designates the role of the object set in the relationship set. Keys for the relationship set are specified by listing sets of attribute or object-set names. Examples of relationship sets for town descriptors and image descriptors follow.

Relationship-Set: Town_Descriptor
Scheme:
Town
Town_ID
Town_Name
County
Population
Avg_Income
Town_Area
Town_Boundary
Town_Center: Point
Key:
Town
Town_ID
Town_Name, County
End

Relationship-Set: Image_Descriptor
Scheme:
Image
Image_ID
Satellite_ID
Time
Key:
Image
Image_ID
Satellite_ID, Time
End

From the perspective of the underlying database model, each object set and each relationship set represents a relation. The name for these relations is the name of the object or relationship-set descriptor. For an aggregation the relation scheme is the set of object-set names of its subcomponents. For all other types of object sets the relation represented is a single-attribute relation and thus the scheme is the object set name. For relationship-set relations the scheme is directly given in the relationship-set descriptor.

These relations are not necessarily in first normal form. A tuple in the Image_Descriptor relation above, for example, conceptually contains an Image, which is itself a relation. The relation Image contains a collection of elements each of which is a relation. Since Image is constructed as a power set of Image_Point, the scheme for each of these relations is the scheme for the aggregation Image_Point.

At the level of the conceptual database model and above, some of the objects may be considered to be continuous. Images, regions, curves, town areas, and town boundaries in Figure 1 are all considered to be continuous. In the conceptual database model these objects are thought of and operated on as if they have an infinite number of tuples. Thus, for example, when a region and an image are joined, the user need not be concerned about whether the discrete points representing the region and the image plane are the same.

At the level of the conceptual database model all the relational algebra operations may be applied. In this paper we follow the notation in [18] and use the operators \(\pi, \sigma, \cup, \setminus, \delta, \) and \(\cap\).

We also use a less-well-known computation operator \(\chi [6]\). The \(\chi\) operator as the form \(\chi A \cdot f(A)\) and computes values for a new attribute \(A\) for \(r\) according to formula \(f\), optionally partitioned by values of tuples in \(X\). For example,

\[
\pi_{\text{Total_Pop}} \chi_{\text{Total_Pop}} \cdot \sum_{\text{Population}} \text{Town_Descriptor}
\]
computes the total population for all towns represented in relation Town_Descriptor.

Since our relations are not necessarily in first normal form, we also define an unnest operator \(\nu\) [27]. The unnest operator has the form \(\nu r\) where \(A\) is an attribute in the scheme of relation \(r\) whose type allows sets of relations in its domain. Let \(R\) be the scheme of relation \(r\) and let \(S\) be the scheme of a relation in the domain of \(A\). We require \(R \cap S = \emptyset\). The effect of the \(\nu\) operator is to replace each tuple \(t\) of \(r\) by the set of tuples \(\{t(R - A)\} \cup s\) where \(s\) is the relation instance in \(t(A)\) and to replace the relation scheme \(R\) of \(r\) by \((R - A)S\). For example, \(\nu_{\text{Image_Descriptor}}\) is a relation whose scheme is \((\text{Intensity_Value}, \text{Point}, \text{Image_ID}, \text{Satellite_ID}, \text{Time})\) and whose tuples are (logically) the union of tuples of each image in Image, each having its image identifier and other attributes attached.

A user may write queries for the LSO model at the level of the conceptual database by invoking relational operators and by invoking defined operators on objects in the database. Suppose, for example, that there is an image of Utah county in the state of Utah whose Image_ID is 104773 and that there is a Town_Area region named Provo which represents the region in Utah county of the town of Provo whose Town_ID is 70005. An image of Provo could be created by invoking the query

\[
\pi_{\text{Point}, \text{Intensity_Value}} \cdot \chi_{\text{Image_ID} = 104773} \cdot \text{Image_Descriptor} \cup \nu_{\text{Point}} \cdot \text{Town_Area} \cdot \text{Town_ID} = 70005 \text{ Town_Descriptor}
\]
As a more-extensive example that involves both lexical and spatial data and illustrates the power of using high-level abstractions, consider the request:

For the town with the largest income in Jefferson, Iron, and Cheyenne Counties, create an image of a twenty-kilometer square area with the town at its center. Make a mosaic using the most recent images that cover the area.

We assume that we have images that completely cover the areas of interest, and we assume that the standard unit of measure is a kilometer and that the time value is different for every image. To shorten the example, we are also making some simplifying assumptions about mosaic creation [33]. The result is shown in Figure 2.

In the first part of the query in Figure 2, we find the location of the town we want. We do this by first limiting the Town_Descriptor relation to only towns in Jefferson, Iron, and Cheyenne counties. We then compute the income for each of these towns and find the highest income. Using the highest income, we select the town with the largest (total income and project on the town center of this town. We are assuming that there are not two towns with the same largest income. Without this assumption, we can make an explicit check and resort to a secondary selection criteria, if necessary.)

In the second part of the query, we create a twenty-kilometer square region with the highest-income town at its center. We first unnest the Town_Descriptor relation to obtain the X and Y components of the location. We then invoke an operator to create a rectangular region given its four corner points. It is assumed that this operator is defined in the Region object set. It is important to observe that the twenty-kilometer square region is logically an infinite relation having a tuple for each point in the region.
4.1. Object Representation

The option of having multiple representations is particularly important for spatial objects. Spatial objects with regular features can be represented succinctly, often with only a few data values, whereas the same type of spatial object with irregular features may require a massive amount of data.

Several techniques, for example, may be used to store conceptually-infinite regions. If a region is rectangular and oriented with its sides parallel to the axes of the standard coordinate frame, it is possible to store the region as four real numbers representing the lower-left and upper-right corners of the region. If a region is not rectangular, but has a boundary that is a closed curve, then the region can be represented by a sequence of vectors with an assumed convention of either clockwise or counterclockwise boundary traversal. If a region is a collection of several regions whose boundaries are closed curves, then the region can be represented by a set of vector sequences. Any region can also be represented by a grid sampling of \( z - y \) pairs.

Any (conceptually-infinite) image can be represented as a discrete image with an interpolation algorithm for determining intermediate points. Either a default or a user-provided interpolation algorithm can be used. The most common storage technique for a discrete image is an array of intensity values. Various compression techniques such as block coding and transform coding can be used to save space.

When there are several alternate representations, all can be used. A particular object would (probably) have only one representation at any time, but may be converted to any of the alternate representations for processing. Objects in an object class need not all have the same representation.

4.2. Query Optimization

Database query optimization has been extensively studied. Much of what is known has been surveyed in an article by Jarke and Koch [15]. The application of database query optimization to higher-level data models that include complex objects, however, is relatively unexplored.

In this subsection we make five observations about our LSO implementation model. These observations constitute a basic framework for LSO query optimization. We then explain how this optimization framework can be used to optimize the query in Figure 2.

#1: Algebraic Equivalence for Lexical Objects. By appropriately restricting the types of operations allowed, LSO-model queries can be transformed into a single algebraic expression. The query in Figure 2, for example, can be written as a single expression by substituting right-hand-side expressions for appearances of left-hand-side temporary names. This provides a basis for high-level query optimization through the laws of expression equivalence. Since many of the operators are relational algebra operators, established expression-optimizing strategies such as size reduction by performing selection and projection as early as possible, expression simplification, and recognition of common subexpressions can be applied to transform an expression to an equivalent, more-optimal expansion.

#2: Algebraic Equivalence for Spatial Objects. Laws of expression equivalence can also be applied to image-oriented operations. We can often replace more-expensive operations by less-expensive operations. For example, an operation that selects a subpart of each image in a set of images can be replaced by an operation on stored external information for each image to determine if the subpart would be empty followed, if nonempty, by an operation that selects the subpart. This elimination of inapplicable images based on stored image information can result in enormous savings. Even when the entire image cannot be ignored, we can often translate subpart selection into code that need not access the entire image.

#3: Operation Order for Data Characteristics. Translation to high-level code can follow much of the work that has been done for optimal query translation for database systems. In addition, there are several strategies that are particularly useful for image processing. For example, we can order a sequence of region intersections so that the smallest is done first and a sequence of region unions so that the largest is done first. We can also sort images to be considered by given preferences stored in descriptors such as most recent in time or least cloud cover, and we can rearrange operations to access the fewest possible points of images and to reduce reprocessing of points.

#4: Choice of Internal Representation. As high-level code is transformed into intermediate code, additional optimization can take place. As explained above, objects in an object class can have several different internal representations. With each representation, space and time requirements for each operation and for conversion to and from alternative representations is provided. Thus, the cost of various alternatives can be estimated and the best representation can be chosen.

#5: Global Optimization for Downstream Operations. The most common internal representation for an image (both finite and infinite) is an array of intensity values. An important optimization consideration for this representation is to use the lowest resolution consistent with operations downstream. Another important consideration is to align grid points with the points on the final output grid. Both of these transformations can be done as the points of an image are initially accessed.

These observations provide a framework for an optimization strategy. As an example of how this optimization strategy can be applied, we explain how the query in Figure 2 can be optimized.

The first part of the query involves neither regions nor images. It can thus be optimized by forming a relational algebra expression and optimizing it (Observation #1) and then by translating it optimally into high-level code (Observation #3). A possible result is shown in the following code, which accesses each tuple in Town_Descriptor only once in finding the location of the highest-income town.

```sql
Maz_income := 0
for each tuple t in Town_Descriptor
if t(County) = 'Jefferson' or t(County) = 'Iron'
or if t(County) = 'Cheyenne' then
if (t(Population)*t(Avg_income)) > Maz_income then
Maz_income := t(Town-Center)
Maz_income := t(Population)*t(Avg_income)
```

In the second part of the query, the computations need no optimization. The region can be stored as a single-tuple relation whose four components represent the lower-left and upper-right corners of the rectangle in the standard coordinate frame (Observation #4).
Without optimization the third part of the query would not execute in reasonable time. The first statement in the third part logically retains points in images that intersect with the twenty-kilometer square region. Thus, images that do not intersect need not be considered. We assume that the area covered by an image is part of the Image_Descriptor relation in an attribute called Image_Area available at the implementation level (although not necessarily at the logical level since it can be easily obtained logically by projecting on the Point attribute of an image). We are also assuming that the images are rectangular and that an Image_Area can thus be represented by a few discrete values. Since the intersection can be determined from the descriptor information alone, the images need not be accessed (Observation #3). Hence, the statement can be improved by first executing the following code.

```c
for each tuple t in Image_Descriptor if (t(Image_Area) \ intersect twentyK_region) \#0 then point_and_time_info := point_and_time_info \ U \{t(Image_ID, Time, Image_Area)\}
```

Note that point_and_time_info in this optimized code is a set of selected images denoted by their Image_ID along with their Time value and Image_Area, not a set of image-point tuples.

The second statement in the third part requests the maximum time for each point. Thus, we sort the image descriptors in point_and_time_info with the most-recent time first (Observation #3).

```c
point_and_time_info := sort_descending(point_and_time_info, Time)
```

With point_and_time_info sorted, we can get the result by taking each image in time order and using the part of the image that covers any portion of the twenty-kilometer region that has not yet been covered (Observation #3). We can stop processing as soon as all the twenty-kilometer region has been covered. When part of the region has been covered, it may be the case that the next image to be considered does not intersect with any uncovered part of the twenty-kilometer region and thus need not be considered (Observation #2). The result is to be a raster image for display, but need not be sampled nor quantized to any finer degree than is required by the display operator (Observation #5). In our example default sampling and quantization are used, and thus this information is known. The following code reflects these improvements.

```c
region_to_cover := twentyK_region
result_image := create_black_discrete_image(20, 20) /* initializes result_image to be a 20 by 20 kilometer grid with default sampling intervals */
for each tuple t in point_and_time_info
intersection_region :=
\{t(Image_Area) \ intersect region_to_cover\}
if intersection_region \# 0 then
image := get_image(t(Image_ID))
for each result_image point p in intersection_region
z := get_intensity_value(p, image)
result_image := replace_intensity_value(result_image, p, z)
region_to_cover := region_to_cover \ intersect intersection_region
if region_to_cover \# 0 then exit
display(result_image)
```

A prototype for our LSO data model has been implemented on a Vax at Brigham Young University. It runs under the UNIX operating system and uses the programming languages C and EQUEL and the INGRES database system. The prototype is implemented as an interpreter. It may be invoked either to operate on a previously prepared file of statements or to execute statements interactively. Statements are of two types: (1) operations listed in an object class descriptor and (2) INGRES language statements. The interpreter translates the statements to EQUEL and then compiles and executes them.

5. CONCLUSION

The multi layer architecture that we have presented is designed to accommodate the integration of heterogeneous objects and to provide a high-level platform on which user applications can be built. The key to our approach is vertical integration of layers of decreasing levels of abstraction in considering objects, relationships among objects, and operations on objects. There is no major significance to our choice of exactly six layers. Depending on one's focus of attention, it may be convenient to merge or subdivide layers. Only the topmost and bottommost layers are anchored to the requirements of the application and of the resident source of the actual data. The role of the intermediate layers is simply to provide the conceptual isolation necessary to subdivide the software into manageable modules with clearly defined functionality and interfaces. When the application changes, changes will propagate from the top; when the machine changes, changes will propagate from the bottom. In either case, we expect that most of the intermediate layers will remain unaffected.

At the conceptual level, we have defined the LSO data model which combines the advantages of object-oriented and relational approaches. We have shown how to process queries based on several different types of objects, including images, regions, and lexical data. We also presented an implementation model for LSO and discussed several strategies for query optimization, including methods developed for query optimization in traditional relational database systems, of shortcuts generally used in image processing, and of algorithms for data structures, and of the elementary information-theoretic aspects of disparate sampled and quantized signals. Our computational model preserves full freedom in choosing the most effective approach taking into consideration the characteristics of the data itself, of the desired operations, and of the host environment.

Although we have taken some steps toward creating a high-level, integrated data model for lexical and spatial data processing, much remains to be done. Research topics we are currently pursuing include the development of operations that allow visualization, creation, and manipulation at the conceptual-model level and the specification of algebra and complete minimal sets of operations or, alternatively, convenient sets of operations for various types of geographic information systems.

References