Journal of Visual Languages and Computing (1996) **7**, 361–375

A Visual Database Environment for Scientific Research

REX M. JAKOBOVITS, LARA M. LEWIS, JAMES P. AHRENS, LINDA G. SHAPIRO, STEVEN L. TANIMOTO AND JAMES F. BRINKLEY

Department of Computer Science and Engineering, University of Washington, Seattle, Washington 98195 U.S.A.

Received 1 April 1996; revised 25 July 1996; accepted 13 August 1996

This paper describes a visual database environment designed to be used for scientific research in the imaging sciences. It provides hierarchical relational structures that allow the user to model data as entities possessing properties, parts and relationships, and it supports multi-level queries on these structures. A schema constructor interface allows users to define for each structure, not only its components, but also its visualization, which is built from its components using graphical primitives. Finally, an experiment management subsystem allows users to construct and run computational experiments that apply imaging operators to data from the database. The experiment management system keeps track of the experimental procedures developed by the user and the results generated by executing these procedures.
© 1996 Academic Press Limited

1. Introduction

EXPERIMENTATION with image-related data often involves a number of disjoint applications communicating via flat files, each with its own internal data representation. This approach leads to extraneous processing, as data is marshalled in and out of the various structures and of the file system. Furthermore, it tends to result in an unwieldy collection of cryptic disk files that the researcher must manage, making it difficult to browse and correlate the intermediate data.

The Database Environment for Vision Research (DEVR) is an entity-oriented scientific database system designed to facilitate experimentation with image-related data. It provides a framework in which computer vision researchers may structure their internal data to promote interoperability between applications. DEVR frees the researcher from having to manage data at the file system level, and it enables the user to formulate sophisticated queries across all aspects of the experimental process.

DEVR offers a dynamic data definition language for modeling vision data. It includes an application programmer's interface, which allows users to integrate the database with existing image processing and vision applications. DEVR's query processor supports a wide range of multi-level queries on complex user-defined types. A schema constructor interface allows users to define for each structure, not only its

This work was supported by the National Science Foundation under grant No. IRI-9116809 and by NASA/CESDIS under subcontract No. 555-21.

1045-926X/96/040361 + 15 $25.00/0
© 1996 Academic Press Limited

components, but also its visualization, which can be built from its components using graphical primitives. Finally, an experiment management subsystem allows users to construct and run computational experiments that apply imaging operators to data from the database. A prototype DEVR system was implemented on top of the Object Database Environment (ODE) [1].

2. Related Work

Work on pictorial information systems has been going on since the late 1970s. For an introduction, see the text by S. K. Chang [6]. A relatively recent system of his is discussed in S. K. Chang *et al.* [5]. Two other researchers, N. S. Chang and Fu [7] were also early players who developed a pictorial query language. Later, Brolio *et al.* [3] built a system called ISR (Intermediate Symbolic Representation) that interfaces to the symbolic structures used in their (VISIONS) vision system. At the same time, Goodman [11] developed a persistent object store for Lisp that provided integrated programming language and data management support for development of knowledge-based vision systems. A general visual information system, VIMSYS (Visual Information Management System), was developed by Gupta *et al.* [12].

Some of the recent work on pictorial databases has been targeted at particular scientific applications. In the biomedical area, Cardenas and Chu at UCLA [16] [8] have developed the KMED (Knowledge-based Multimedia Medical Distributed Database System) to help manage general medical research projects. In the area of Earth Sciences, Katz and Stonebraker [18] developed an information system to efficiently manage global change data. Hachem *et al.* [13] and Smith *et al.* [29] have both developed scientific database management systems for managing GIS data.

A number of image database systems that concentrate on retrieval of images by content have been developed. In general systems, Kato [14] [17] has developed an experimental database system called ARTMUSEUM that is intended to be an electronic art gallery. Niblack's research group at IBM Almaden has developed a general-purpose image database system called QBIC (Query by Image Content) [20] that has become a commercial product. QBIC allows retrieval of images by color, texture, and the shape of image objects or regions. Pentland's group at MIT developed the Photobook system [21], a set of interactive tools for browsing and searching images and image sequences. The philosophy behind Photobook is to use several different semantics-preserving representations for images and to provide the user with retrieval tools based on these representations, rather than trying to provide a single representation and matching procedure. In addition to these general systems, a number of specific kinds of image matching have been developed [9, 15, 19, 22].

DEVR is a general scientific database system motivated by the needs of researchers in the imaging sciences. It is closest to the work of Hachem *et al.* [13] and that of Smith *et al.* [29] in trying to provide a full information system, rather than a 'query-by-content' facility. It differs from both of these in providing a simple, but general, model in which scientific users, particularly those in the imaging sciences, can express their data. We now define the data model and give examples of its use in an image analysis application.

3. The HRS Model

Almost every imaging system uses a different format for its data. There are several major image formats and countless higher-level data structures used in imaging. An important question in our work is how to structure this data in order to simplify the work of the researcher and to promote a degree of interoperability of software for different groups. The relational model has been very popular in business database systems, but has fallen short in meeting the needs of scientific researchers. The newer object-oriented systems are much more flexible, but what they provide is so general that structuring data is still a programming art. We have designed a system that lies somewhere between the two, an entity-oriented, hierarchical, relational database system. The building block of the system is the *hierarchical relational structure* (HRS) which comes from the relational data structure of Shapiro and Haralick [25] that was designed for use in a spatial information system and extended for use in relational matching algorithms [4].

In the HRS model, every entity in the system (images, regions, edges, etc.) is described by a schema consisting of three components: *properties, parts,* and *relations.* The properties component of a schema is a table of attribute definitions; each entry specifies an attribute label and declares its type. Properties may be either atomic (i.e. float, integer, string, etc.), or complex types (i.e. full HRS entities). Instances of the HRS defined by a schema will have attribute *values* that correspond to the entries in the schema's property table. These values record global information about the entity, such as the number of rows in an image, or the slope of an edge. An example of a complex property is the histogram of an image.

The parts component consists of any number of part sets, which are collections of other entities in the system. This allows the user to represent the natural decomposition of spatial and image data in an organized hierarchy. For example, a **View_Class** HRS may be defined to contain an **Images** part set, which in turn may contain an **Edges** part set. The relations component consists of attributed relations over the parts of that entity. Each relation is made up of a set of tuples. A tuple consists of an ordered list of pointers to entities in the parts sets, and an optional list of attributes. For example, the **Image** HRS may contain a **proximity** relation, whose tuples consist of pairs of edges and a numeric attribute describing the distance between them.

Type checking is performed dynamically as HRS objects are constructed, ensuring that the attributes, parts and relations of each entity are consistent with the type constraints imposed by the schema. The schema model could be extended to support more flexible constraints, such as two-way relationship pointers, numeric range restrictions, set cardinality requirements and set membership conditions.

4. Example Application: TRIBORS

The HRS data model has been used successfully to support a number of imaging applications including robot vision and medical imaging. Figure 1 shows some of the data types used in the *Triplet-Based Object Recognition System* (TRIBORS) [23], an application that uses synthetic images to create probability models for use in 3D object recognition. TRIBORS was originally implemented without the HRS model

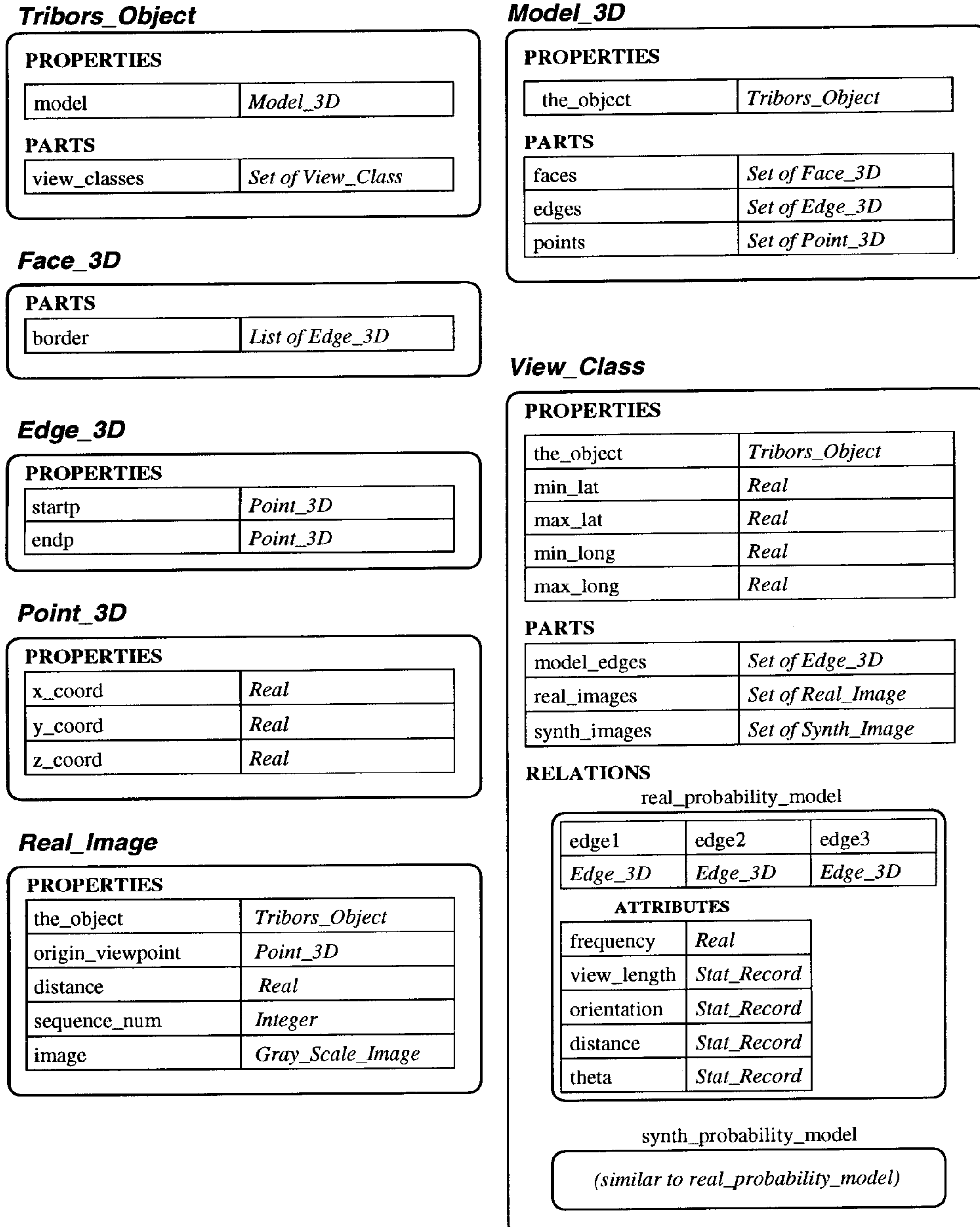

Figure 1. Schemas from the TRIBORS Application

or the DEVR system, using arbitrary data structures and ASCII file dumps to maintain data between executions. The input image files were scattered in various directories maintained by the system's designer. The HRS model easily supported the TRIBORS data types, and DEVR's application programmer's interface was used to link TRIBORS with the database. DEVR can now maintain the input images, synthetic images and intermediate data structures (such as extracted edges).

TRIBORS recognizes 3D objects from 2D images. Each 3D object has a full 3D CAD model and a set of 2D *view classes.* Each view class represents a region of contiguous views on the viewing sphere in which a specific set of features is visible. In the DEVR data model, each 3D object to be processed by TRIBORS is represented by a **Tribors_Object** HRS, which consists of a 3D model and a set of view classes. The **Model_3D** HRS is decomposed into faces, edges and points. Each **View_Class** HRS has properties defining its region on the viewing sphere and a part set of its visible 3D edges. The **View_Class** HRS also contains part sets of real and synthetic images of the object taken from viewpoints in the specified viewing region. These in turn reference viewable gray scale images. TRIBORS generates a probability model for each view class, which is stored as a relation in the HRS for that view class. Each tuple of the probability relation consists of a triple of edges from the model, with attributes describing the orientation of the segments and the frequency of the triple's occurrence within the training images for that view class. The actual CAD models from TRIBORS experiments have been successfully imported into DEVR, including multiple view classes consisting of over a hundred images and their corresponding spatial entities.

5. Multi-Level Queries

The system supports multi-level queries based on recursive constraint trees. A set of HRS entities of a given type is filtered through a network of constraints corresponding to the parts, properties and relations of that type. Queries can be constructed interactively with a menu-driven interface, or they can be generated dynamically within a vision application using the programmer's interface. Query objects are persistent and reusable. Users may keep libraries of query templates, which can be built incrementally, tested separately, cloned and linked together to form more complex queries.

5.1. Queries over Properties and Parts

A query is modeled as a *recursive constraint tree,* which consists of a root node that includes references to zero or more children, each of which is a recursive constraint tree. The components of a root node are its schema type, property constraints, part constraints and relational constraints.

The schema type is one of the atomic or user-defined schemas in the database; this will be referred to as the *base schema.* A set of entities of this type is returned by the execution of the query. The property constraints correspond to the properties of the base schema. Since property values can be either atoms or HRS structures, property

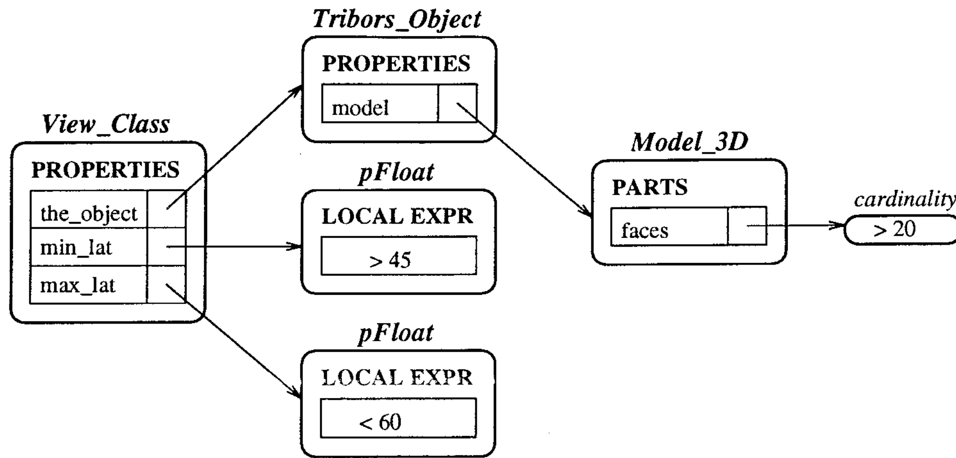

Figure 2. Query on a set of view classes in TRIBORS

constraints can be either Boolean expressions that operate on atomic values or subqueries that operate on HRS structures. The subquery constraints form new nodes of the recursive constraint tree.

The parts section of an HRS contains zero or more sets of entities, each of which is an HRS. Each part set has a schema that covers all the parts in that set. As for non-atomic properties, parts constraints are subqueries that operate on HRS structures. However, while a property constraint is concerned with a single HRS, a parts constraint is concerned with a whole set of HRSs of the same type. Thus, in addition to the subquery that expresses a constraint over HRSs of a given type, there is also a *cardinality requirement* that indicates how many entities in the returned parts set must satisfy this constraint. Cardinality may be expressed as an absolute number or as a percentage of the entities to be returned.

To illustrate the expressive power of the DEVR query model, Figure 2 shows the following query that was tested on the TRIBORS application: '*For all objects whose models have at least* 20 *surfaces, find the view classes whose latitude falls between* 45 *and* 60.'

The following are further examples of queries that a vision researcher might apply to the TRIBORS database:

1. Find all 3D edges of a particular model.
2. Find all models that have a face whose border has more than six 3D edges.
3. Find all 3D edges of a particular model that share an endpoint with a given 3D edge.
4. Find all view classes associated with a particular model.
5. Find all view classes (of any object) that contain more than 15 model edges.
6. Find all real images associated with view classes whose models have more than 25 3D edges.
7. Find all edge triplets of the real probability model of a selected view class that have frequency less than 0.3.
8. Find all models that have at least 20 real images associated with a single view class.

Users may construct queries interactively via the menu-driven *Query Specification Interface,* which prompts for Boolean constraint expressions and sub-query links. In addition, a graphical interface has been designed, in which queries will be visualized as a network of icons that can be manipulated with a mouse.

The query object acts as a filter on a candidate set of HRS entities of the return type, yielding a result set which is the subset of those candidates satisfying every constraint in the query. The system provides a Set class which enables the user to store the results of queries for further processing and browsing. The Set class includes facilities for iterating over its members and maintaining local indexes. To test whether a candidate entity satisfies a constraint, a depth-first, recursive traversal of the constraint tree is performed. Each constraint in the tree is applied to the corresponding node of the candidate entity, whose components must satisfy the conditions of that constraint. If all nodes of the constraint tree are satisfied, a pointer to the candidate entity is inserted into the result set.

5.2. Advanced Queries

The relations section of an HRS contains zero or more attributed relations over tuples of parts. These relations form a *structural description* [26] of the entity represented by the HRS in terms of its parts and their inter-relationships. Two relational descriptions can be compared to produce a numeric quantity called the *relational distance* [28]. Queries involving the relations must take in an instance of an HRS and return the set of HRSs whose relational distance satisfies a specified constraint. This ability to find good matches for structural descriptions of entities has been used extensively in our computer vision research [27] [4]. It leads us to consider which general matching capabilities would be appropriate.

Atomic property constraints in DEVR are standard Boolean expressions that could be applied to data in any relational database system. Non-atomic property constraints invoke subqueries; this ability is part of any object-oriented database system. Parts constraints not only invoke subqueries, but also consider the question of how many entities in a parts list must satisfy a constraint. Relational constraints allow a form of structural matching that could be used to retrieve images according to their content. As indicated in the related literature, there are now many different algorithms for retrieving images by content, using distance measures based on color, texture, shape and (in a few cases) relationships among extracted regions. The usual form of a query in these systems is to give the system an image and ask it to return matching images from the database according to a particular distance measure and acceptability threshold.

In order to add full query-by-image-content to the DEVR system, the form of the queries must be generalized. Boolean expression constraints involve comparison of atomic values. We would like to be able to compare *any two structures* according to an arbitrary built-in or user-provided function. The function could compare two images, two relational structures or two sets of scientific data. The query must allow the user to specify the image or other structure to be matched, the function that does the matching, and a Boolean constraint that the result must satisfy. For example, suppose that there is a function called *Histogram-Distance* that inputs two images in KHOROS viff format and returns a real-valued similarity measure. Suppose that the

user has defined a schema called *gray-scale-image* that has one property called *image-data* whose value is a KHOROS viff format image. Suppose that the user wants to retrieve all instances of *gray-scale-image* whose *image-data* value is similar to a particular KHOROS viff format image called *test-image.* Then in a query whose base type is *gray-scale-image,* the user would enter a constraint associated with the *image-data* property such as:

$$\textit{Histogram-Distance}(\textit{test-image}, *) < 5.0$$

Conceptually, this tells the system to use *Histogram-Distance* to compare *test-image* to each image referenced by the *image-data* field of a *gray-scale-image* entity and to return all *gray-scale-image* instances that satisfy the constraint. In a real system that stores hundreds or thousands of images, comparing an input image to the entire database is impractical. Most standard database systems use indexing mechanisms such as B-trees and hash tables to avoid large searches. Standard indexing mechanisms do not, however, extend to image content. Berman [2] has developed a method for organizing a database of images based on a known image-distance metric and a corresponding retrieval method that is able to eliminate many images from consideration based on their precomputed distances to a set of index images. We are now working on the extension of this technique to allow queries that express image distance as a combination of multiple distance measures. These kinds of queries are being implemented in current research, but are not yet part of the DEVR system.

6. Visualization Construction

The DEVR human-computer interface is composed of several graphical tools with which the user can access and manipulate different aspects of the database. The main window of the system can be thought of as a toolbox which provides the user access to these tools via a menu. Each graphical tool produces its own window with its own particular visual interface. A subset of these tools forms the needed components for the visualization subsystem of DEVR. These include the Schema Constructor, the Graphic Editor within the Schema Constructor and the Instance Browser. The following sections describe the process of defining an HRS schema, defining graphical elements to associate with it, creating the graphical elements and finally, browsing instances of the schema and its graphical elements.

6.1. Schema Construction

Through the HRS Schema Constructor, the user can create and modify new schemas for his own HRSs. The HRS schema creation process allows the user to add, modify and delete properties, parts and relations within the schema. New HRS schemas can copy the properties, parts and relationships from other HRS schemas. Once the user has finished defining a new HRS schema, he can add it into the database where it can be shared by all users. After an HRS schema is entered into the database, it may only be modified or deleted if no instances of it exist.

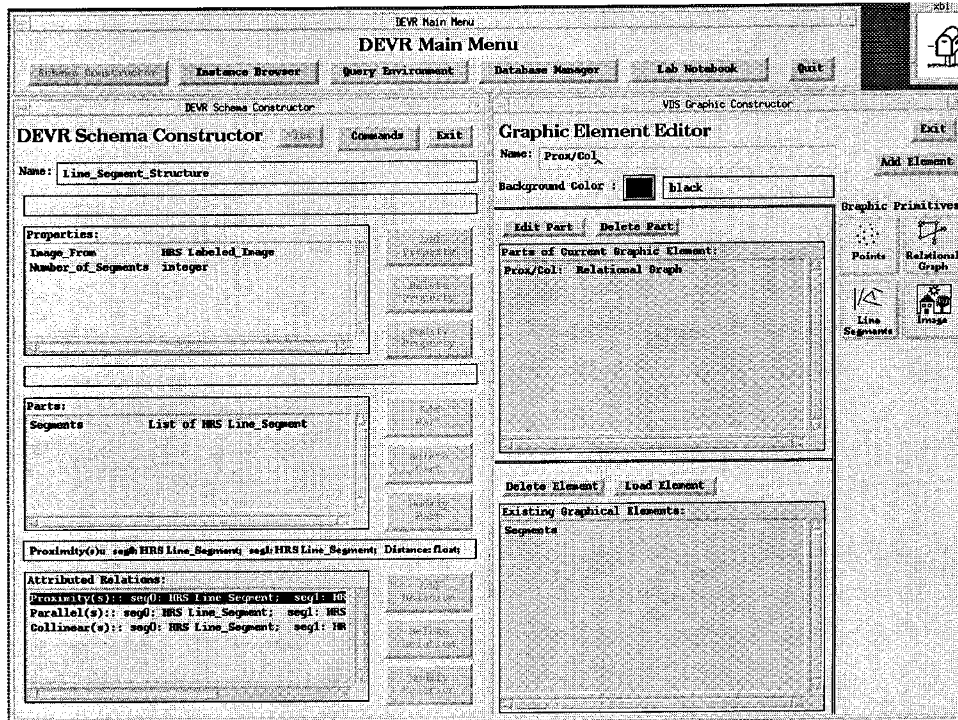

Figure 3. The DEVR Schema Constructor and Graphic Editor

Since DEVR is intended for a wide group of users, many of whom are not comfortable with programming, we provide a graphical user interface to facilitate user input. The names of new properties, parts and relationships are typed in by the user, but almost everything else can be specified through a selection process. The type of each property, part and relation can be an atomic type (i.e. integer, real, character string), another HRS, a union of HRSs, a list or multi-dimensional array of one of these types, or an undefined schema marker.

Figure 3 (left side) illustrates the process of defining the HRS Line Segment Structure. Two properties, Image_From and Number_of_Segments; one part, Segments; and three symmetric relations, Proximity, Parallel and Collinear, have been defined. When adding a new relation to the schema of an HRS, the system brings up a window allowing the user to specify a name for the relation and to create, modify and/or delete two types of information: the tuple elements over which the relation holds and the attributes of the relation. The user selects tuple element types from the types found in the parts list. If the order of the tuple elements is insignificant, the user can mark the relation as symmetric. When adding tuple elements and attributes, other windows appear for specifying their names and types. Once the user has the properties, parts and relations of an HRS schema defined, he/she can use the Graphic Editor within the Schema Constructor to define graphical elements that are associated with the schema.

6.2. Graphical Elements

In a strict object-oriented model, one might expect each object (or entity) to contain a method for displaying itself. We have found this approach to be limiting in several ways. First, some entities may require several visualizations and/or the data to be visualized may span multiple entities. Therefore, having a single visualization routine for each entity is inadequate. Second, as mentioned earlier, the visualizations that a user will want to create are highly dependent on the domain of his/her data and the techniques used to process and/or analyse it. Thus, it would be difficult, if not impossible, to create a set of predefined visualization methods that would suit every user's needs in every situation. This forces the user to write his/her own visualization routines, which is precisely what we are trying to avoid.

For these reasons, we decided that instead of having a canned visualization routine for every desired visualization type, we would provide the user with graphical building blocks or primitives that she could combine to form desired visualizations. We refer to such visualizations as 'graphical elements'. Once a user has defined the properties, parts and relationships of a new HRS schema, she can define graphical elements for the schema via the graphic editor within the Schema Constructor. The graphic editor window appears beside that of the Schema Constructor to allow easy interaction. Figure 3 (right side) shows the Graphic Editor.

The graphical element creation process begins by specifying the name and background color of the graphical element. Along the right side of the editor are icons representing each of the available graphical primitives. In the current system there are only four primitives: sets of points, sets of line segments, images and relational graphs. These four graphical primitives were chosen as a small sample set to fit the needs of our sample HRS schema sets. We believe that these primitives will serve well for a variety of vision applications, but additional graphical primitives can be added in the future.

In Figure 3, a graphical element, named 'Prox/Col', with a black background color is being created for the HRS Line_Segment_Structure. Currently, 'Prox/Col' contains a single graphical primitive of type relational graph, which illustrates the proximity and collinear relations among the line segments. Another graphical element, 'Segments', has previously been defined for HRS Line_Segment_Structure.

The user creates a graphical element by selecting the desired graphical primitives. Each graphical primitive has a corresponding window. The window prompts the user for the source of the data needed to produce instances of the graphical primitive, and the user can also select colors, patterns, labels and symbol types. In this phase of the definition, the user can select various properties, parts and relations of the current HRS schema from which data for the graphical primitives is to be retrieved. He/she can also follow links in the HRS schema to other schemas that it contains and select properties, parts and relations from these other HRS schemas. Thus the visualization for a complex HRS can be made up of graphical elements from many different portions or levels of its structure.

When a user creates a graphical element, it is stored as a metadata property of the HRS schema. Visualizations for the instances of an HRS can now be created using the stored metadata and a set of graphical primitive creation routines. Graphical element instances are not created at the time that the corresponding HRS schema instance is

created. Instead, graphical element instances will be lazily created the first time a user requests to view them. This approach can save a significant amount of unnecessary time and space when the user does not need to view the graphical elements of each HRS instance. After a graphical element instance is created, it will be stored along with the HRS instance.

6.3. Instance Browsing

After the user has defined a set of HRS schemas and created instances with actual data, he will be able to view the data via the Instance Browser.[a] The DEVR browsing environment was designed using the metaphor of having piles or stacks of HRS entities on one's desk. There are three stacks aligned in a horizontal fashion. Each stack can hold up to three HRS entities. Unlike stacks on one's desk, however, the environment ensures that the stacks stay neat and orderly and provides the user with an easy and intuitive way to manipulate the HRS entities. Above the stacks is a header or title bar which contains various tools for loading and manipulating different working sets of HRS entities.

Figure 4 shows the window design of the Instance Browser, in which the user is viewing three HRS Gray_Scale_Image entities and the associated HRS Line_Segment_Structure entities. The Gray_Scale_Image entity, 'f1', has its graphical element, 'Gray scale', displayed in the first viewing box. The Line_Segment_Structure, entity, 'f1.lines', has its graphical element, 'Segments', displayed in the second viewing box. While each of these graphical elements only contain a single graphical primitive, this is not always the case.

7. Experiment Management

An experiment management system provides computer-based support for scientific research work [30]. Interviews with imaging scientists working on complex remote sensing and medical analysis problems identified the following desirable properties for such a system:

- Exploratory—an experiment management system should facilitate the scientist's exploration of different algorithmic solutions and help the scientist to identify their effects on the results.
- Responsive—algorithm results should be returned as quickly as possible, particularly if the scientist is waiting for them.
- Satisfies User Requirements—an experiment management system should schedule and execute algorithms based on the scientist's requirements for resource utilization and algorithm execution. For example, the scientist should be able to specify which results are most important, what processing resources are available and how to utilize these resources.
- High-Level—the interface for an experiment management system should let the

[a] The Instance Browser has been designed but not implemented.

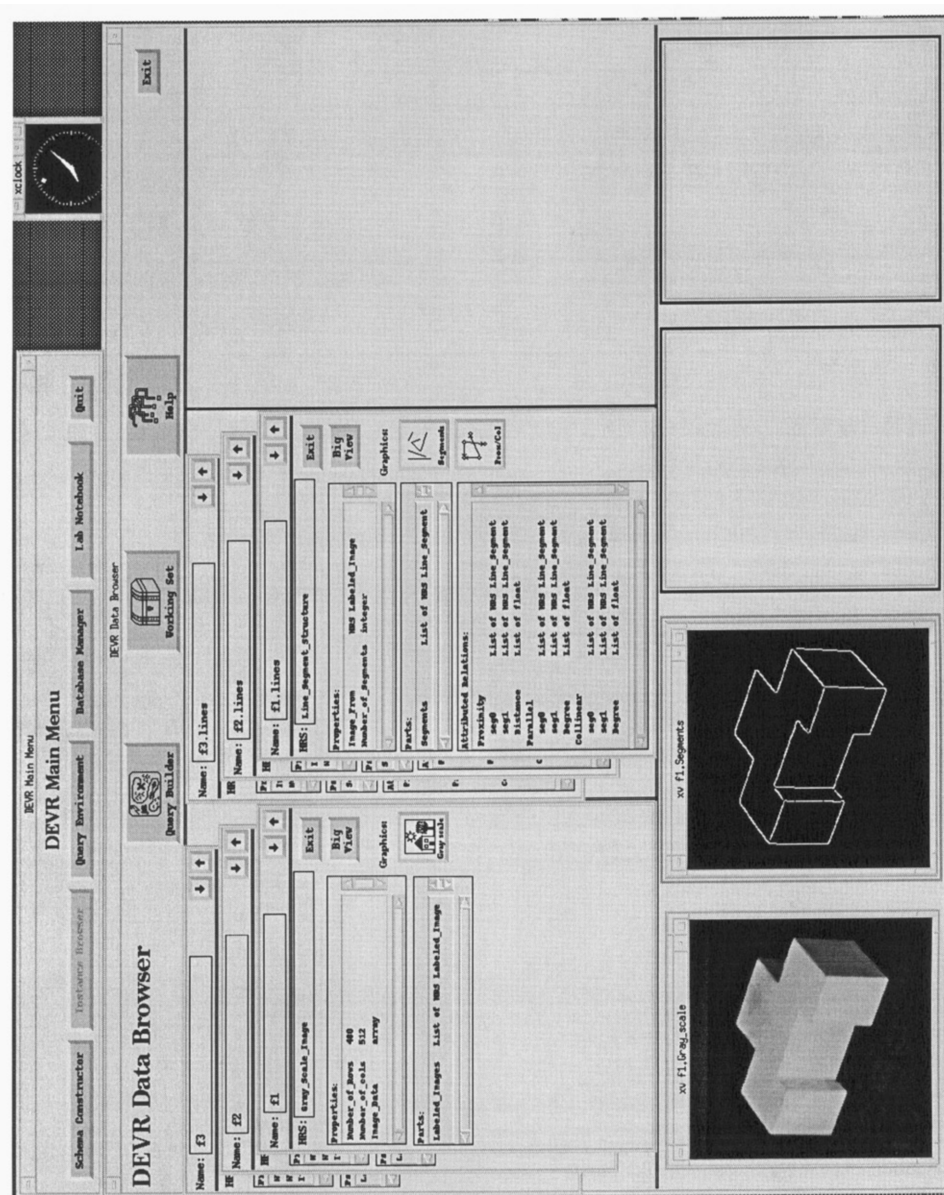

Figure 4. The DEVR Instance Browser

scientist specify a high-level description of his algorithms and requirements. An experiment management system should provide support for scientists who are not computer experts.

- Organized—an experiment management system should record and organize the scientist's computer-based research work for later retrieval. This increases the scientist's productivity.

DEVRs experiment management system has these properties. The main components of this system are (1) a visual programming environment, (2) the underlying scientific database, (3) a scheduler for networks of workstations, and (4) an executor that runs the experiments and keeps track of results. The database itself is used to organize and store information about program graphs and results.

The scientist uses a data-flow based visual programming environment (currently Khoros 2.0 [24]) to specify his algorithms in a declarative manner. This makes it easy to explore different algorithms by interactively modifying the data-flow program graph. The visual programming environment interacts with the database and the experiment management system using special input and output operators. The database input operator inputs the results of database queries. The database output operator stores program graph results in the database along with associated metadata. This metadata contains information about how and when a result was created. Queries on this metadata can later be used to retrieve specific results.

Many scientists have access to a network of workstations that can be used for parallel execution of computationally-intensive experiments. In our system, the scheduler and executor automatically schedule and execute a program graph on a network of workstations based on the scientist's requirements for resource utilization and algorithm execution. The requirements are specified declaratively as constraints, which can be either requirements or preferences. Requirements must hold in the resulting schedule, whereas preferences are used to guide a search for an optimized schedule.

8. Conclusions and Future Work

DEVR provides a unified data model, a powerful query processing facility and an associated experiment management system. The HRS data model promotes interoperability between applications and provides a practical framework in which data may be shared among researchers. A scientific user can design schemas for entities that include the graphics necessary for their visualization. The query facilities allow the construction of powerful, multi-level queries to retrieve the hierarchical structures. The inclusion of an experiment management system makes a total package in which scientists can develop, run and analyse the results of their experiments.

A prototype DEVR system has been designed and partially implemented. Schema construction and multi-level querying are operational, but the browser for visualization of results was not implemented due to time and funding constraints. The experiment management system uses the visual programming environment of Khoros 2.0, a public domain visualization package, and interfaces to the database system through special storage/retrieval icons.

The tools developed for DEVR are a good start toward the development of a full

image database system including retrieval of images and related structures according to their content. We intend to continue our work in this direction.

References

1. R. Agrawal & N. H. Gehani (1989) ODE (Object Database and Environment): the language and data model. In: *Proceedings of the ACM-SIGMOD 1989,* Association for Computer Machinery, pp. 36–45.
2. A. Berman (1994) A new data structure for fast approximate matching. Technical Report 1994-03-02, Dept. of Computer Science, University of Washington, Seattle, Washington.
3. J. Brolio, B. A. Draper, J. R. Beveridge & A. R. Hanson (1989) ISR: A database for symbolic processing in computer vision. In: *IEEE Computer,* **22,** 22–30.
4. O. Camps, L. G. Shapiro & R. M. Haralick (1992) Object Recognition using prediction and probabilistic matching. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems,* IEEE, pp. 1044–1052.
5. S. K. Chang, C. W. Yan, D. C. Dimitroff & T. Arndt (1988) An intelligent image database system. In: *IEEE Transactions on Software Engineering,* **SE1** (5), 681–688.
6. S. K. Chang (1989) *Principles of Pictorial Information Systems,* Prentice Hall, Englewood Cliffs, NJ.
7. N. S. Chang & K. S. Fu (1981) Picture query languages for pictorial data-base systems. In: *IEEE Computer* **14,** 23–33.
8. W. W. Chu, A. F. Cardenas & R. K. Taira (1993) A knowledge-based multimedia medical distributed database system—KMeD. In: *Proceedings of the Workshop on Advances in Data Management for the Scientist and Engineer,* (W. Chu, A. Cardenas & R. Taira, eds) (WADMSE).
9. A. Del Bimbo, P. Pala & S. Santini (1994) Visual image retrieval by elastic deformation of object sketches. In: *IEEE Symposium on Visual Languages,* 216–223.
10. S. Ghandeharizadeh, V. Choi, C. Ker & K. M. Lin (1992) *Design and Implementation of the Omega Object-based System.* Computer Science Dept., University of Southern California.
11. A. Goodman, R. M. Haralick & L. G. Shapiro (1989) Knowledge-based computer vision: integrated programming language and data management system design. In: *IEEE Computer,* **22,** 43–58.
12. A. Gupta, T. Weymount & R. Jain (1991) Semantic queries in image databases. In: *Proceedings of the IFIP 2nd Working Conference on Visual Database Systems,* pp. 204–214.
13. N. I. Hachem, M. A. Gennert & M. O. Ward (1993) The Gaea system: a spatio-temporal database system for global change studies. In: *Proceedings of the Workshop on Advances in Data Management for the Scientist and Engineer* (W. Chu, A. Cardenas & R. Taira, eds) (WADMSE).
14. K. Hirata & T. Kato (1992) Query by visual example. In: *Advances in Database Technology—EDBT*'92 Berlin: Springer-Verlag, pp. 56–71.
15. C. E. Jacobs, A. Finkelstein & D. H. Salesin (1995) Fast multiresolution image querying. In: *Computer Graphics Proceedings,* Annual Conference Series. Addison-Wesley, Reading, PA, pp. 277–286.
16. A. Joseph & A. F. Cardenas (1988) PICQUERY: a high-level query language for pictorial database management. In: *IEEE Transactions on Software Engineering,* **14,** (5).
17. T. Kato, T. Kurita, N. Otsu & K. Hirata (1992) A sketch retrieval method for full color image database. In: *11th International Conference on Pattern Recognition,* IEEE Computer Society Press, Los Alamitos, CA, pp. 530–533.
18. R. H. Katz, D. A. Patterson, M. R. Stonebraker, C. Grautier, M. D. Dahlin, J. A. Fine & E. L. Miller (1993) Design of a large capacity object server supporting Earth System Science researchers. In: *Proceedings of the Workshop on Advances in Data Management for the Scientist and Engineer* (W. Chu, A. Cardenas & R. Taira, eds) (WADMSE).
19. P. M. Kelly & T. M. Cannon (1995) Query by image example: the CANDID approach. In: *SPIE Vol. 2420 Storage and Retrieval for Image and Video Databases III,* International Society for Optical Engineering, Bellingham, WA, pp. 238–248.

20. W. Niblack, R. Barber, W. Equitz, M. Flickner, E. Glasman, D. Petkovic, P. Yanker, C. Faloutsos & G. Taubin (1993) The QBIC project: querying images by content using color, texture, and shape. In: *Proceedings of the SPIE Conference on Storage and Retrieval for Image and Video Databases,* International Society for Optical Engineering, Bellingham, WA, pp. 173–181.
21. A. Pentland, R. W. Picard & S. Sclaroff (1993) Photobook: tools for content-based manipulation of image databases. In: *Technical Report* 255, MIT, Media Lab.
22. R. W. Picard & T. P. Minka (1995) Vision texture for annotation. In: *Technical Report* 302, MIT, Media Lab.
23. K. Pulli (1995) Tribors: a triplet-based object recognition system. Technical Report 95-01-01, Department of Computer Science and Engineering, University of Washington, Seattle, WA.
24. J. R. Rasure & C. S. Williams (1991) An integrated data flow visual language and software development environment. In: *Journal of Visual Languages and Computing,* **2**, 217–246.
25. L. G. Shapiro & R. M. Haralick (1980) A spatial data structure. *Geo-Processing* **1**, 313–337.
26. L. G. Shapiro & R. M. Haralick (1981) Structural descriptions and inexact matching. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence,* **PAMI-3**, 504–519.
27. L. G. Shapiro, J. D. Moriarty, R. M. Haralick, & P. G. Mulgaonkar (1984) Matching three-dimensional objects using a relational paradigm. In: *Pattern Recognition* **17**, 385–405.
28. L. G. Shapiro & R. M. Haralick (1985) A Metric for comparing relational descriptions. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence,* **PAMI-7**, 90–94.
29. T. R. Smith, J. Su, D. Agrawal & A. El Abbadi (1993) MDBS: a modeling and database system to support research in the Earth Sciences. In: *Proceedings of the Workshop on Advances in Data Management for the Scientist and Engineer* (W. Chu, A. Cardenas & R. Taira, eds) (WADMSE).
30. E. Soloway & W. Martin (1993) Computer-based Support for Scientific Data Analysis. In: *Proceedings of the Workshop on Advances in Data Management for the Scientist and Engineer* (W. Chu, A. Cardenas & R. Taira, eds) (WADMSE).