Appeared in Proc. IFIP 2.6 3rd Working Conference on Visual Database Systems (VDB-3), 1995.

Querying Multimedia Data from Multiple Repositories by Content: the Garlic[1] Project

W. F. Cody, L. M. Haas, W. Niblack, M. Arya, M. J. Carey, R. Fagin, M. Flickner, D. Lee, D. Petkovic, P. M. Schwarz, J. Thomas, M. Tork Roth, J. H. Williams and E. L. Wimmers
IBM Almaden Research Center

Abstract: We describe Garlic, an object-oriented multimedia middleware query system. Garlic enables existing data management components, such as a relational database or a full text search engine, to be integrated into an extensible information management system that presents a common interface and user access tools. We focus in this paper on how QBIC, an image retrieval system that provides content-based image queries, can be integrated into Garlic. This results in a system in which a single query can combine visual and nonvisual data using type-specific search techniques, enabling a new breed of multimedia applications.

1 Introduction

Many applications today require access to a broad range of datatypes. A patient's medical folder contains MRI scans (image), lab reports (text), doctors' dictated notes (audio), and address and insurance information (record-oriented database data). A geographic information system needs maps, satellite images, and data about roads, buildings, and populations. In many of these areas, specialized software has emerged to allow key datatypes to be queried efficiently, or to support type-specific predicates. For example, there are special systems for fingerprint recognition, for finding specific molecular structures, and to locate areas that overlap or that contain a specific object on a map. The expanding role of multimedia data in many other application domains has similarly resulted in special purpose systems that provide content based search of their data. Since multimedia data is largely visual and hard to describe precisely, it will be increasingly important to support content based searches that can be specified visually "by example" and that allow for degrees of similarity in the answer set.

The increasing diversity of datatypes and the need for special-purpose data servers is occurring even in traditional application areas like insurance (e.g., to manage videos of damaged property), catalog sales (e.g., to manage collections of photos for product spreads) and advertising (e.g., to manage shots of magazine ads). In these traditional applications, this new data must be managed in coordination with the large amounts of business data and text data that are already managed by a variety of information systems. In the current environment, developing a multimedia application requires the developer to deal with different interfaces for several different data systems, while worrying about how to locate the right system to handle each part of the query, how to optimize the accesses to the various data systems and how to combine the results into a meaningful form for the user. All these tasks are inhibitors to the creation of modern multimedia applica-

1. Garlic is not an acronym. Most members of the team really like garlic, and enjoy our laboratory's proximity to the Gilroy garlic fields!

tions that exploit the rich data environment we live in.

Garlic is an object-oriented multimedia middleware system that is designed to address this problem. Garlic allows existing data management components, such as a relational DBMS, a full text search engine, or a face recognition system, to be integrated into an extensible information management system. Applications can access any of the data in the underlying data sources through a common, nonprocedural interface, and can exploit the specialized query capabilities of those sources. A single query can access data in several repositories, using the type-specific predicates they support. Garlic also provides a powerful query/browse application that includes type-specific query interfaces in a uniform query framework.

In this paper, we show how Garlic enables applications that need content-based search of visual (and nonvisual) data stored in separate specialized servers. The paper is organized as follows. In the next section, we describe related work. An overview of Garlic is given in Section 3. Section 4 shows how visual data can be incorporated into Garlic. It introduces an image retrieval system supporting content-based image queries (QBIC), describes the steps and the decisions involved in integrating QBIC into Garlic, and then shows how queries combining visual and nonvisual predicates can be processed. At the end of this section we briefly describe a Query/Browse application and show how it allows visual data to be browsed and queried in conjunction with other data reachable through Garlic (Section 4.4). We summarize our contributions and discuss future work in Section 5.

2 Related Work

The multimedia area is expanding at a rapid pace. It includes work on hypermedia systems, specialized servers (e.g., video servers), image and document management tools, interactive games, authoring tools, scripting languages, and so forth. In the personal computer industry, a large number of small-scale multimedia software packages and products have emerged due to the availability and affordability of CD-ROM technology. Several companies are offering "multimedia database" products. These products combine the functionality of a DBMS (typically based on a relational or object-oriented model) with the ability to store images, text, audio, and even short video clips. These systems store and manage all their data, and typically provide keyword search for pre-annotated multimedia data. It is not clear that these systems can scale to large volumes of data.

Mainline database vendors have only recently started to pay attention to multimedia data. The Illustra object-relational DBMS [26] provides media-specific class libraries (DataBlades(tm)) for storing and managing multimedia data. IBM, Sybase, Oracle and others can store image, video and text in their databases, but support for searching these types by content is just starting to appear. IBM's new UltiMedia Manager is the first product to offer content-based image query (based on QBIC [19] technology) in conjunction with standard relational search. Garlic differs from these systems in that it aims to leverage existing intelligent repositories, such as text and image management systems, rather than requiring all multimedia data to be stored within and searched by a single DBMS. Garlic's open approach should enable it to take advantage of continuing advances in multimedia storage and search technology. It should also be more effective for legacy environments, where multimedia data collections (such as document or image libraries) and business data already exist in forms that cannot easily be migrated into a new DBMS.

Content-based retrieval of data is highly type-specific. Years of research have produced a solid technology base for content-based retrieval of documents through the use of various text indexing and search techniques

[22]. Similarly, simple spatial searches are well-supported by today's geographic information systems ([29], [30], e.g.). Content-based retrieval of visual data is still in its infancy. Although a few specialized commercial applications exist (such as fingerprint matching systems), most content-based image retrieval systems are university and research prototypes. Examples include [20], [12], and [24]. Further, with the exception of simple approaches like attaching attributes to spatial objects, or associating user-provided keywords with images, these component search technologies remain largely isolated from one another.

In the database community, much research has been done in the area of heterogeneous distributed database systems (also known as multidatabase systems). These systems aim to enable applications that span multiple DBMS. Surveys of the relevant work can be found in [7] and [10]. Commercial middleware products now exist for providing uniform access to data in multiple databases, relational and otherwise, and to structured files, usually through the provision of a unified relational schema. Models with object-oriented features have been employed in projects such as [21], [5], [8] and others. What distinguishes Garlic from these efforts is its focus on providing an object-oriented view of data residing not only in databases and record-based files, but also in a wide variety of media-specific data repositories with specialized search facilities. With the exception of the Papyrus [5] and Pegasus [23] projects at HP Labs, we are aware of no other efforts that have tried to address the problems involved in supporting heterogeneous, multimedia applications.

3 Garlic Overview

Figure 1 depicts the overall architecture of the Garlic system[4]. At the leaves of the figure are a number of data repositories containing the data that Garlic is intended to integrate. Examples of potential data repositories include relational and non-relational database systems, file systems, document managers, image managers, and video servers. Repositories will vary widely in their ability to support content-based search, from a video server which can simply retrieve by video name, to a relational DBMS with its powerful query language. While Garlic will accommodate (i.e., provide access to) more limited servers, we are particularly interested in enabling a richer style of query for a broader range of datatypes. Thus we focus on repositories that provide content-based querying of multimedia datatypes, and on the technology needed to incorporate them into Garlic, in such a way as to exploit their special abilities.

One special repository shown in Figure 1 is the Garlic complex object repository. This repository, provided with Garlic, is used to hold the complex objects that most Garlic applications need to relate together legacy information from different systems, or to create new multimedia objects. For example, an advertising agency that had information about its clients in a relational database, stills of ads in an image server, video clips on a video server and financial reports in a document manager might build Garlic complex objects representing the ad campaigns to link all of this information together.

Above each repository is a repository wrapper. A repository wrapper serves two purposes. First, it exports to Garlic a description of the data types and collections of data that live in that underlying repository. This description is basically a schema for that repository instance, expressed in the Garlic Data Model [4] (a variant of the ODMG-93 object model [3]). It also describes to Garlic the search capabilities of this repository type -- what predicates it supports. Second, the wrapper translates data access and manipulation requests (i.e., queries) from Garlic's internal protocols to the repository's native protocol. Initially, wrappers will have to be created by hand; eventually, we plan to provide tools to ease the task of wrapper generation.

Query processing and data manipulation services, especially for queries where the target data resides in

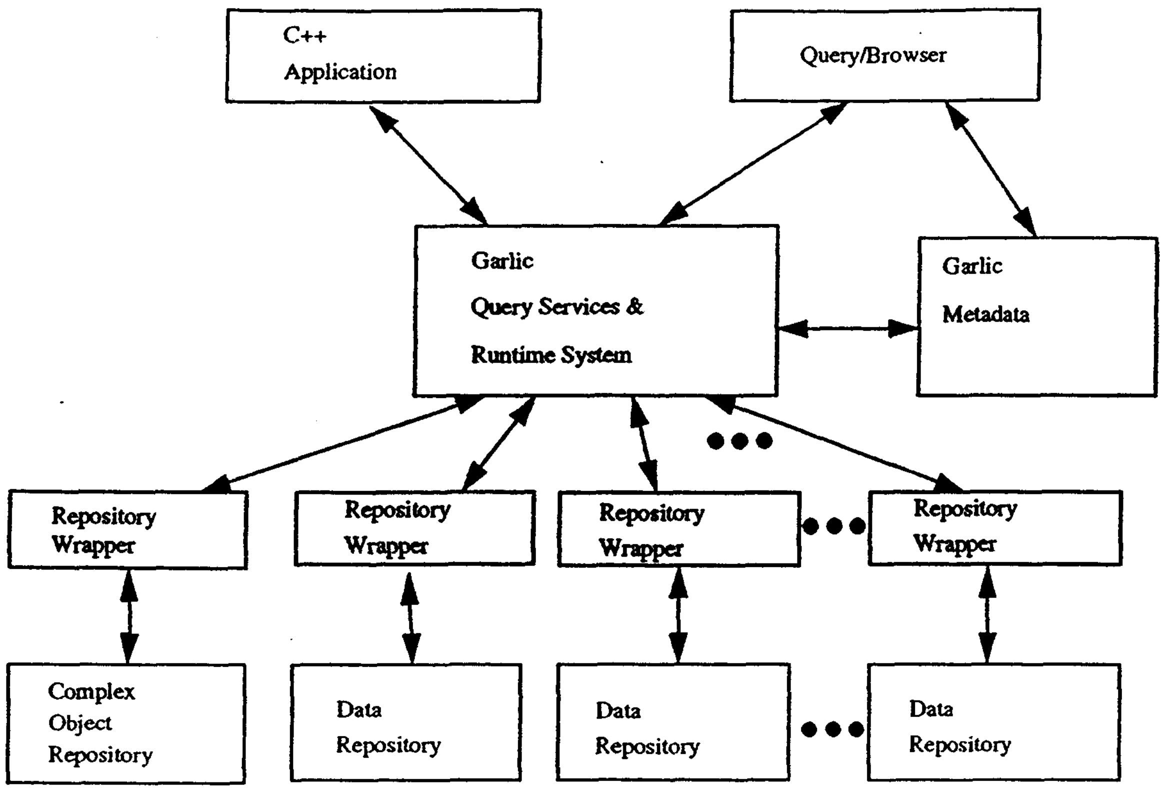

Figure 1. Garlic System Architecture

more than one repository, are provided by the Garlic Query Services and Runtime System component shown in Figure 1. This component presents Garlic applications with a unified, object-oriented view of the data accessible by Garlic. This view may be a simple union of all of the repository wrapper schemas, or it may involve subsetting or restructuring of those schemas. Garlic Query Services processes users' and applications' queries, updates and method invocation requests against this view. Queries, expressed in an object-oriented extension of SQL called GQL, are broken into pieces, each of which can be handled by a single wrapper. This process relies on Garlic metadata that describes both the unified Garlic schema and the individual wrapper schemas. The subqueries are initiated by the Garlic Runtime System and the results are combined and returned to the user.

Garlic applications interact with the Query Services and Runtime System through Garlic's object query language and a C++ application programming interface (API). One particularly important application, which is also shown in Figure 1, is the Garlic Query/Browser. This component of Garlic will provide end users of the system with a friendly, graphical interface that supports interactive browsing, navigation, and querying of Garlic databases.

4 Querying Visual Data in Garlic

In this section, we focus on how queries involving visual data can be handled in Garlic. We start by describing one particular image repository that we are integrating; the QBIC repository provides the ability to search for images by various visual characteristics such as color, texture or layout. We then discuss the design of a wrapper for this repository. Once a wrapper is defined, it is possible to query data in this repository through Garlic. The advantage of Garlic, however, is its ability to handle queries spanning data in visual and other repositories. We illustrate this with an example involving three repositories. Finally, we describe the Garlic query/browser application, and show how it could be used in the same example.

4.1 Query by Content of Image Data -- the QBIC Repository

QBIC [19] is a research prototype image retrieval system that uses the content of images as the basis of queries. The content used by QBIC includes the colors, textures, shapes, and locations of objects (e.g., a person, flower, etc.) or specified areas (e.g., the sky area) in images, and the overall distribution and placement of colors, textures, and edges in an image as a whole. Queries are posed graphically/visually, by drawing, sketching, or selecting examples of what is desired. A sample QBIC query is "Find images with a generally green background that have a red, round object in the upper left corner", where the image predicates (red, round, ...) are specified graphically using color wheels and drawing tools, by selecting samples, and so on.

QBIC is a stand-alone system. It has two main components, database population, which prepares a collection of images for query, and database query. Each component has its own user interface and engine. In this section, we describe these two components, and in the next, consider the issues involved in making QBIC's collections and query function accessible to Garlic.

4.1.1 QBIC Database Population

The QBIC database population step is a one-time process that prepares images for later query. The images are loaded or imported into the system, and several utility operations are performed -- preparing a reduced 100x100 "thumbnail", converting each image to a common system palette and storing available text information. An optional but important step is "object/area identification" in which objects or areas in an image -- a car, a person, swatch of background texture -- are identified. This may be done manually, semi-automatically, or fully automatically, depending on the nature of the images and the objects they contain. For unconstrained natural scenes and general photo clip art, objects are usually identified manually by outlining with a mouse, or by using semi-automatic tools such as flood-fill algorithms for foreground/background identification, or spline-based edge tracking to refine a rough user outline. Automatic methods such as background removal can be used in constrained cases such as images of museum artifacts on generally uniform backgrounds, or images of industrial/commercial parts in a fixed position and under controlled lighting. In any case, the result of object/area identification is a set of outlines or, more generally, bit masks (to allow for disconnected and overlapping areas) defining objects and areas in the images.

For each object/area and for each image as a whole, a set of numeric features are computed that characterize properties of image content. These features are listed in Table 1, and described briefly below.:.

Average and Histogram Color: QBIC computes the average Munsell [17] coordinates of each object and image, plus a *k* element color histogram (*k* is typically 64 or 256) that gives the percentage of the pixels in each object/image in each of the *k* colors.

Texture: QBIC's texture features are based on modified versions of the coarseness, contrast, and directionality features proposed in [25]. Coarseness measures the scale of the texture (pebbles vs. boulders), contrast describes the vividness of the pattern, and directionality describes whether or not the image has a favored

TABLE 1. *QBIC Features*

Objects	Images
Average color	Average color
Histogram color	Histogram color
Texture	Texture
Shape	Positional edges (sketch)
Location	Positional color (draw/paint)

direction or is isotropic (grass versus a smooth object).

Shape: QBIC has used several different sets of shape features. One is based on a combination of area, circularity, eccentricity, major axis orientation and a set of algebraic moment invariants. A second is the turning angles or tangent vectors around the perimeter of an object, computed from smooth splines fit to the perimeter. The result is a set of 64 values of turning angle. All shapes are assumed to be non-occluded planar shapes allowing each shape to be represented as a binary image.

Location: The location features are the x and y centroid of the object.

Positional edge (sketch): QBIC implements an image retrieval method similar to the one described in [9],[12] that allows images to be retrieved based on a rough user sketch. The feature needed to support this retrieval consists of a reduced resolution edge map of each image. QBIC computes a set of edges using a Canny edge operator, and then reduces this to a 64 x 64 edge map, giving the data on which the retrieval by sketch is performed.

Positional color (draw): Positional color or "draw" features are computed by gridding the image into a number of roughly square subimages and, for each subimage, computing a partial color histogram that captures the main colors in the subimage, texture parameters for the subimage, etc. The set of computed features, one for each subimage, is the draw feature.

4.1.2 QBIC Image query

Once the set of features for objects and images has been computed, queries may be run. Queries are initiated by a user in an interactive session by graphically specifying a set of image and object properties and requesting images "like" the query specification. For example, images may be requested that contain objects whose color is similar to the color of an indicated object, or a color selected from a color wheel. Full image queries are based on the global set of color and texture features occurring in an image. For example, images may be retrieved that are globally similar, in terms of color and/or texture, to a given image, or, using a menu-based color or texture "picker", a user can select a set of colors and textures and request images containing them in selected proportions. Sample pickers for various features are shown below.

All retrievals on image features are based on similarity, not exact match, and similarity (or inversely, distance) functions are used for each feature or feature set. Most of the similarity/distance functions are based on weighted Euclidean distance in the corresponding feature space (e.g. three dimensional average Munsell color, three dimensional texture, or 20 dimensional shape). Special similarity measures are used for histogram color, turning angle shape, sketch and positional color, as described in [19]. These measures can be used individually or in a weighted combination. Also, "multi-queries" can be formed, querying on multiple objects, each with multiple properties, and on multiple image attributes, as in a query for an image with a red, round object, a green fish-shaped object, and a blue background.

Example queries are shown in Figures 2, 3, 4, and 5. In all cases, the returned results are ranked, and are shown in order with the best result in the leftmost position, next best in the next position, and so on. Each image returned is displayed as a reduced "thumbnail". The thumbnails are active menu buttons that can be clicked on to give a list of options. The options include: initiate the query "Find images like this one", display the similarity value of this image to the query image, display the (larger) full scale image, place the image in a holding area for later processing, or perform as user defined image operationf or comparison .

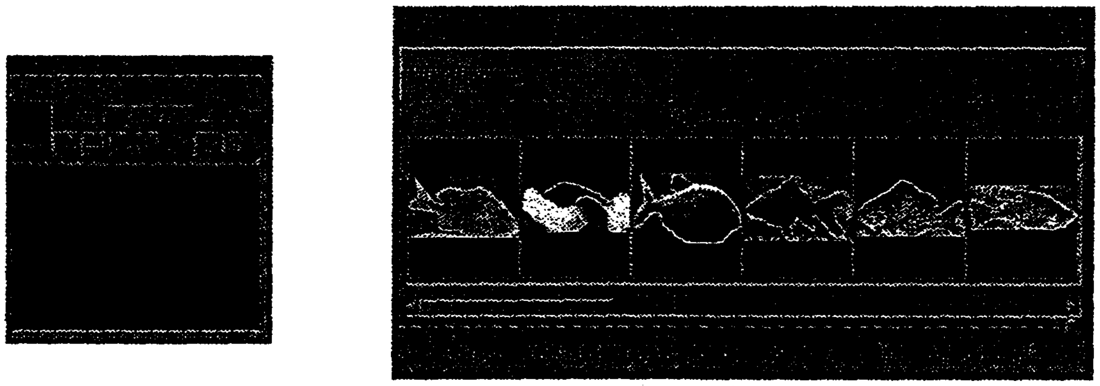

Figure 2. Example shape query. Left: Freehand sketch of shape. Right: Query results showing first six returned items.

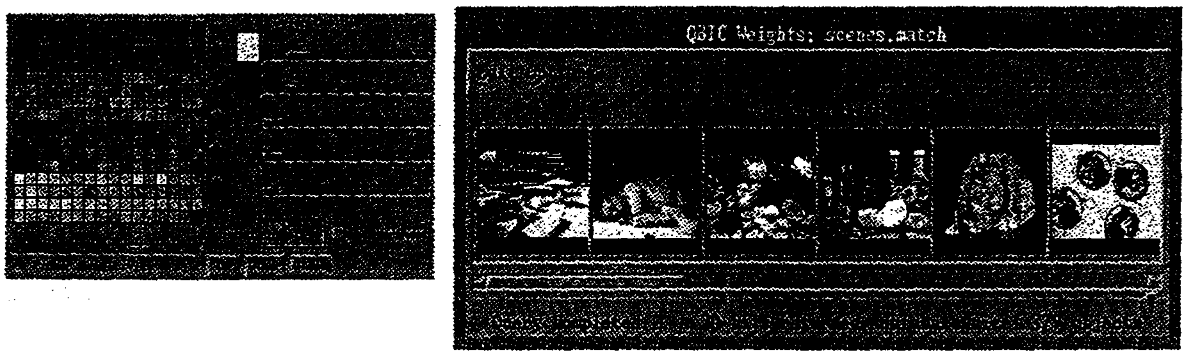

Figure 3. Example color histogram query. Left: Color selection show 15% yellow, 13% blue. Right: Query results showing first six returned items.

4.2 Wrapping a QBIC Repository

In this section is to show how QBIC can be integrated into Garlic. The goal of this integration is to enable applications to exploit QBIC's special facilities for image search in conjunction with other kinds of search on other types of data. So far, we have not thought about integrating QBIC's database population component. Thus, in this section we discuss integration of the two pieces of the database query component of QBIC: the query formation interface and the query engine.

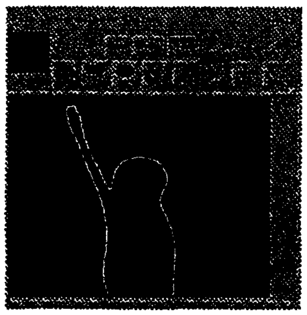
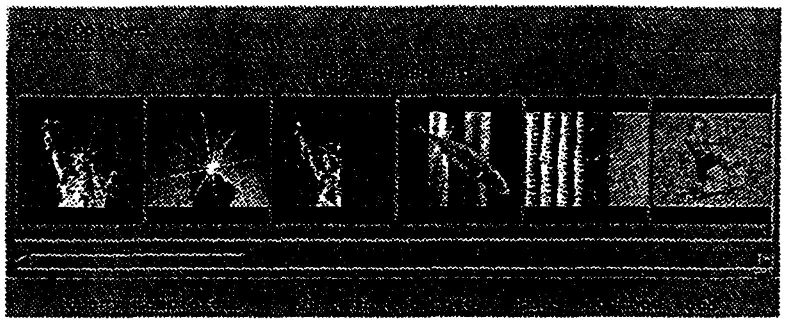

Figure 4. Example query by sketch. Left: Freehand drawn sketch. Right: Query results showing the first six returned items.

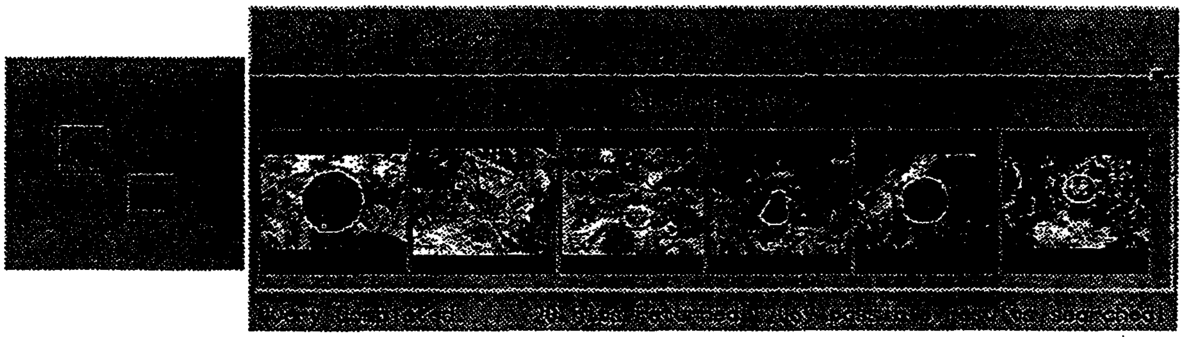

Figure 5. Example 'multi" query. Left: A visual query specification for a scene containing a red, round object (the red icon) on a green background (the green icon, where the rectangular box indicates a scene attribute). Right: The query results showing the first siz returned items.

QBIC's specialized query engine was developed as a stand alone system with its own user interface for querying image data. This architecture is similar to many systems on the market which provide content-based querying of particular datatypes (e.g., text, images, maps, molecular structures). To integrate this type of system into Garlic the user interface components must be separable from the search components. In an increasing number of these systems the search engine is accessible through published application programming interfaces (APIs), making integration as a repository feasible. However, the query formation interface is not usually accessible through an API. Thus there may be different levels of integration with Garlic. If a specialized user interface is not separable from the callable search engine, the system can either be integrated as a monolith with no exploitation of Garlic's ability to provide cross repository queries or to integrate and synchronize presentation of results, or the search engine can be integrated as a repository and other user interfaces exploited for query formation. One drawback of this latter approach is the loss of the familiar interface that a particular system provided. However, we believe the benefits of a closer integration with Garlic (and consistency of user interface when accessing multiple similar repositories) will outweigh the costs for most applications that need Garlic functionality. Thus, we are trying to borrow or develop good general query interfaces for specific types, including image.

Since QBIC, unlike most systems, actually has not only a separable but an accessible query formation interface, we take advantage of its generality to integrate it with the Garlic query/browser (Section 4.4) as the basis of our general image query interface. The search engine will be "wrapped" so that it presents itself to

Garlic as an image database manager with an object-oriented schema. In the next two subsections we discuss some of the issues involved and choices made in this integration process.

4.2.1 Integrating the QBIC Query Formation Interface

The QBIC pickers provide intuitively appealing and general mechanisms for users to specify colors, textures, and other image features. Because of this, we have chosen to integrate them so that they may be used to query non-QBIC image databases. The QBIC query formation functions will be packaged as a shared library, and the functions will interact with the user in the same way that they do in QBIC today.

It must be possible to use the feature specification structures in this library to query images in different repositories with different computations for the same feature (e.g., different shape feature vectors for the same shape). Thus, QBIC pickers will not compute a feature vector but will capture the user specification in a small image (e.g. a 100 x 100 color distribution) which can be input to the feature computation functions in any image database supporting query by content for the same feature. This also eliminates the need for client machines to have implementations for potentially expensive feature computations. The cost is that "image literals" must now be handled by Garlic's Query Services. These literals will be carefully passed "around" the system in order to minimize copying and query cost. (Similar mechanisms are used to handle long fields in relational databases today [15]).

Another requirement is that it must be possible to integrate the resulting image query within the complete user query being built by the Query/Browser. The QBIC query formation functions will therefore capture the logical expression of the user's query in a text form with references to the image literals discussed above. The text form will be a subset of the Garlic Query Language which can be pieced into the full GQL query that the Query/Browser will submit to Garlic Query Services.

The thumbnails available from QBIC in response to an image query will be displayed by the query/browser using the image display tools available at the client. These tools must support "drag and drop" protocols so that the returned images can be moved into QBIC's query formation functions to exploit the "query by example" paradigm.

4.2.2 Wrapping the QBIC Query Engine

Typical information servers, whether general purpose or domain specific (e.g., Lotus Notes, Excalibur's Electronic Filing System or ACR/NEMA DICOM Medical Image Servers), organize the data they manage under a schema that presents a model of that data to the user. Document systems compose a document from pages and then organize the documents into folders, filedrawers, cabinets, etc. Medical image servers organize tomographic images into series, series into studies and studies into sections of a patient folder. Although instances of these data objects and data collections can be added, the object and collection types in each schema are fixed by the underlying system. Furthermore, the systems support several levels of search capability through a published API. We believe this model of an information server is representative of an increasing segment of the information server market. Trends in industry standardization of domain-specific data models and in marketplace standardization of general purpose information and data management systems will further support this model. Therefore, most repository wrappers in Garlic will bridge the gap between Garlic's object-oriented model and a fixed schema in a similar modeling discipline.

However, QBIC is a research prototype, and does not have a published data schema or APIs. Instead of describing the data stored, QBIC's file-based data organization is oriented around handling image and feature vector data structures. To integrate QBIC into Garlic so that Garlic can exploit QBIC's data and search capability, the QBIC wrapper must present an object-oriented schema to Garlic, and be able to map this schema down to the file structures and call interfaces currently provided by the QBIC search engine. It is a virtue of Garlic's architecture that even in this case integration is possible.

The query engine wrapper has two parts: a model of the data in QBIC and of the predicates QBIC can apply, and code that translates between GQL queries and QBIC's call interfaces and returns results to Garlic. The model for QBIC's image data must express the relationships between, raw base images, scenes that have outlined objects in them, and thumbnails of the raw images as well as of the images with outlined objects. Although these data objects are stored as bitfiles or as records in data files in QBIC, the QBIC wrapper provides Garlic with a more integrated view. This view allows navigational access from one object to its related objects through the Query/Browser, the use of image feature queries over particular collections in a type safe manner and the incorporation of QBIC data (as Garlic objects) into Garlic complex objects (e.g., advertising campaigns, or resumes) without copying the large data objects into Garlic.

Interface definitions satisfying these requirements are given in Figure 6 There are three key interfaces (classes), one for full QBIC scenes, one for outlined objects within a scene, and the third containing the actual image (*BasePixelImage*). A *QBICScene* has pointers to the raw image and a thumbnail (both instances of *BasePixelImage*). It also has a set of pointers to objects outlined in that scene. These objects are represented by the *OutlinedObjects* interface. Again, each outlined object has pointers to the raw image, and to a thumbnail of that image in which the object is outlined. *OutlinedObjects* also point back to the *QBICScene* they occur in. Finally, the *BasePixelImage* class provides exactly the information needed to interpret the image bits faithfully, including width, height, and pixel size. Appropriate methods are provided with each interface definition to allow searching and manipulation of these classes. These interface definitions shield Garlic users from the details of how QBIC keeps track of which image features have been computed for a given scene, or a given object. It also hides the actual feature values. All of these are managed by the QBIC repository, but are only accessible to Garlic through the interface methods..

The interface definitions are exported by the wrapper and copied into Garlic structures used by Metadata Services to record schema information. They are used by Garlic Query Services during query compiliation (e.g, to ensure type safe queries) and by users and applications to examine the objects available in a Garlic database. The wrapper also exports a set of named collections. These collections are assigned identifiers by Garlic upon import and the wrapper is responsible for maintaining mappings between these identifiers and the underlying repository entities. For instance, if it is desired to make a set of *QBICScenes*, called *Wilderness_Shots*, available to advertisers, a QBIC server will register the directory containing the thumbnail files to Garlic as a collection during the wrapper creation process. QBIC will guarantee that the same set of features is computed for each *Wilderness_Shot* scene. Therefore, any feature-based search of the *Wilderness_Shot* collection can be assumed to be exhaustive by the user. The QBIC wrapper will map a Garlic OID for the *Wilderness_Shot* collection into a reference to this directory, and will map method invocations, such as the *match_image* search predicate, into the appropriate calls against the control file structures in the QBIC search engine.

The second part of the wrapper handles queries. The QBIC wrapper is passed that part of a user's query that applies to collections that are exported by QBIC. A feature of QBIC is that searches can be performed against lists of images that are subsets of the exported collections, or against an entire collection. This allows

```
interface QBICScene : persistent {
     relationship BasePixelImage original_image;
     relationship BasePixelImage original_image_thumbnail;
     relationship set <OutlinedObjects> scene_objects
                         inverse OutlinedObjects::original_scene;
     fuzzybool match_image (in QBICScene image_srch_arg);
     void QBdisplay();
      ...
}
interface OutlinedObjects : persistent {
     relationship BasePixelImage original_image;
     relationship BasePixelImage original_thumbnail_obj;
     attribute int[2] upperleft;
     relationship BasePixelImage objectmask;
     relationship QBICScene original_scene inverse QBICScene::scene_objects;
     void QBdisplay();
      ...
}
interface BasePixelImage : persistent {
     attribute int image_width;
     attribute int image_height;
     attribute float pel_size;
     attribute char[n] image_bits;
     attribute char getpel (in int x, in int y);
     attribute char[n] getimage (in int n);
     QBdisplay();
      ...
}
```

Figure 6. A Wrapper Schema for QBIC

Garlic Query Services considerable flexibility in choosing how to execute a query (Section 4.3).The query fragment sent to QBIC is represented by an abstract parse tree that has all references to Garlic objects bound to unique identifiers which the wrapper can map to underlying repository entities. Any literals needed to evaluate the query (e.g., a sketch to be matched) will also be passed. The wrapper creates an iterator, which provides the answer set (in a relevance sorted order created by QBIC) to Garlic's Runtime System. After mapping the Garlic subquery into QBIC entities and function calls, the wrapper relies on the client/server mechanisms provided by QBIC, e.g., RPC, to remotely execute the appropriate search and return the answer set. The answer set contains identifiers that can be mapped to Garlic OIDs, can be filtered and/or can have methods applied to them.

4.3 Queries over Visual (and Other) Data

Once a wrapper is defined for QBIC, QBIC data can be queried through Garlic. But the power of Garlic lies in its ability to answer queries that span multiple data types in multiple repositories. In this section we will show how queries in Garlic can combine predicates over visual and other data. To illustrate how queries are processed, we need both wrapper schemas for each repository and a global Garlic schema. We complete this set of schemas for a simple subset of our advertising example. We assume that in addition to a QBIC repository with images from magazine ads, the agency also has a text repository that stores financial reports for each campaign. The contents of this repository and the commands to create it are indicated in C++ notation in Figure 7. Suppose that the agency wants to correlate their reports with the magazine ads. They can

```
class Document {
        ....
public:
     char* title;
     char* text;
     Date date;
     int matches(char* search_expr);
}
```

```
make_doc_db /financial/documents
add_doc /financial/report1.text
add_doc /financial/report2.text
....
```

Figure 7. Text Repository Contents

use Garlic complex objects to do this. The wrapper schemas for the text repository and for the complex objects managed by the Garlic complex object repository are given in Figure 8. (The wrapper for the QBIC repository was shown in Figure 6). Notice that the text wrapper renames the *title* attribute of *Document* to *campaign*, based on the wrapper designer's knowledge of the actual documents being stored. Also, note that there is no magic involving complex objects. Once the complex object schema is defined, the complex object repository must be populated. In some cases this can be done through a query, but in our example this would have to be done by hand (unless there were some information in the document to identify the associated images, or vice versa).Finally, one possible Garlic schema for this example is given in Figure 9. This schema promotes the *campaign* attribute of the report into the *Campaign* objects themselves, so that *Campaigns* now have a name, a set of magazine ads, and a report.

```
interface Document(extent Document): persistent {
     attribute String campaign;
     attribute Date date;
     attribute String text;
     fuzzybool matches(String search_expr);
     void QBdisplay();
}
```

Figure 8. (a): Text Wrapper Schema

```
interface Campaign (extent Campaign): persistent {
     attribute String campaign_name;
     relationship Set<QBICScene> magazine_ads;
     relationship Document report;
}
```

Figure 8. (b): Complex Object Repository Schema

The Garlic Query Language extends SQL with additional constructs for traversing paths composed of inter-object relationships, for querying collection-valued attributes of objects, and for invoking methods within queries. These extensions are similar to those of other recent object query language proposals (e.g., [2], [13], [6]), including the ongoing efforts of the SQL-3 committee [14]. To get a flavor of these extensions, consider the following query, written against the Garlic schema of Figure 9:[2]

2. We are still working out the exact details of our SQL extensions. This example is provided to give the reader a feeling for what we intend, and should not be taken too literally!

```
interface Campaign (extent Campaign): persistent {
     attribute String campaign_name;
     relationship Set<Scene> mag_ads;
     relationship Document report;
}
interface Document (extent Document): persistent {
     attribute String campaign;
     attribute Date date;
     attribute String text;
     fuzzybool matches(in String search_expr);
     void QBdisplay();
         ....
}
interface Scene(extent Scene): persistent {
     void QBdisplay();
     fuzzybool match_image(in Scene sketch_arg);
         ....
}
     ...
```

Figure 9. Global Garlic Schema

```
select C.campaign_name, C.report, C.mag_ads
from Campaign C, C.mag_ads A
where (C.report.date > "1989") and
        A.match_image(SKETCH) > .5
```

This query finds the campaigns and the associated report and magazine ads for those campaigns that ran in the last five years and which had a magazine ad that resembled a particular image (for example, a user-drawn sketch). This would be useful for those situations in which the ad executive remembers roughly what a particular ad looked like and when it was run, but not the details of the campaign. The query illustrates several of Garlic's object-oriented SQL extensions. First, it contains a number of path expressions. Second, it contains an invocation of the *match_image*() method of the *Scene* object. This method passes QBIC a literal representing the sketch in an appropriate form for QBIC (this may have been produced visually by a sketch picker), and returns a number indicating the "goodness" of the match. Finally, *C.mag_ads* in the select clause illustrates the retrieval of an unflattened set.

To answer this query, Garlic first translates it into an internal representation which reflects the query's semantics. Each operation is then re-written in terms of the underlying wrapper schemas, using the Garlic metadata. Next, Garlic decomposes the query into a plan containing a number of smaller queries, each of which can be answered by a single repository. The plan also specifies how the results of each subquery should be combined to form the final answer. For example, one possible plan for our query would be to ask the text wrapper for the *oids* of reports written after 1989, then ask the complex object repository for the *oids* of the magazine ads associated with these reports, then probe QBIC with the list of ad *oids* to see if those ads match the sketch sufficiently closely, and finally, get the report title (campaign name) associated with the document *oid* of the surviving campaigns. Other plans are certainly possible, and it would be up to the optimizer to choose among them based on its estimates of cost.

In Garlic's distributed environment the issue of optimization is very important. The amount of work that each server does in order to handle its part of the overall query can vary greatly, from efficient range searches on a primary key in a relational database, to the costly computation of feature vectors followed by the computation of an expensive distance measure against an entire collection of images in QBIC. Ideally, Garlic would sequence the data system accesses in order to exploit parallelism and the special functions that a server provides (e.g., relevance sorted answer sets) while minimizing potentially wasted time and expense at the servers and in the Garlic system itself. Optimization will require the specification and use of several new pieces of information. We need computational models of feature calculations and distance measures in order to distinquish between the costs of different feature predicates applied within QBIC. Selectivity factors that can aid in predicting the amount of data returned by a similarity query are also needed. Finally, models must be created that can reflect the existence of special purpose indexing structures, e.g., multi-dimensional indexes for feature vectors, in their estimates of a similarity query's cost. These will all be captured in the descriptive part of a repository wrapper for use by Garlic's Query Services. In addition, Garlic will maintain information on processor speeds, I/O rates and communication costs for its installed servers and networks, in the tradition of relational optimizers.

It is the responsibility of each repository wrapper to convert its individual subplan into a form the underlying repository can understand -- either one or more queries in that repository's query language, or a sequence of calls to its native search API. The wrappers will execute their subplans in a demand-driven fashion under the control of the Garlic runtime system, returning a stream of values to Garlic for any final processing.

This final processing may involve joins, projections or restrictions, as in any middleware database system. However, Garlic has an additional challenge: to reconcile the different query semantics of its various repositories. While in database management systems data items are returned if and only if predicates are true, QBIC and other repositories managing multimedia data return data items in order of "closeness" to a given predicate. We are currently developing a set of SQL extensions and query processing algorithms to support queries that involve both exact and approximate search criteria. This work involves introducing into SQL the notion of graded sets, in which each object is assigned a number between 0 and 1 for each atomic predicate; this number represents the degree to which the object fulfills the predicate, with 1 representing a perfect match. Boolean combinations of predicates can then be handled using the rules for combining predicates in fuzzy logic [27]. To enable query writers to specify the desired semantics, GQL permits the specification of the number of matching results to be returned and whether or not rank-ordering (rather than an attribute-based sort order, or an arbitrary order) is desired for the query's result set. We are also devising new query processing algorithms that will produce the best N results efficiently, without materializing every intermediate result item that matches to any degree at all.

4.4 Visual Query/Browse in Garlic

The purpose of the Garlic Query/Browser component is to provide end users of the system with an easy and highly visual way to access and manipulate the data in a Garlic database, as the typical end user will not normally want to write GQL queries. As its name implies, the Query/Browser will provide support for two basic data access functions, namely querying and browsing. However, unlike existing interfaces to databases, the Query/Browser will allow users to move back and forth seamlessly between querying and browsing activities, using queries to identify interesting subsets of the database, browsing the subset, querying the contents of a set-valued attribute of a particularly interesting object in the subset, and so on.

The Query/Browser will support exploration of a Garlic database by allowing users to browse through the contents of Garlic collections (via next/previous buttons or scrolling) and to traverse relationships by clicking on (selecting) objects' reference attributes. When multiple related objects are being simultaneously displayed, synchronous browsing will be implied (*a la* [18], [1]). Consider what an advertising executive might do to find the campaign she wants without writing any GQL. She might start by just browsing through campaigns. Figure 10a shows the screen after she has chosen to browse the *Campaign* collection. By clicking on the *report* field, she can see the associated report (10b). Since the *Document* interface has a *QBdisplay* method, the Query/Browser invokes that method to display the report. (For purposes of this paper, we assume that *QBdisplay* is a distinguished method, provided to allow the Query/Browser to display objects of that type). Similarly, selecting *mag_ads* will show images of the ads (10c), using *Scene*'s *QBdisplay* method. Clicking *next* on the *mag_ads* window will browse through the ads for this campaign. *Next* in the *Campaign* window (10d) will move to the next campaign, and the report and ads related to that campaign.

The Query/Browser will support querying via a "query-by-graphical-example" paradigm, extending the

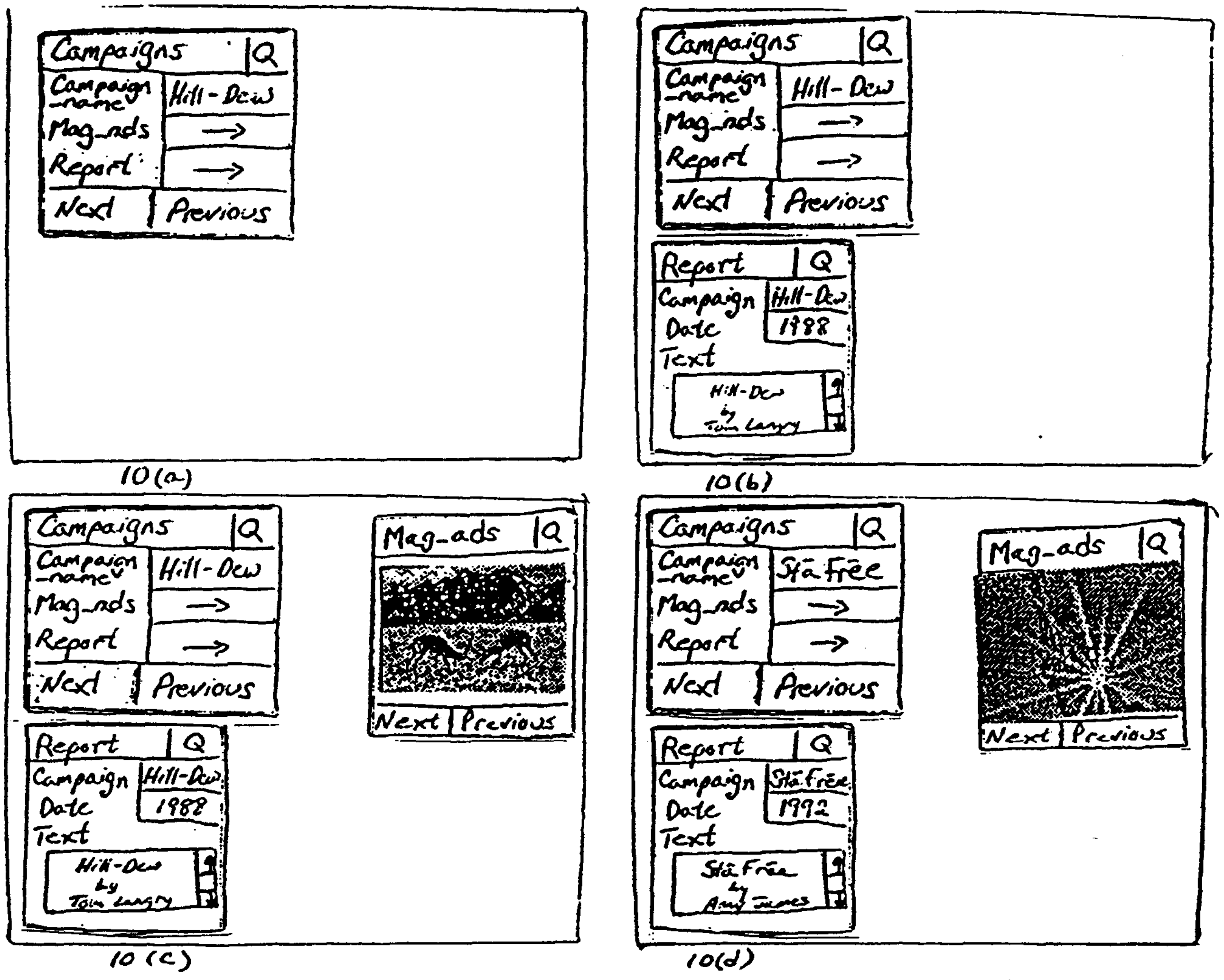

Figure 10. Browsing using the query/browser

well-known query-by-example paradigm [28] for use in formulating queries over an object database. Suppose our account exec, tired of browsing, decides to specify the query we looked at above. She clicks on the query button in the top corner of the *Campaigns* window, and then clicks on the fields she wishes to restrict (Figure 11a). In Figure 11b, she has specified the predicate on *reports (date>1989)* and has chosen to do a query by sketch on *mag_ads*. This results in the appearance of a scene picker, with which she sketches the scene she remembers (Figure 11c). When she's done specifying predicates, she selects the *DO_IT* button to

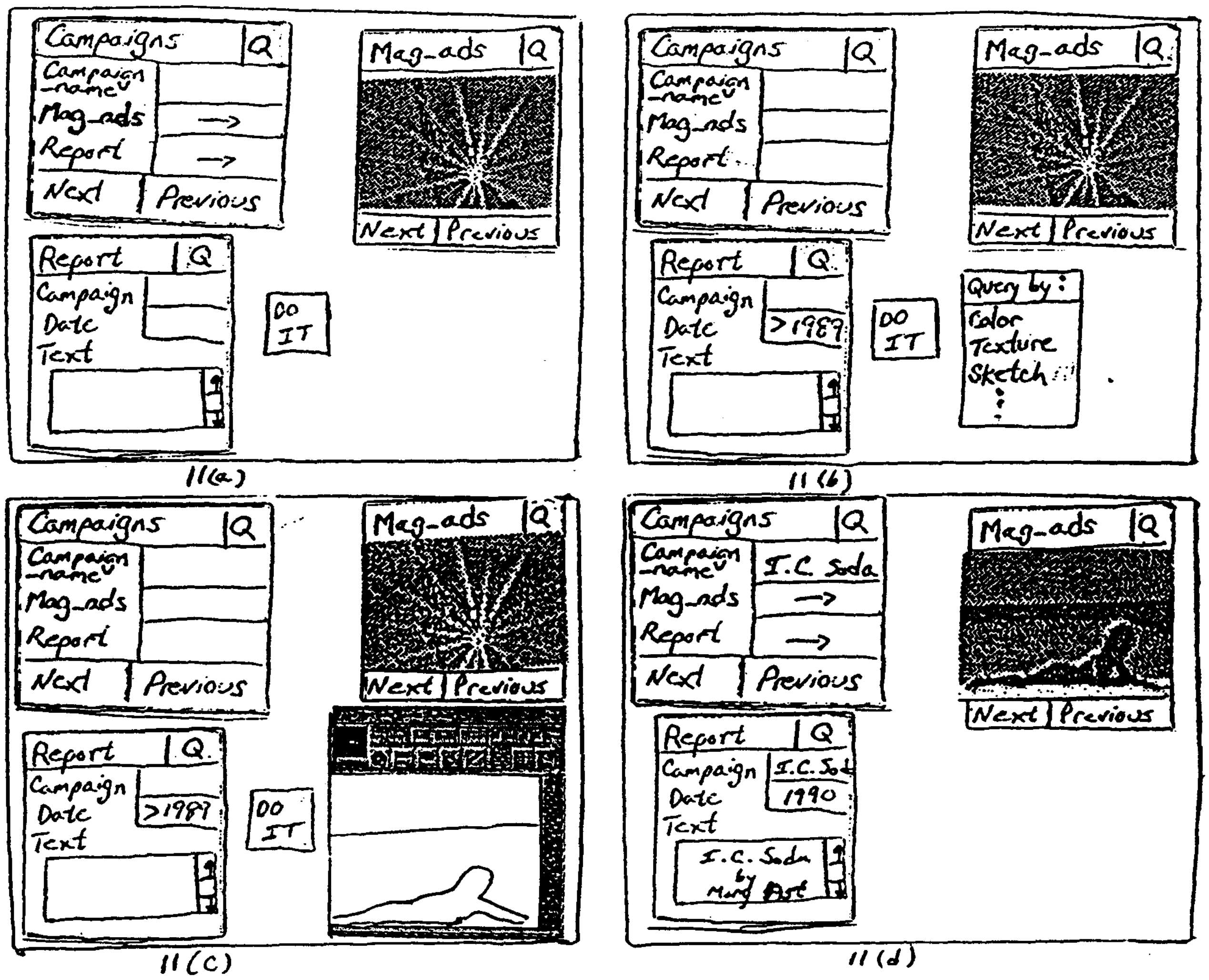

Figure 11. Querying in the Query/Browser

cause the query to execute. She can then browse the results (Figure 11d), with the query's constraints remaining active until explicitly cleared.

In addition to smoothly combining querying and browsing, the Garlic Query/Browser will also provide other useful features for exploring and manipulating the contents of a heterogeneous multimedia data collection. First, the objects on the display at any given time will be active objects -- the Query/Browser will remember their Garlic identities and will provide a graphical means of obtaining a list of their available methods and requesting that one of the methods be applied to the object of interest (prompting for method arguments if needed). Second, clicking on "query" followed by a multimedia (e.g., image, audio, video, or text) attribute of a displayed object will result in the display of a type-specific picker (or set of pickers) to support the construction of a media-specific predicate on that attribute of the object, as discussed in Section 4.2.1. The Query/Browser will contain a number of such pickers to support the graphical specification of content-based multimedia predicates. In time, the Query/Browser will become still more sophisticated, supporting the graphical definition of end-user views. Ultimately, we believe that good support for customizing the browser's behavior with respect to a given Garlic database and Garlic user may lead to a new paradigm for visual application development, at least for applications of a "browsy" (i.e., navigational) nature.

5 Conclusions, Status and Future Work

We have presented an overview of the Garlic project at the IBM Almaden Research Center, the goal of which is to build a heterogeneous multimedia information system (MMIS) capable of integrating data from a variety of traditional and non-traditional data repositories, and allowing query by content of any type of data. We described the overall architecture for the system, which is based on repositories, repository wrappers, and the use of an object-oriented data model and query language to provide a uniform view of the disparate data types and data sources that can contribute data to a Garlic database. As we explained, a significant focus of the project is support for repositories that provide media-specific query capabilities. We described QBIC, a system that provides query by image content, and showed how QBIC could be integrated into Garlic so that queries might range over data in this and other repositories simultaneously. We also described exploratory access to Garlic by end users via the Garlic Query/Browser.

The Garlic project was initiated in early 1994. Our current target is to have an initial "proof of concept" prototype running (or at least limping) by the end of 1994. This prototype will be demonstrated by a simple application involving data that spans a relational DBMS (DB2 C/S), a QBIC repository, and a full text search engine. The goal of the first prototype is to understand the nature of wrappers, the challenges involved in query translation and processing, and the efficacy of the query/browser as an end-user window into a collection of multimedia data.

In the longer term, we expect the Garlic project to lead us into new research in many dimensions, including object-oriented and middleware query processing technologies, extensibility for highly heterogeneous, data-intensive environments, database user interfaces and application development approaches, and integration of exact- and approximate-matching semantics for multimedia query languages. There are also many interesting, type-specific issues, such as what predicates should be supported on image and video data, how to index multimedia information, how to support similarity-based search and relevance feedback, and what the appropriate user interfaces are for querying particular media types. We believe that significant challenges exist in each of these areas, and that solutions must be found to meet the emerging demand for large-scale multimedia data management.

6 Acknowledgments

We would like to thank Rakesh Agrawal for his input in the start-up phase of the Garlic project; he contributed significantly to our vision for both the project as a whole and the query/browser in particular. John McPherson and Ashok Chandra have been particularly supportive of our efforts throughout; we thank them for their encouragement and many suggestions. Many others contributed to the definition of the Garlic project, including: Kurt Shoens, K.C. Lee, Jerry Kiernan, Peter Yanker, Harpreet Sawhney, David Steele, Byron Dom and Markus Tresch.

7 References

[1] R. Agrawal, N. Gehani, And J. Srinivasan, "OdeView: The Graphical Interface to Ode", Proc. ACM SIGMOD Conference, Atlantic City, NJ, May 1990.

[2] F. Bancilhon, S. Cluet, and C. Delobel, "A Query Language for the O2 Object-Oriented Database System", Proc. DBPL Conference, Salishan Lodge, Oregon, June 1989.

[3] R. Cattell, ed., "The Object Database Standard: ODMG-93 (Release 1.1)", Morgan Kaufmann Publishers, San Francisco, CA, 1994.

[4] Carey et al., Garlic paper

[5] T. Conners, W. Hasan, C. Kolovson, M. Neimat, D. Schneider, and K. Wilkinson, "The Papyrus Integrated Data Server", Proc. 1st PDIS Conference, Miami Beach, FL, December 1991.

[6] S. Dar, N. Gehani, and H. Jagadish, "CQL++: A SQL for a C++ Based Object-Oriented DBMS", Proc. EDBT Conference, Vienna, Austria, 1992.

[7] A. Elmagarmid and C. Pu, eds., Special Issue on Heterogeneous Databases, ACM Comp. Surveys 22(3), September 1990.

[8] D. Fang, S. Ghandeharizadeh, D. McLeod, and A. Si, "The Design, Implementation, and Evaluation of an Object-Based Sharing Mechanism for Federated Database Systems", Proc. IEEE Conf. on Data Eng., Vienna, Austria, April 1993.

[9] K. Hirata and T. Kato, "Query by Visual Example", Advances in Database Technology EDBT '92, Third International Conference on Extending Database Technology, Springer-Verlag, Vienna, Austria, March 1992.

[10] D. Hsiao, "Federated Databases and Systems: Part I -- A Tutorial on Their Data Sharing", VLDB Journal 1(1), July 1992.

[11] M. Ioka, "A Method of Defining the Similarity of Images on the Basis of Color Information", IBM Tokyo Research Lab, Technical Report RT-0030, 1989.

[12] T. Kato, T. Kurita, N. Otsu and K. Hirata, "A Sketch Retrieval Method for Full Color Image Database", International Conference on Pattern Recognition (ICPR), IAPR, The Hague, The Netherlands, pp. 530--533, September 1992.

[13] W. Kim, "A Model of Queries for Object-Oriented Databases", Proc. VLDB Conference, Amsterdam, the Netherlands, August 1989.

[14] K. Kulkarni, "Object-Oriented Extensions in SQL3: A Status Report", Proc. ACM SIGMOD Conf, Minneapolis, MN, May 1994.

[15] Lehman and Lindsay VLDB Long Field MGR

[16] R. McConnell, R. Kwok, J. C. Curlander, W. Kober and S. S. Pang, "$\Psi - S$ Correlation and Dynamic Time Warping: Two Methods for Tracking Ice Floes in SAR Images", IEEE Transactions on Geoscience and Remote Sensing", 29:6, pp. 1004-1012, November, 1991

[17] M. Miyahara and Y. Yoshida, "Math. Transform of *(R,G,B)* Color Data to Munsell *(H,V,C)* Color Data", Vis. Comm. and Image Proc., SPIE, Vol. 1001, pp. 650-657, 1988.

[18] A. Motro, A. D'Atri, and L. Tarantino, "The Design of KIVIEW: An Object-Oriented Browser", Proc. 2nd Int'l. Expert Database Systems Conference, Tysons Corner, VA, April 1988.

[19] W. Niblack, R. Barber, W. Equitz, M. Flickner, E. Glasman, D. Petkovic, and P. Yanker: "The QBIC Project: Querying Images by Content Using Color, Texture and Shape", Proc. SPIE, San Jose, CA, February 1993.

[20] A. Pentland, R. Pickard, and S. Scarloff, MIT Media Lab: "Photobook: Tools for Content Based Manipulation of Image Databases", Proc. SPIE, San Jose, CA, 1994.

[21] R. Rosenberg and T. Landers, "An Overview of MULTIBASE", in Distributed Databases, H. Schneider, ed., North-Holland Publishers, New York, NY, 1982.

[22] G. Salton, "Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer", Addison-Wesley Publishers, 1989.

[23] M. Shan, "Pegasus Architecture and Design Principles", Proc. 1993 ACM SIGMOD Conference, Washington, DC, May 1993.

[24] M. J. Swain and D. H. Ballard", "Color Indexing", International Journal of Computer Vision, 7:1, pp. 11-32, 1991.

[25] H. Tamura, S. Mori and T. Yamawaki, "Texture Features Corresponding to Visual Perception", IEEE Transactions on Systems, Man, and Cybernetics, SMC-8:6, pp. 460-473, 1978.

[26] M. Ubell, "The Montage Extensible Datablade Architecture", Proc. ACM SIGMOD Conference, Minneapolis, MN, May 1994.

[27] H. J. Zimmermann, Fuzzy Set Theory and its Applications, Kluwer Academic Publishers, Boston, MA, 1990.

[28] M. Zloof, "Query-By-Example: A Data Base Language", IBM Systems Journal 16(4), 1977.

[29] Understanding GIS -- The ARC/INFO Method, ESRI Inc. (1990).

[30] "SPANS: SPatial ANalysis System", TYDAC Technologies: Corporate Overview (1990).