Int J Digit Libr (1999) 2: 178–189

INTERNATIONAL JOURNAL ON
Digital Libraries

Drag-and-drop multimedia: an interface framework for digital libraries

Lawrence D. Bergman[1], **Jerre Shoudt**[2], **Vittorio Castelli**[1], **Chung-Sheng Li**[1], **Loey Knapp**[2]

[1] IBM T.J. Watson Research Center, 30 Saw Mill River Rd., Hawthorne, NY 10532, USA;
E-mail: {bergman,vittorio,clsi}@watson.ibm.com
[2] IBM Government Systems, 6300 Diagonal Highway, Boulder, CO 80501, USA;
E-mail: {jschoudt,knappl}@us.ibm.com

Received: 15 December 1997/Revised: June 1999

Abstract. In this paper, we describe a new interface for querying multimedia digital libraries and an interface building framework.

The interface employs a drag-and-drop style of interaction and combines a structured natural-language style query specification with reusable multimedia objects. We call this interface DanDMM, short for "drag-and-drop multimedia". DanDMM interfaces capture the syntax of the underlying query language, and dynamically reconfigure to reflect the contents of the data repository.

A distinguishing feature of DanDMM is its ability to synthesize integrated interfaces that incorporate both example-based specification using multimedia objects, and traditional techniques including keyword, attribute, and free text-based search.

We describe the DanDMM-builder, a framework for synthesizing DanDMM interfaces, and give several examples of interfaces that have been constructed using DanDMM-builder, including a remote-sensing library application and a video digital library.

Key words: Digital library – User-interface – Multimedia – Drag-and-drop – Query interface

1 Introduction

In this paper we present a new user-interface model designed for formulating queries to multimedia information repositories. Many digital libraries contain heterogeneous data from a wide variety of sources. Contents may include manuscripts, images, aerial photographs, maps, remote sensing data, scientific records, and others [1–4]. Search tasks range from simple keyword-based searches (e.g., "find all paintings by Picasso from 1932") to complex evaluations involving multiple object types with spatial and temporal relationships (e.g., "find all soil records for semi-arid rangeland adjacent to 'large' lakes and downwind of the Brush Creek fire"). It is the latter form of query that we are particularly interested in supporting.

There are several ways in which a user can specify the desired content to be retrieved from multimedia repositories. Example-based interfaces, including those of QBIC [5], Virage [6], and VisualSeek [7], allow a query to be constructed by providing one or more example images. The search engine uses a set of matching operators to search a database of features pre-extracted from items in the repository. The user typically can adjust parameters that control the weights of different features (for example, shape, color, texture in the case of images) during the matching process. Although this is a fairly intuitive way to specify a query, it often fails to capture a user's intent. Some of these systems provide for more extensive semantic specification, for example, VisualSeek supports the definition of simple spatial relations between patches of colors or of texture in an image through the SaFe interface. In all these systems, however, the level of semantic specification is fairly low.

Natural language interfaces for querying databases have been under development for a number of years [8–10]. Although these interfaces are typically easy to use, they tend to have limited applicability due to the system's inability to capture precise meanings from natural language. In addition, much of the available multimedia material in digital libraries is accompanied by little or no semantic labeling describing its content.

Query languages provide a more precise specification but tend to be significantly more difficult to use. Text-based languages such as SQL [11] allow very precise specification of complex queries, but are typically employed only by expert users. Graphical query languages [12] attempt to make querying more accessible, but also tend to have a steep learning curve, and to provide limited sup-

port for the wide variety of datatypes and relationships required for querying a multimedia repository.

Menu or form-based interfaces are not adequate for multimedia repositories; direct manipulation interfaces containing multimedia objects are required [13]. Several graphically-oriented direct manipulation interfaces for digital libraries have been developed including DLITE [14] and Delaunay [15]. These use drag-and-drop techniques to specify queries and result presentation, but do not address the issue of expressing query semantics beyond simple attribute specification.

In order to support a rich set of semantic operators, a query language interface should be combined with an example-based interface to draw on the strengths of both. A recent effort in this direction is the IFQ system [16], where queries are specified using a graph that represents query objects and relationships, with embedded image or video examples.

Our user-interface model, drag-and-drop multimedia (DanDMM, rhymes with "tandem") is an attempt to combine the best features of the existing interfaces, and to augment them to provide a highly flexible general-purpose environment for query construction that combines both example-based specifications and structured query language in a seamless fashion.

We have adopted an English-like query pattern, with phrases representing actions and relationships and with objects which are distinguishable elements in the repository. These objects can be, for example: textual fields (e.g., "author" or "source"); spatial features, such as fires, roads, or land cover types; images; video or audio clips.

Multimedia objects can be easily manipulated by the interface to dynamically construct dictionaries, and can be seamlessly embedded into phrases. Nested queries can also be specified, either directly, or by assigning identifiers to subqueries and using these names in the construction of more complex queries. Phrases constructed with the DanDMM interface can contain optional syntax, which allows advanced users to access features that might be confusing for novices.

The DanDMM interface style enables the construction of arbitrarily long and complex query sentences and provides an explicit view of all portions of the query. It supports any context-free query language.

An important aspect of providing user-interface tools for digital libraries is support for rapid prototyping of interfaces, particularly interfaces that can dynamically reflect the contents of the library and reconfigure information in the interface based on these contents. For example, when a user selects a particular parameter to be specified, the interface might display the list of its available values, or change the interactor to an appropriate type (e.g., a slider for a numeric value, a choice box for enumerated text). We have created a builder framework for constructing DanDMM interfaces based on specifying the structure of the query language to be supported. The interface specification can include dynamic behavior that permits the interface to reconfigure based on repository contents, or on user actions.

DanDMM interfaces are distinguished from prior interfaces by the flexible way in which multimedia objects are incorporated directly into queries. This provides for rapid prototyping of frontends for digital libraries that support example-based queries, as well as more traditional query operators.

This paper focuses on two main topic. The first is the particular style of drag-and-drop interface that we call DanDMM. Note that DanDMM is not a specific interface – it is a type or style of interface incorporating multimedia objects into natural language style phrases and subphrases, used to composed queries via drag-and-drop operations. Throughout the paper we will use the terms "DanDMM interface" and "DanDMM-style interface" interchangeably – both mean an interface built using the DanDMM interface builder, and embodying the drag-and-drop natural language style with embedded multimedia described in this paper.

The paper also describes the framework that we have constructed for implementing DanDMM-style interfaces. The DanDMM-builder framework (or DanDMM interface builder) allows one to readily construct DanDMM interfaces through specification of a configuration file, including information about query language syntax, interface layout, and communication between interface and application.

The paper is organized as follows: Section 2 contains a description of the DanDMM interface functionality and of its components. Section 3 provides a detailed example of the use of the interface. Section 4 describes the framework for quickly prototyping DanDMM-style interfaces, discusses the configuration file used to specify these interfaces, and concludes with an example of such an interface definition file. Section 5 is devoted to the architecture of the current implementation of the DanDMM interface builder, which is currently available as a set of Java classes. DanDMM-style interfaces have been built as front ends for several multimedia repositories, some of which are described in Sect. 6. Future work is discussed in Sect. 7 and the conclusions are in Sect. 8.

2 The DanDMM interface

2.1 Interface functionality

Our user-interface model, drag-and-drop multimedia, is aimed at providing a highly flexible, general-purpose environment for query specification that combines both structured query language and example-based constructions in a seamless fashion.

Queries are expressed as English-like patterns, which consist of a combination of phrases and objects. Phrases represent query prototypes, actions, and relationships between objects. Objects are distinguishable elements in the

repository and can be: the usual textual fields; multimedia entities, such as images, video or audio clips; spatial features, such as fires or roads; or attributes and derived quantities of the above, such as color, texture, duration, size, shape, land-cover types, etc.

The interface was designed to meet a number of criteria:

- Query constructs should be capable of expressing the full power of the underlying search engine including specification of object attributes, types, relationships, and constraints;
- Query constructs should support any context-free query language;
- Query constructs should be human-readable in a natural-language style;
- The interface should automatically enforce syntactic, and, where possible, semantic constraints, helping the user to formulate reasonable queries;
- The interface should dynamically reconfigure to reflect the contents and structure of the data repository;
- Inclusion of multimedia objects in example-based queries should be supported;
- The interface should facilitate the process of query refinement by providing a set of reconfigurable and reusable query components, and by allowing query results to be incorporated into query statements;
- The interface should provide optional syntax in order to support both basic operations by entry-level users as well as more advanced operations. This optional syntax should be provided as in-line expansions of the base syntax in a natural fashion.

To meet these objectives we have combined the power of visual programming with natural-language structure through the use of drag-and-drop techniques.

The DanDMM interface provides draggable entities representing objects and phrases, i.e., query elements, which can be dropped onto a palette or into placeholders within other phrases. Queries, objects, attributes, relationships, and constraints are represented as specific phrases. Phrases can be combined with each other using drag-and-drop operations to build up query sentences. As described in more detail in Sect. 4, this style enables the construction of queries from any context-free grammar.

The interface imposes constraints on the construction process, disallowing drops that would result in syntactically incorrect combinations of objects and relationships. Draggable components include text items, standard interface widgets such a type-ins and choice boxes, and multimedia objects. Drag-and-drop interface components may be predefined, defined by the user as subqueries, or may be objects returned from a previous query. Queries constructed with the interface appear as a English-language phrases, with embedded multimedia objects. Arbitrarily long and complex query sentences can be constructed while maintaining an explicit view of all their portions.

Interface elements may also dynamically reconfigure based on user actions and on application (e.g., database) objects. Available dynamic behaviors include

- Changes to a phrase based on user manipulation of a widget within the phrase, e.g., adding items typed into a type-in area to a choice list within the same phrase;
- Changes to a phrase using information supplied by application objects, e.g., updating a choice list to reflect attribute values available from a application repository;
- Marshalling of information contained within a phrase for delivery to an application object.

Query reuse and refinement are well supported by the drag-and-drop interface – portions of query phrases may be removed or moved to other queries, allowing for both modular construction of sentences and phrases, and straightforward specification of nested queries. The user can rely on results of previous queries to modify and refine the current query. For instance, if the query engine supports relevance feedback, previous results can be incorporated in the current query definition as positive or negative examples.

To simplify the construction of complex, nested queries, the interface provides a mechanism for naming subqueries or objects. When a subquery is named, it becomes a new entity type that can be used in place of a subquery clause. Similarly, new object types may be created (e.g., a "forest" definition). When the query is issued, subquery and object names are expanded into their full definitions, before being sent to the server.

Optional syntax is provided by in-line expansion of query phrases when special add/delete widgets are activated.

2.2 Interface components

In this section we will describe the basic components and capabilities of a DanDMM-style interface. These will be illustrated by means of an example in Sect. 3.

The interface as seen by the DanDMM user consists of a set of menus on the left-hand side of a phrase palette, within which queries are constructed (Fig. 1). A list box contains a series of items, each of which can be selected by clicking on it with a mouse button, dragged with the mouse button held down, and then dropped onto the palette by releasing the mouse button. A menu item may be a single atomic object such as a text item, a user interface widget, such as a choice list or a button, or a multimedia object such as an image. Alternately, a menu item may consist of a phrase, drawn on the palette as a rectangular panel containing a linear sequence of atomic objects.

The most important widget is one known as a *receptacle*. A receptacle, drawn as an empty colored rectangle within a phrase, represents an incomplete portion of

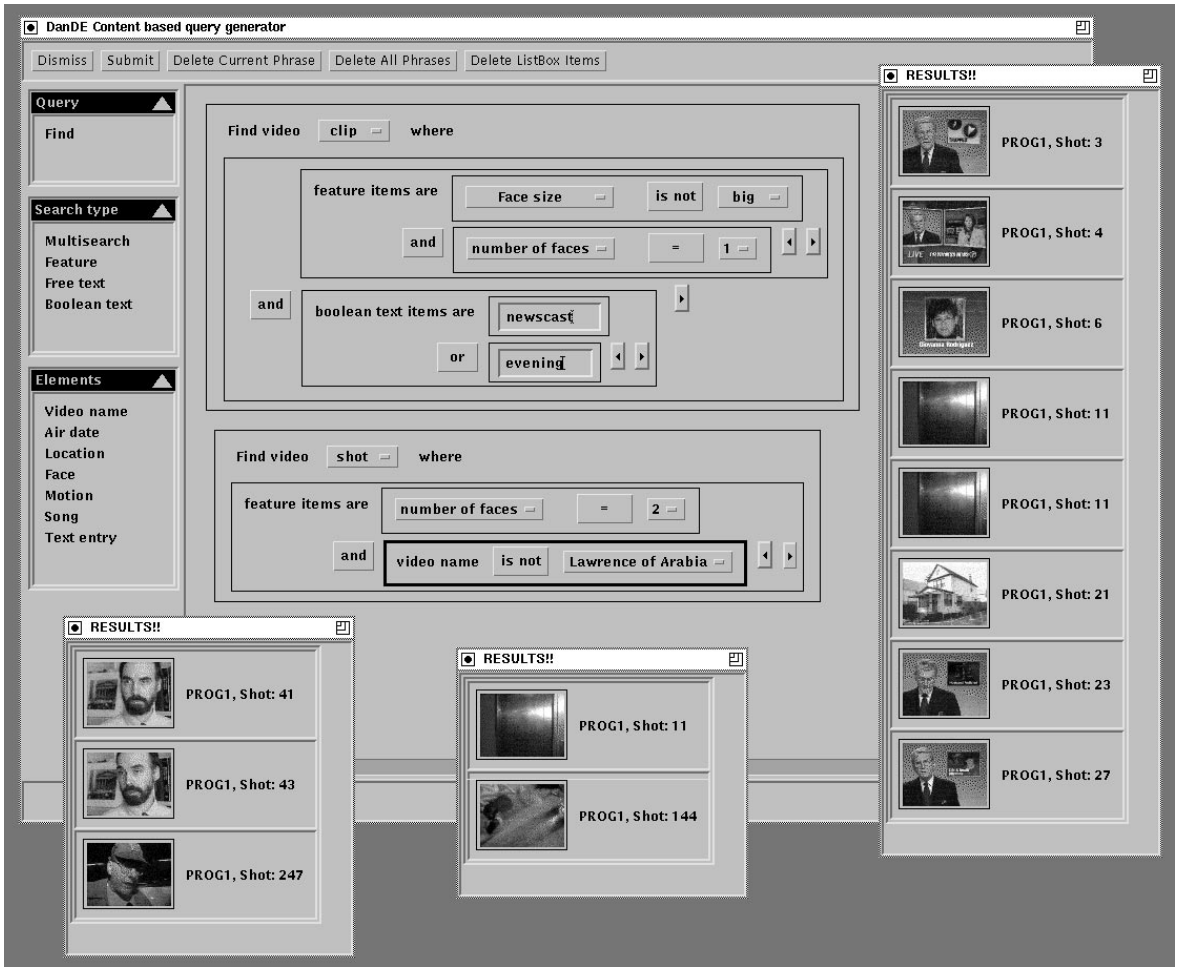

Fig. 1. The DanDMM interface showing both query phrases, and results windows. Three menus, labeled `Query`, `Search type` and `Elements` are visible on the left. The phrase palette contains two queries. Three results of the top query are shown near the *bottom-left corner*. Results from previous queries are also displayed at the *bottom* and on the *right-hand side*

a query phrase. The receptacle is "filled in" by dragging a phrase from a menu or from the palette and dropping it into the receptacle. When a receptacle receives a drop, it expands to the size of the component being dropped, and triggers a resize of the enclosing phrase. With multiply nested sub-phrases, resize operations apply to all appropriate levels of nesting.

Figure 2 shows a DanDMM phrase containing both filled and unfilled receptacles. Note that queries constructed in this fashion are composed of hierarchically nested sets of subcomponents.

Multimedia objects (such as images) for drag-and-drop operations can be imported from a separate digital library application. Additionally, DanDMM has a limited ability to directly display query results from a digital library application. Figure 1 shows three result windows containing thumbnail images produced by library queries. Multimedia objects in these DanDMM result windows can be dragged directly unto the DanDMM palette or dropped into phrase receptacles.

Besides receptacles, several other widgets are available in the current implementation. These are: text labels, type-in fields, toggle buttons, choice (pull down) lists, and add/delete buttons. Multimedia objects currently include images; video clips are represented by still images, with plans for adding real-time video support in the near future. Figure 3 shows a choice list in use, as well as text labels, a type-in field, and an empty receptacle.

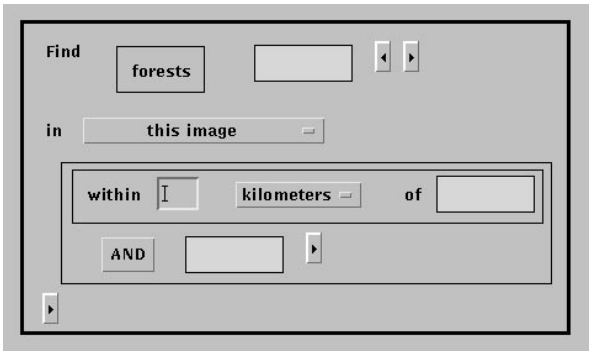

Fig. 2. A query phrase showing both filled and unfilled receptacles

Add/delete buttons are special widgets that provide support for optional syntax. These buttons allow phrases

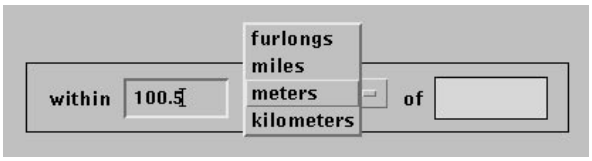

Fig. 3. Widgets within a query phrase: a type-in field, a choice list and a receptacle

to expand to an arbitrary number of predefined widget sets. The widget is initially displayed as an "add" button containing a rightward-pointing triangle as seen at the bottom of the phrase shown in Fig. 2. When the button is pressed, a set of widgets is added to the panel, and a "delete" button (containing a leftward pointing triangle) is placed to the right of the newly added items. The "add" button is repositioned to the right of the new "delete" button. Figure 2 has both an add button and a delete button to the immediate left of it at the right end of the first row of widgets. Pressing the "delete" button causes the set of widgets associated with the button (including the "delete" button itself) to be removed from the panel.

3 Usage scenario

In this section we will illustrate the use of a DanDMM-style interface by means of an example. It is important to note that we are describing use of an instance of a DanDMM-style interface. This demonstrates one of the many possible interfaces that might be constructed for a wide variety of digital library applications. Each interface might support quite different query capabilities, although each will have a look and feel similar to that described in this example.

The example provided in this section is taken from a remote-sensed imagery digital library application we have constructed. The application supports a variety of query operators using both pre-defined entities (such as forests, and agricultural areas), as well as entities defined by the end-user. New entities are defined using sample image clips and constraints. In this example, we show the use of an image clip to define a new entity (lake), and then the use of that entity in constructing queries. The sample application is described in more detail in Sect. 6.

Figure 4 shows a portion of the satellite image browser application. We see an image, with a subregion selected using a bounding box. Figure 5 shows a section of the DanDMM interface for this application. The menu items include an operator called "Add Image Object" which is used to import the selected subregion from the image browser. This item is selected with the mouse, and the "Submit" button is pressed to add the image clip. Although queries are normally built using drag-and-drop operations, simple operators that don't require parameter specification (such as "Add Image Object") can be simply executed from the menu. Image clips that are imported in this manner become draggable entities, and can be used in query construction.

Figure 6 shows a phrase that is used to construct an object definition. This phrase was dragged onto a query construction area (known as the "phrase palette") from one of the menus. In this case, the menu item "Define simple object" seen in Fig. 5 was dragged onto the phrase palette. When dropped, it automatically expanded into the phrase seen in Fig. 6. We have typed the name of the object to be defined, "lake", into a type-in area. Notice

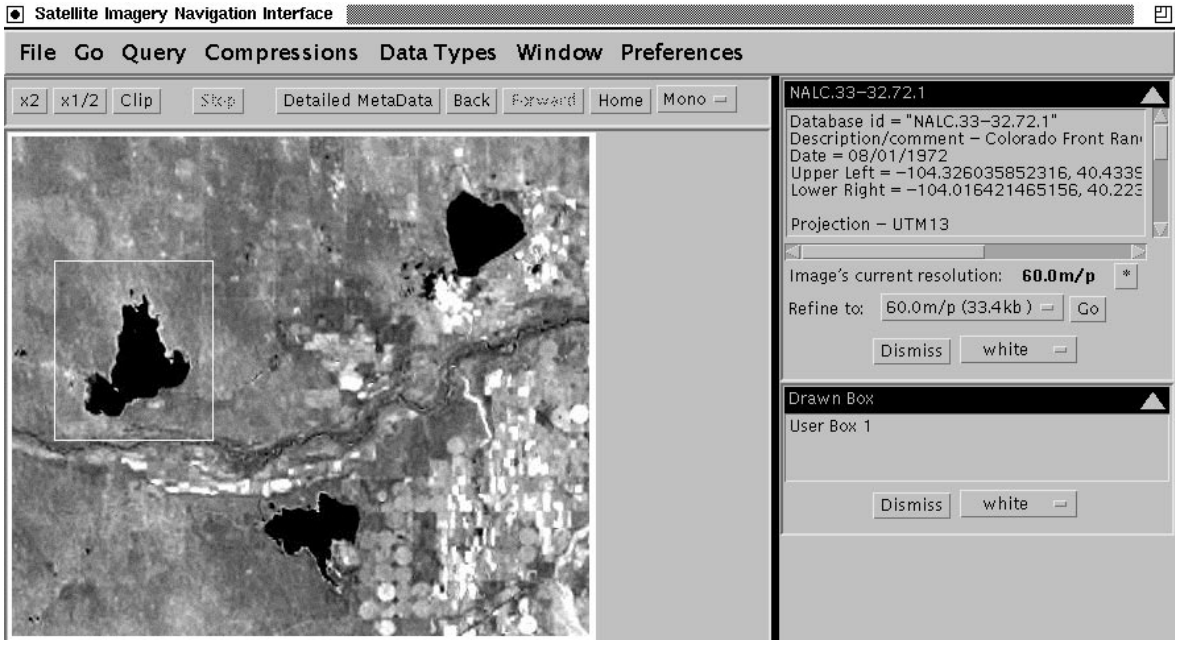

Fig. 4. A satellite library application showing a portion of an image selected for use in query construction

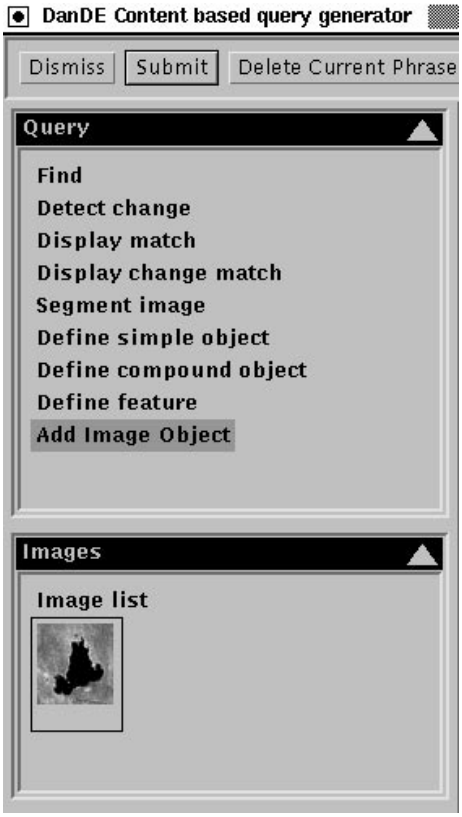

Fig. 5. A section of the DanDMM menus for the satellite library application. An "Add Image Object" operator has been used to import an image clip from the main application

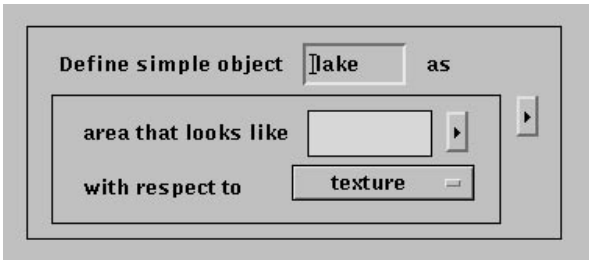

Fig. 6. A partially completed phrase for creating a user-defined object definition

that there is a "blank" to be filled in. Such blanks, known as "receptacles" are a key to the power of DanDMM interfaces. Receptacles are filled using drag-and-drop operations and accept only syntactically correct entries.

Figure 7 shows the completed object definition phrase. The previously imported image clip has been dragged from the menu into the phrase receptacle, to complete the definition. This definition specifies that the sample image is to be used to provide a prototype for image features (in this case land use category) to be matched when "lake" is requested. Once the phrase has been completed, the object definition is added to the list of available entities by pressing the "Submit" button. Figure 8 shows a DanDMM menu containing the newly defined "lake" object, as a result of submitting the definition phrase.

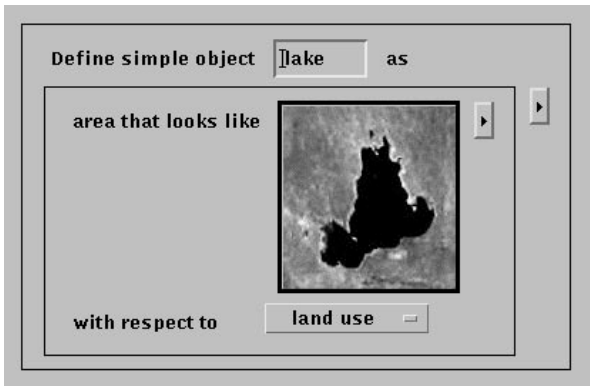

Fig. 7. Completed phrase for creating a user-defined object definition

This menu item is a draggable entity, available for building queries.

Fig. 8. Submitting the lake definition phrase adds a "lake" entity to a DanDMM menu

Figure 9 displays a "Find" phrase (the "Find" phrase was selected from the menu seen in Fig. 5). This phrase specifies a query to the satellite digital application server. Note that there are two empty receptacles in this phrase. Figure 10 shows the result of dragging the "lake" item from the menu into the phrase. Also shown in this figure

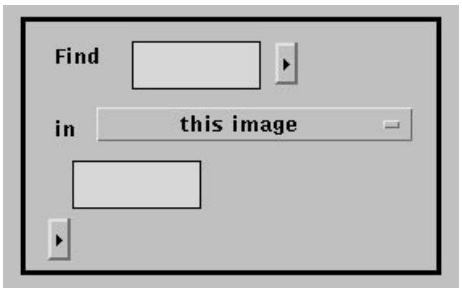

Fig. 9. A "Find" query phrase with empty receptacles yet to be filled

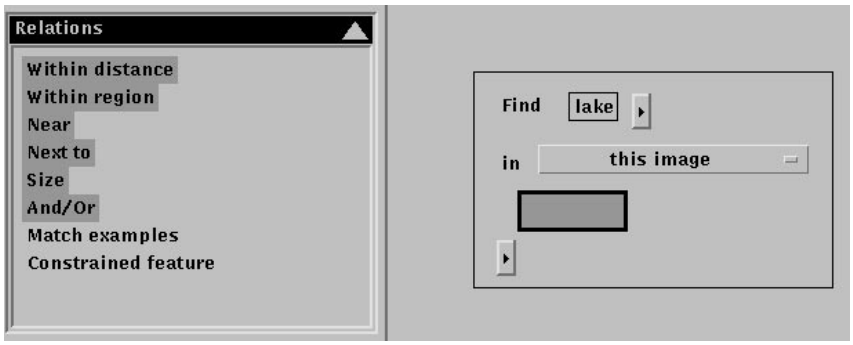

Fig. 10. The "Find" query phrase with a "lake" item dropped into the first receptacle

is the DanDMM menu that contains candidate items for filling the remaining empty receptacle. Note that the candidate phrases are highlighted. By clicking on the empty receptacle with the right mouse, we have requested that DanDMM display all menu items which are syntactically appropriate choices for this receptacle; only such items can be dropped into the receptacle.

Figure 11 shows the final "Find" phrase after dropping the "Within distance" phrase into the empty receptacle, and then dropping "Fire" (from the menu shown in Fig. 8) into a receptacle in the "Within distance" phrase, and typing the number 2 into the type-in area (Fig. 2 contains an example of an unfilled "Within distance" phrase).

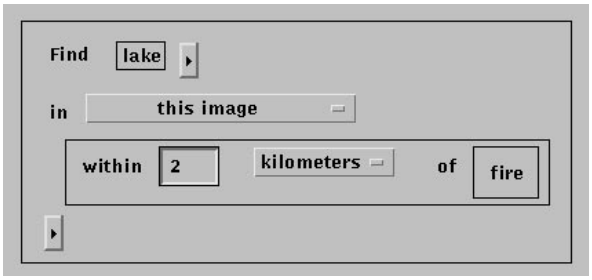

Fig. 11. The completed "Find" query phrase the first receptacle

This completed query phrase can now be submitted to the application server by clicking "Submit". Note that different phrases perform different actions when "Submit" is activated. The interface designer determines actions to be performed by each phrase when "Submit" is pressed. These actions are part of the interface file definition, described in Sect. 4.1.

A powerful feature of DanDMM interfaces is the ability to provide hidden syntax, through the use of "Add" buttons. The "Find" phrase has an Add button (triangle inside a small button) at the bottom in each of the previous figures. Figure 12 shows the result of clicking on this button. A set of additional clauses is revealed, followed by a "Delete" button that can be used to hide them.

Figure 13 shows choice of an alternative selection for filling the last receptacle in the "Find" phrase. In this

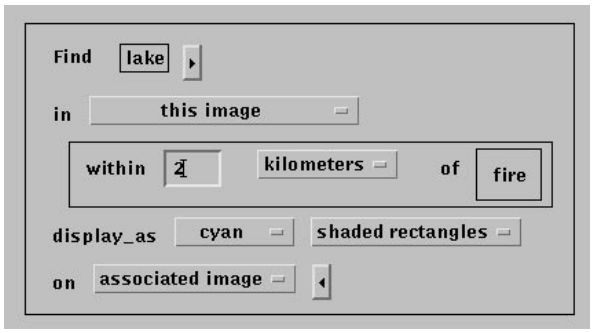

Fig. 12. Additional syntax for "Find" exposed by activating an "Add" button

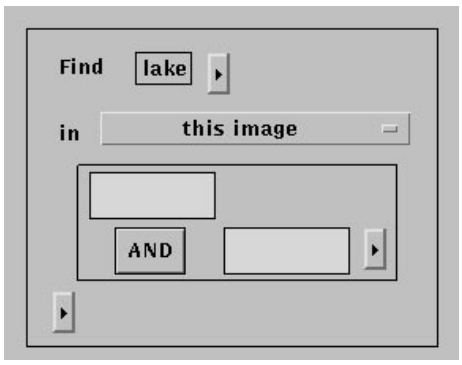

Fig. 13. Find phrase with an incomplete Boolean phrase for specifying query constraints

case, the "And/Or" phrase (shown in the menu in Fig. 10) was dropped into the empty receptacle. This phrase provides for Boolean combinations of constraints. Figure 14 shows two phrases dropped into the Boolean expression. One is the "within distance" phrase from the previous example. We have also created two additional receptacles for Boolean terms, by clicking the add button originally to the right of the second Boolean receptacle. Note that each new receptacle gets its own delete button, which can be used to remove it if desired. Also notice that the bottom "And" button has been toggled to "Or" by clicking on it.

Again, it is important to note that the examples given here show one instance of a DanDMM-style in-

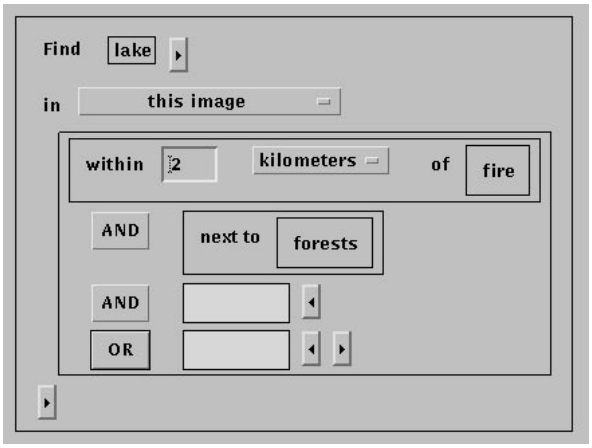

Fig. 14. Find phrase with two constraint phrases dropped into the Boolean phrase. Two additional receptacles have been added by clicking add buttons. Note that the final Boolean connector has been changed to an "Or" by clicking on the toggle button

terface. Interfaces for different applications can vary widely, and each application can have multiple alternate DanDMM interfaces. This interface, for example, might have been designed with an "Add" button to the right of the bottom receptacle in the Find phrase for adding addition Boolean terms directly, rather than through use of a separate And/Or subphrase. All DanDMM-style interfaces, however, will be characterized by composable, nested components for specifying queries and subqueries, the ability to reuse these components via drag and drop operations, dynamic updates of the interface to add user-defined or application-supplied entities, free-form mixing of multi-media objects with more tradition UI widgets, and the ability to hide or reveal optional syntax.

4 The DanDMM-builder framework

DanDMM interfaces are specified via the DanDMM Specification Language (DSL). DSL conceptually consists of three portions: a structural specification, a representation specification, and a dynamic behavior specification.

The structural specification encapsulates the allowable configurations for the interface. In particular, it specifies the ordering of elements within phrases, and the legal composition of phrases and sub-phrases. The structure of the query language represented by a given DanDMM interface is specified using a modified Backus-Naur Form (BNF) grammar specification [17]. BNF provides a compact representation of context-free grammars. BNF is used in DSL to specify the syntax of the application query language; this is in contrast to previous systems which have employed BNF to describe the layout or behavior of the interface [18].

Each terminal or non-terminal on the right-hand side of a BNF expression may be accompanied by one or more modifier clauses. These clauses define the element's representation (widget type and initialization information) as well as its dynamic behavior.

Dynamic behavior is specified using a library of behavior operators. These operators include functions to get and set values in a per-phrase or a global symbol table, to add items to DanDMM menus, and to invoke Java methods in application objects. This last facility allows a DanDMM interface to specify interactions with application objects for each individual phrase. Methods may be invoked to submit queries, to retrieve information for use in dynamically configuring widgets, or to communicate interface state to an application. A set of Java API's is available to allow applications to package multimedia objects in DanDMM format, and to dynamically build new query clauses for on-the-fly installation in a running interface.

The DSL for a given interface is stored in a DanDMM definition file. These definition files allow a good deal of flexibility in creating DanDMM interfaces. Definition files may be written in advance and served to the client on request. Alternatively, the definition files may be created on the fly by a server application. This allows the server to dynamically construct a DanDMM interface that contains, for example, lists of library objects and legal relationships between them.

4.1 Interface definition

In this section we present a partial definition file, to give the flavor of specifying a DanDMM interface.

The sample file given here specifies a query phrase for a simplified video digital library interface.

```
Syntax
{
    <find_clause> = <find_clause_body>
        {
            Submit: JAVA ("application_obj",
                          "submit_query",
                          String ( $))
        };
    <find_clause_body> = "find"
            <object_name>
                { WIDGET (TYPEIN) }
            "in repository"
            <repository>
                { WIDGET (RECEPTACLE) };
    <repository> =
            <obj_archive>
            { WIDGET (TEXT, "obj archive") }
        | <image_archive>
            { WIDGET (TEXT, "obj archive") }
        | <live_video>
            { WIDGET (TEXT, "live video feed") }
}
```

```
Layout
{
    List: "Queries"
    {
        <find_clause> : "Find"
    }
    List: "Repositories"
    {
        <obj_archive> : "Objects"
        <image_archive> : "Images"
        <live_video> : "Live Video"
    }
}
```

Fig. 16. Menus specified by the sample configuration file

The "Syntax" portion of the file specifies a "Find" query phrase. The phrase is shown in the top portion of Fig. 15. The bottom portion shows the phrase with one of the possible repositories dropped into the receptacle. Note that the BNF specifies all allowable values for this receptacle (specified by non-terminal <repository>). Figure 16 shows the two menus specified by this configuration file.

Notice that the configuration file specifies a Java method that is to be invoked on an application object when the "Submit" button is pressed for this phrase. An object called "application_obj" in a global object dictionary is referenced. A "submit_query" method is invoked on this object with a single parameter containing a string that marshalls information from all subphrases of the query phrase.

Although this example is only partial, it gives the flavor for configuring a DanDMM interface. Not shown here is syntax for specifying dynamic reconfiguration of interface components, details of specifying output syntax for passing queries to application objects, some of the other widget types including choice boxes and toggle boxes, and specification of optional syntax (add buttons).

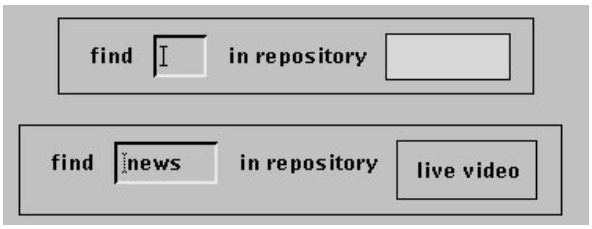

Fig. 15. *Top:* phrase specified by <find_clause> in the sample configuration file *Bottom:* phrase with <live_video> phrase dropped into the receptacle

5 System architecture

The DanDMM-builder framework has been implemented in Java using Java's AWT windowing toolkit to manage the interface. It is available as a set of classes to be invoked from a library application. The application places objects to be accessed by the DanDMM system (either interface objects, or the application instance itself) into the application dictionary, allowing DanDMM to control a user session and to invoke application callbacks.

Figure 17 diagrams the architecture of the DanDMM system. DanDMM interfaces are specified in definition files using the DanDMM Specification Language. These files are interpreted by an Interface Interpreter and used to create interfaces. Where application objects are needed for interface initialization (via Java clauses), the Interface Interpreter issues a request to the Interface Runtime Executive which, in turn, retrieves application objects from the Application Object Dictionary, invokes the appropriate methods on them, and passes return results to the Interface Interpreter. Once the DanDMM Interface has been generated, it is managed by an Interface Run-

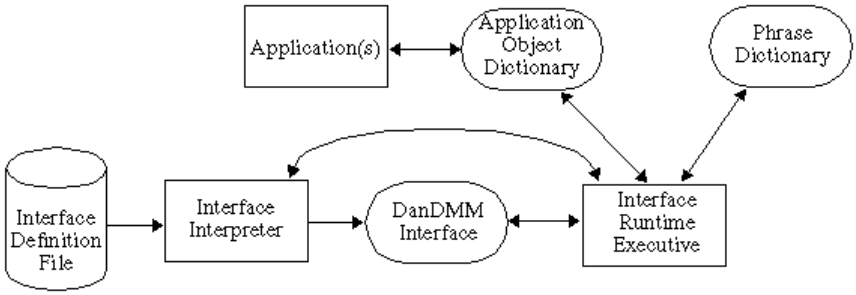

Fig. 17. The DanDMM system architecture

time Executive. The Executive stores prototypes of each phrase in the interface and uses them to evaluate drag-and-drop operations for syntactic correctness. The Executive also processes all actions required when dynamic behavior is invoked, either from the "Submit" button, or through activating interface widgets within phrases. The Executive accesses the Phrase Dictionary when setting or retrieving values, or modifying DanDMM menus, and accesses the Application Object Dictionary to invoke Java methods.

6 Implementation examples

Using the DanDMM framework, we have implemented a query interface for a remote-sensing digital library. Figure 18 shows the interface for this query system. A variety of objects to be searched are listed including land cover types, such as water and forest, and other objects with spatial extent such as wildfires and wildfire burn scars (the latter two hidden in the off-screen portion of the "Objects" menu). Queries are constructed from sentence structures such as "Find", and "Define Simple Object". Qualifying sub-phrases such as "Within distance" and "Next to" are used as components of the query sentences in order to restrict the search. The connective "And/Or" is used to build up sets of more complex qualifiers. Query sentences are submitted to an external search engine as an ascii string consisting of nested function calls.

This sample interface includes the capability of using image examples in constructing queries. A separate navigation application manages image display and simple query tasks including retrieving images by geographical area or name, and multi-resolution zoom. More involved content-based queries are performed via the DanDMM interface. A phrase within the DanDMM interface can issue a request to the navigation application (shown in Fig. 4) for the current image, outlined portion of an image, or set of sub-images and install it in a menu. These images can then be dragged into phrases designed to accept them. Figure 18 shows such a phrase (the "Define simple object" phrase). A set of images has been added to the interface (in the "Images" menu) and then dragged into a phrase that defines a new example-based object type. This phrase when "submitted" adds an entry to a menu containing a set of object type names (seen in the "Objects" menu). This new object type can then be added to a query clause. This simple method for importing images from an application and using them to define new query objects by example provides a powerful mechanism for iterative query refinement.

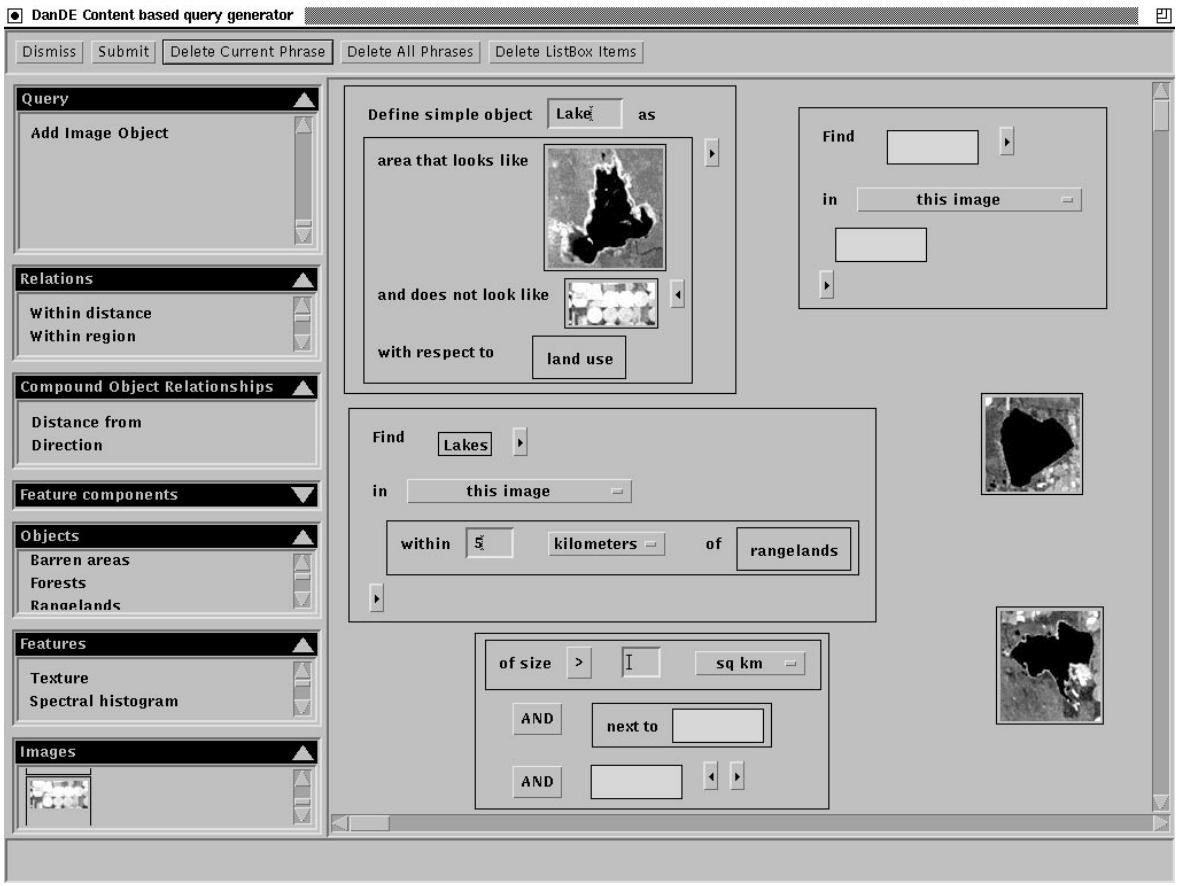

Fig. 18. A Satellite Library Application constructed using DanDMM-builder

Figure 18 also shows the power of DanDMM interfaces for query component reuse. Note the partially constructed Boolean expression on the palette (middle bottom). Such subexpressions can be constructed independently, and then dragged into receptacles as needed. Subphrases can also be dragged out of receptacles and dropped onto the palette. This enables a very free-form interface style accommodating a wide range of usage patterns.

A second sample application is a video digital library interface, seen in Fig. 1. This interface is designed to support multi-search capability including parametric search (e.g., video name, air date), feature search (e.g., location, motion), and free text search on both close-captions and speech transcriptions. Boolean operations are supported among all of these types. Results from a search are returned as either names or thumbnail images; we have plans to implement support for video clips in DanDMM including extraction of sub-images from individual frames for formulating example-based queries. We have formulated two alternative interfaces for this application using DanDMM. The first is a static interface in which all search attributes are listed in menu boxes. The second interface uses dynamic behavior to reconfigure attribute values based on the attribute name selected.

Figure 19 displays two versions of the same interface component from the second (dynamic) interface. The two components differ in choice of the attribute name in the choice list on the left. For each selection, the interface queries the digital library application for a set of attribute values, which are used to populate the choice box on the right.

Fig. 19. User selection from the left choice widget causes the right widget to change contents

Both the static and the dynamic interface were readily specified using the DanDMM specification language. This indicates the power of the interface builder for prototyping multimedia interfaces.

7 Future work

An important part of user-interface development is evaluation of the effectiveness of the interface for performing user tasks. We are working with a group developing user-interfaces for video libraries on a study to compare a DanDMM-style interface with alternate interface styles. This study will focus on ease of use and expressibility of such DanDMM interfaces.

We are pursuing or considering a number of functional and application-related extensions to our current work on DanDMM. One direction is the support for additional types of multimedia objects and for corresponding operations. We are currently implementing support for video objects. At the time of this writing, video is represented as still-image icons, which may be drag-and-dropped just like any other object in DanDMM. We intend extending this to a facility for playing video using VCR-style controls, including the ability to select a single frame for inclusion in a query phrase.

We are also planning capabilities for manipulating multimedia objects within DanDMM. In particular, the ability to interactively draw on images in order to specify particular features or subregions for a query would be extremely useful. We have yet to address how such capabilities would be incorporated into the specification language.

A macro facility, allowing a user to create parameterized subqueries would be a natural and useful extension of the DanDMM interface. We also envision using the DanDMM interface to construct specification files; we anticipate that the current implementation will readily allow us to create an interface which would facilitate the task of writing DanDMM specification files.

In this same vein, the current internal structure used to implement the DanDMM interface is a tightly coupled conglomeration of phrase syntax specification, phrase instances, and phrase appearances. The separation of these elements would lend flexibility to the overall architecture. Furthermore, exposing a programmatic interface to this more modular architecture would allow easy extension of DanDMM either statically at interface design time, or dynamically from within a DanDMM application. The structure of a DSL file is basically static (although it could be dynamically generated outside of DanDMM, by a Common Gateway Interface script for instance). Although contents of individual widgets can be changed dynamically, the structure of a phrase cannot. This limits the degree to which an DanDMM interface can reflect the current state of a database. The capability of specifying within DSL the relationships between database elements and an DanDMM interface would eliminate this limitation.

8 Conclusions

In this paper we have presented a system for synthesis of highly interactive multimedia query interfaces. The interface designer provides a syntactic description of the query language and embeds information about appearance and dynamic behavior. From this description, a structured drag-and-drop interface is automatically generated.

The resulting DanDMM interface provides a limited yet powerful style for query construction. Its strengths include: English-language style queries, automatic restric-

tion to syntactically correct queries, inclusion of multimedia objects in the queries, and dynamic reconfiguration of query constructs based on user actions and/or communication with application objects.

A particularly powerful feature of DanDMM is the ability to incorporate both example-based and more traditional query-styles into an integrated interface for a digital library application. The user can create libraries of semantic definitions using a variety of definition mechanisms – example-based, keyword-based, parametric, and others – and combine these with predefined entities via a single consistent interface style.

The DanDMM system has been implemented in Java, and provides an easy-to-use mechanism for communicating with application objects. Support has been provided for retrieving and displaying multimedia objects as well as incorporating them into queries. This provides a flexible and powerful facility for query of multimedia digital libraries.

References

1. Mintzer, F.C., Boyle, L.E., Cazes, A.N., Christian, B.S., Cox, S.C., Giordano, F.P., Gladney, H.M., Lee, J.C., Kelmanson, M.L., Lirani, A.C., Magerlein, K.A., Pavani, A.M.B., Schiattarella, F.: Toward on-line worldwide access to vatican library materials. IBM J. Res. Devel. 40(2):139–162, 1996
2. Ogle, V., Wilensky, R.: Testbed development for the Berkeley Digital Library Project. D-Lib Magazine, `http://www.dlib.org/dlib/july96/berkeley/07ogle.html`, July/August 1996
3. Smith, T.R.: A Digital Library for Geographically Referenced Materials. IEEE Computer 29(5):54–60, 1996
4. Castelli, V., Bergman, L.D., Kontoyiannis, I., Li, C-S, Robinson, J.T., Turek, J.J.: Progressive search and retrieval in large image archives. IBM J. Res. Devel. 42:253–268, 1998
5. Niblack, W., Barber, R., Equitz, W., Flickner, M., Glasman, E., Petkovic, D., Yanker, P., Faloutsos, C., Taubin, G.: The QBIC project: Querying images by content using color texture, and shape. In Proc. SPIE – Int. Soc. Opt. Eng. vol. 1908, Storage Retrieval for Image and Video Databases, 1993, pp. 173–187
6. Bach, J.R., Fuller, C., Gupta, A., Hampapur, A., Horowitz, B., Humphrey, R., Jain, R.: The Virage image search engine: An open framework for image image management. In: Proc. SPIE – Int. Soc. Opt. Eng., vol. 2670, Storage and Retrieval for Still Image and Video Databases, 1996, pp. 76–87
7. Smith, J.R., Chang, S.F.: VisualSeek: A fully automated content-based image query system. In: Proc. of the IEEE Int. Conf. on Image Proc., (Lausanne, Switzerland), 1996
8. Keim, A.D., Lum, V.: Visual query specification in a multimedia database system. In: Visualization '92, Boston, MA: Oct. 1992, pp. 194–201
9. Wong, K.P., Lum, V.Y.: Approximate retrieval of multimedia objects with natural language queries. In: Proc. of The First International Conference on Visual Information Systems, (Melbourne, Australia), Feb. 1996, pp. 152–164
10. Kaneen, E., Wyard, P.: A spoken language interface to interactive multimedia services. In: IEE Colloquim on Advances in Interactive Voice Technologies for Telecummunication Services, London, UK, June 1997
11. Date, C.J., White, C.J.: A Guide to SQL/DS. Reading, MA: Addison-Wesley, 1989
12. Ozsoyoglu, G., Wang, H.: Example-based graphical database query languages. Computer 26(5):25–38, 1993
13. Davis, B., Marks, L., Collins, D., Mack, R., Malkin, P., Nguyen, T.: The human interface to large multimedia databases. In: Proc. SPIE – Int. Soc. Opt. Eng., High Speed Networking and Multimedia Computing, 1994, pp. 2–12
14. Cousins, S.B., Paepcke, A., Winograd, T., Bier, E.A., Pier, K.: The digital library integrated task environment (DLITE). SIDL-WP-1996-0049, `http:www-diglib.stanford.edu/cgi-bin/WP/get/SIDL-WP-1996-0049`, 1996
15. Cruz, I.F., Lucas, W.T.: A visual approach to multimedia querying and presentation. In: ACM Multimedia, 1997
16. Li, W.S., Candan, S.K., Hirata, K., Hara, Y.: IFQ: A visual query interface and query generator for object-based media retrieval. In: IEEE Multimedia Systems '97, Ottowa, Ontario, Canada, June 1997, pp. 353–361
17. Aho, A., Ullman, J., Sethi, R.: Compilers, Principles, Techniques, and Tools. Addison-Wesley, Reading, MA: 1986
18. Olsen, D.,Jr., Dempsey, E.: Syngraph: A graphical user interface generator. In: Computer Graphics, Proceeding SIGGRAPH '83, Detroit, MI: ACM Siggraph, 1983, pp. 43–50