First publ. in: Visualization '92 : Proceedings of the 3rd annual IEEE conference on visualization : October 19-23, 1992 / IEEE Technical Committee on Computer Graphics. Tokyo : IEEE Computer Society Press, 1992. pp. 194-201

Visual Query Specification in a Multimedia Database System

Daniel A. Keim*
Institut für Informatik, Universität München
Leopoldstr. 11B, 8000 München 40
daniel@dbs.informatik.uni-muenchen.de

Vincent Lum*
Systems Engineering, Chinese University
Shatin, Hongkong
vlum@cusel1.se.cuhk.hk

Abstract

In this paper we describe a visual interface for a Multimedia Database Management System (MDBMS). In spite of the technological advances in display devices, DBMS query languages are still linear in syntax as it was two decades ago. Although natural language interfaces have been found to be useful, natural language is ambiguous and difficult to process. For queries on standard (relational) data these difficulties may be easily avoided with the use of a visual, graphical interface to guide the user in specifying the query. For image and other media data which are ambiguous in nature, we use natural language processing combined with direct graphical access to the domain knowledge to interpret and evaluate the natural language query. The system fully supports graphical and image input/output in different formats. The combination of visual effect and natural language specification, the support of media data, and the allowance of incremental query specification are very effective to simplify the process of query specification not only for image or multimedia databases but also for all databases.

Keywords: *Visual Query Specification, Graphical User Interface, Multimedia Database System, Natural-Language Interface, Information Retrieval, Image Data Management*

1. Introduction

The Visual Query Specification Interface is designed as an integral part of the Multimedia Database Management System (MDBMS) to support a natural query specification process for conventional and image or other media data. Over the last two decades several research projects have been conducted in the area of user interfaces for (multimedia) database systems. The first approach for a graphical database interface is the well-known QBE interface [ZLOO77]. Most recent research in the area of user interfaces focuses on the entity-relationship [WONG82, FOGG84, ROGE88] or the more complex semantic and object-oriented data model [KING84, GOLD85, BRYC86, AGRA90], allowing queries to be directly specified within the schema. In contrast, we use an extension of the relational model to handle and manipulate media data.

Another approach to achieve an easy and user friendly query specification was chosen by researchers in the artificial intelligence area. Much effort went in building natural language processing systems capable of understanding queries expressed in natural language. An overview over the research in this area can be found in [ALLE87] and a good example for the current state of the art is the TEAM system [GROS87]. Because of the problems related to natural language understanding hardly any of the systems is actually used for retrieval in databases. In our system, we are not trying to completely understand natural language. We argue that in order to express a query on formatted data in most cases natural language understanding is not necessary. The user can easily choose the necessary tables and attributes from lists, type the desired values and combine simple conditions into more complex ones. To our knowledge, none of the natural language interfaces to database systems can handle complex combinations of conditions because of the semantic problems related to multiple sentences.

The manipulation of image data is not straightforward as in conventional databases. One main problem is the retrieval of image data based on contents. The difficulty arises because we must match the contents of the media data in the database with the content specification in the query. This difficulty is intrinsically tied to the very rich semantics of multimedia data. To answer queries posed on images a person must draw from a very rich experience encountered in life to derive a good answer. One must have a sophisticated technique at analyzing the contents of the images to get the different semantics associated with the images. Automatic image analysis is not only difficult but almost impossible because little variation in details or application context may change the interpretation of an image. Technology today is not advanced enough to expect systems which have this kind of capability to answer multimedia queries.

However, humans can abstract the contents of multimedia data into text and use the text description equivalent to the original multimedia data to match the user request or query. Figure 1 shows the format of multimedia data which consists of the registration, raw and description data. Raw data

* This research was mainly done while the authors were at the Computer Science Department, Naval Postgraduate School, Monterey, California, USA and was supported in part by NOSC, Direct Funding and the German Scholarship Foundation.

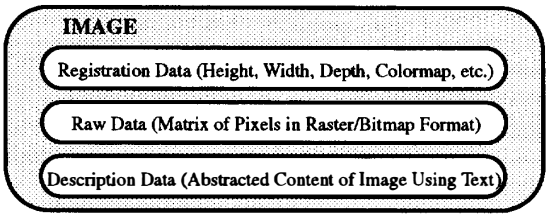

Figure 1: Image Data Format

is the bit string representation of the image, video, sound, etc. obtained from scanning or digitizing the original multimedia data. Registration data enhances the information about raw data and is not redundant. Description data is for specifying the contents of multimedia data. In our system, we assume that users will supply the description data in natural language form.

To accomplish the goal of content retrieval of multimedia data, only a restricted interpretation of natural language is necessary which is done by the parser component using the application dependent dictionary as a semantic basis. The current implementation of the parser uses augmented-transition network parsing and interpretation routines. The details of the parser are beyond the scope of this paper but are given in [ROWE91].

A problem of the natural language approach is that it is generally the case that the description of a multimedia data does not exactly match the description of a user query. The reason is that it is difficult for different users, or even the same user at different times, to describe the same thing identically because the user may use synonyms or generalize/specialize categories belonging to the domain of interest and so on. Hence, in an earlier paper we proposed an intelligent approach to approximate matching by integrating object-oriented and natural language understanding techniques. The details of the natural language processing done by the parser and matcher component of our system are beyond the scope of this paper. They are given in [GUGL92, LUM92]. A prototype of MDBMS has been implemented at the Naval Postgraduate School [LUM89, ROWE91].

A visual user interface like ours is generally applicable in all situations, and the evaluation of queries is less complicated and less time-consuming than pure natural language processing. In the remaining part of this paper, Section 2 outlines the important features of the Visual Query Interface and gives examples for query specifications. Section 3 describes the other database operations (schema definition, insert, update and delete) and Section 4 gives concluding remarks.

2. Visual query specification

The goal of a visual user interface is to support the query specification process allowing the user to use the database system efficiently. It should allow inexperienced users to retrieve data from the database without having to know a specific query language. In today's database management systems the user is forced to think in terms of data model and query language, differing a lot from his way of thinking. Often a user can express a query easily in natural language, but has difficulties to express it in a given query language.

Most queries involve both media and formatted data. For conditions on the media part of a query we use our intelligent matching algorithm which is directly processing natural language. To help the user in describing the desired media data we provide direct graphical access to the domain knowledge used to interpret and evaluate the natural language expression. For conditions on formatted data, natural language is too imprecise and may lead to undesired results. We argue that for formatted data the use of natural language is not necessary. The system already knows the table and attribute names and - if this information is presented to the user in an adequate way - the user can easily specify his query without using natural language. We think, that a visual, graphical user interface is best suited to support the user in the process of specifying a query.

The data model adopted in our system is an extended relational model. Despite some drawbacks the relational model has great advantages: It is well known, widely used and has a firm theoretical basis. For our purpose, we extend the relational model to capture media datatypes and, as shown below, we also extend the query language to allow the manipulation of media data and to facilitate the query specification process.

Before describing the Visual Query Interface in more detail, we first outline ways to achieve a natural query specification process.

2.1 Towards a natural query specification

Usually every user can describe a query (or at least the desired result) easily in natural language. Unfortunately, natural language expressions representing a query are imprecise and difficult to automatically translate into a formal query language to be understood by a database management system. We argue that the gap between the user's way of expressing a query in natural language and database manipulation languages like SQL can be improved considerably.

When comparing the user's natural language (NL) expression for a query with corresponding SQL statements the first difficulty is that the table and attribute names do not exactly match. In a visual user interface this problem is easy to overcome. Lists with all table names can be presented to the user who simply selects the desired ones using a pointing device. In the next step lists with the attribute names of the selected tables will be presented to the user. Again, he only chooses the desired ones using a pointing device. In contrast to natural language processing no difficult and time-consuming matching between the users' NL expressions and the table or attribute names is necessary.

Another difficulty is related to joins between tables. In examining a large number of queries expressed in natural language as well as SQL we found that in most cases the join condition directly corresponds to some specific NL expressions. However, the tables connected by a join condition had to be deducted from the context because in most cases they were implicit. Additionally, the number of joins used in most of the queries was small compared to the number of possible joins. This can be explained by two facts. First, the number of semantically meaningful joins is restricted and second, some of the most frequently used joins are already intended at the design time of the database. In order to provide a natural way of expressing joins, in our system we allow the database designer and user to define and name joins prior to their actual use. A predefined join can involve more than two tables (e.g. two tables joined by means of a third table) thereby providing a simple way of expressing m:n relationships. Once defined and named, all predefined joins can be used to specify a query. Predefined joins differ from views: First, the result of a predefined join is not a table as in the case of a view, but a specific connection between tables. Second, predefined joins allow connections between different levels in nested queries. Examples are given in the next section.

Another thing we learned in examining the process of human query specification is the handling of complex queries composed of an arbitrary combination of conditions. Given a complex data retrieval task the user partitions it into smaller subtasks which are easier to handle. Starting with the easy parts of the query the user deals with all parts and combines the results into the final solution. In our system we support this way of handling complex queries by an incremental query specification which is described in the next section.

2.2 Description of the query specification

In this section, we will describe the Visual Query Interface in more detail by presenting several examples. We will show the main features of the Visual Query Interface, especially those which are different from other graphical database interfaces. We start with a general description of the steps used in the retrieval process.

When starting the MDBMS system the user will be automatically connected to the Visual Query Interface and the first step is to select the desired database. Then the user gets the system menu providing the main database manipulation functions: insert, delete, update and retrieval. When selecting retrieval, the user gets the query specification window. The next step is to select the tables to be used in the query. For each selected table a list with all attributes will be displayed in a separate window and all predefined connections involving at least one of the selected tables will appear in the *Connections* window. To specify the result list (projection) the user has to move the desired attributes to the *Result List*. Now, only the conditions need to be specified. Using connections, attributes from the selected tables and operators provided by the *Tool Box*, the query can be built easily using the mouse. In the *Query Representation* window the query is displayed graphically. Each part of the query is represented by a small box, simple conditions by a single box, subqueries by a double box, and the connection lines are labeled with the kind of connection used. An advantage is that every part of the query can be addressed for edit or delete at any time during the query specification process.

Predefined joins

A special feature of the Visual Query Interface are predefined joins. Predefined joins can be defined at design time of the database by the database designer. Having the necessary connections between tables in mind, the database designer tunes the database so that joins can be executed efficiently. All semantically meaningful joins can already be defined at design time. However, if other joins are needed later, the user can define them at any time using the *NEW* option.

For this paper, let us consider a sample database with the following tables:

mission (m_id, m_name, direction, goal, task)
navy_base (base_id, location, size)
officer (o_id, o_name, address, salary, commander_id, ship_nr, o_image)
ship (s_nr, s_name, class, yr_built, cap_id, mission_id, base_id, s_image)
ship_weapon (s_nr, w_nr, quantity)
weapon (w_nr, w_name, category, type, range, w_image)

Only few of the possible joins between these tables are semantically meaningful; e.g. the only meaningful equijoin between ship and officer is *ship.cap_id = officer.o_id*. Most other equijoins like *ship.s_name = officer.address* or *ship.s_nr = officer.o_id* are senseless and will never be used.

Predefined joins allow an easier specification of complex queries. The user does not need to think about the attributes and conditions necessary to join tables, he simply chooses the desired one out of the predefined joins in the Connections window. Predefined joins can involve more than two tables, e.g. the following SQL statement expresses a three way join between ship and weapon:

Example:

```
select s_name, w_name from ship, ship_weapon, weapon
where ship.s_nr = ship_weapon.s_nr
      and ship_weapon.w_nr = weapon.w_nr
```

To express the join conditions of the same query using the Visual Query Interface, only one step is necessary. After selecting tables and attributes the predefined join ***ship carries weapon*** has to be selected. The result is displayed in the Query Representation window as shown in figure 2.

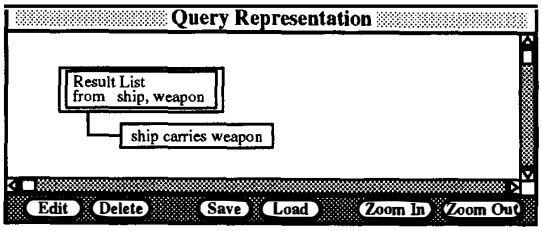

Figure 2: Example for a Predefined Join

Predefined joins can even be used to express joins of relations with themselves, e.g. the query *'Select the name of each officer together with the name of his immediate commander!'* can be easily specified using a predefined join. The user could specify the query as follows: First he has to select the officer table. Since he deals with two instances of this table he has to select it twice resulting in an officer1 and officer2 window. The last step is to select the predefined join *officer* ***is_commander_of*** *officer*.

Two more things about predefined joins need to be mentioned: First, any kind of join (not only equijoins) may be predefined and second, it is allowed to predefine identical joins with different names. This is useful to allow an easy identification of the required predefined join because the same query can be expressed differently. A simple example is *ship* ***carries*** *weapon* and *weapon* ***is_carried_by*** *ship*.

The internal handling of predefined joins and related query optimization issues are beyond the scope of this paper. They are described in [KEIM91].

Incremental query specification

To support incremental query specification we allow the user to start with any part of the query; e.g. to specify the query *"Select the name of ships which carry all weapons of the category 'surface-to-air'! "* the user can start with the subquery *weapons of category 'surface-to-air'* and then continue with the main part of the query without specifying the connection between these two parts. At a later stage, the user may combine the separate parts.

As an additional feature, we provide an option to save and reload any part of a query for later use. If the user needs part of the query later for other queries he may save the desired part by selecting the corresponding items in the Query Representation window and assigning a name to them. Later he can reload the desired part when working on a different query and integrate it in the new query. Furthermore, to enhance the clarity of display, parts of a query can be grouped together and displayed as one box (zoom in). If the user wants to see the query in full detail at a later stage he may use the zoom out option.

Tool Box

The Tool Box allows fast access to all functions supported by the system. The functions are divided into five groups: logical operators and basic elements *(AND, OR, Condition, Subquery)*, comparison operators *(=, >, <, ≥ , ≤)*, nesting operators *(Exists, not Exists, IN, not IN, ALL)*, set operators *(∪, ∩, −, ⊆, ⊇)* and aggregate operators *(AVG, SUM, MAX, MIN, COUNT)*. The semantics of most operators are the same as in SQL. An additional operator, the *'all'*-operator, has been added because it allows an easier and clearer specification of a special category of queries. A detailed description of the *'all'*-operator with a formal definition of its semantics is beyond the scope of this paper and is given in [KEIM91]. *Condition* and *Subquery* options are necessary for the incremental query specification process. Using these options, the user is able to continue the query specification with a different part of the query. When selecting *Condition* the user gets a new condition box and in case of selecting the *Subquery* option he gets a double box for a new subquery.

Media Description Editor

Another important part of our system is the way of specifying the natural language description part of a query necessary when media data are involved. If the user selects a media attribute in the specification of the condition, automatically a special *Media Description Editor* will be displayed in a separate window where the media description can be specified. An example for the Media Description Editor is shown in figure 3. The description editor has special features to support the intelligent matching process mentioned above. The *'Hierarchy'* button supports the user in finding the right description. For a selected (highlighted) word or phrase it presents the corresponding part of the object-oriented domain knowledge providing hints for a better description (see figure 4 for an example). With the *'Weight'* button the user is able to assign weighting factors to subject, verb and object component groups of a natural language expression. The weighting factors are used in the intelligent matching algorithm to combine the weights of the different groups. If the user does not provide weighting factors, an equal factor is assigned to all component groups. When selecting the *'Check'* or *'Done'* button the entered description is instantly sent to the parser. The parser tries to check and interpret it and, in case of an error, gives back the error message. In case of the *'Done'* button the Media Description

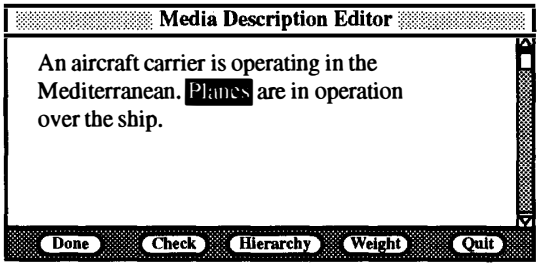

Figure 3: Media Description Editor

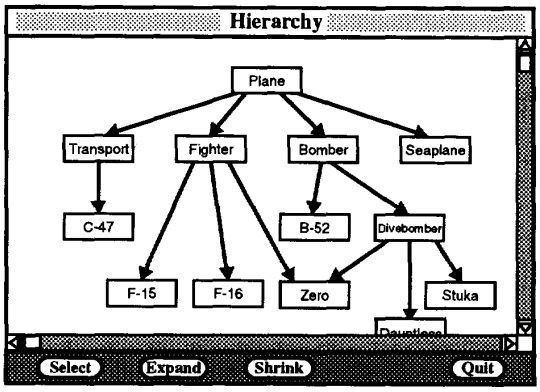

Figure 4: Example for a Noun Hierarchy

Editor disappears if the parser does not return an error. If the user wants to edit the description at a later stage he has to select the corresponding box in the Query Representation window and push the *'Edit'* button.

A larger example

To further explain the query specification process let us walk through a more complex example:

> *'Select the name, base_id and image of ships which can carry all weapons of the category surface-to-air and where the image shows "An aircraft carrier is operating in the Mediterranean. Planes are in operation over the ship.".'*

If a user wants to specify the query he might want to start with an easy part, e.g. *'weapons of the category surface-to-air'*. To specify this part the user first selects *Subquery* in the *Tool Box* providing him a second double box for the subquery. Then he selects *weapon* in the Tables window. As a result he gets all attributes of the weapon's table in a separate window and by clicking to *w_nr* he selects the desired attribute. The next step is to specify the condition. By clicking to *Cond* in the Tool Box he gets an empty condition box in the Query Representation window and by clicking to the attribute *category* in the weapon's window, '=' in the Tool Box and typing in *surface-to-air* he fills the box with the actual condition.

As the next part the user might want to specify the image description condition *'image shows "An aircraft carrier is operating in the Mediterranean. Planes are in operation over the ship."* '. The specification process for this part is similar to the specification of the first part. The user selects the *ship* table and after getting a new condition box he selects the attribute *s_image* from the ship window. Because *s_image* is a media attribute, the system automatically provides the special *Media Description Editor* window. In this window the user can type the natural language description for the image, in our example *"An aircraft carrier is operating in the Mediterranean. Planes are in operation over the ship."*. When selecting the *'Done'* button the description will directly be interpreted by the parser.

The last step is to specify the main part of the query and to compose the parts into the final result. Starting with the beginning of the query (*'Select name, base_id and image'*) the user moves the attributes *s_name, base_id* and *s_image* to the Result List window. By selecting *Cond* from the Tool Box and *ship carries weapon* from the connections window the user specifies the join condition. Now, as the last part of the query, the user has to specify the 'all'-condition. This can be accomplished by getting a new condition box, clicking to *w_nr* in the weapons window, *'='* and *'all'* in the Tool Box and the double box representing the subquery *'weapons of the category surface-to-air'* in the Query Representation window. The last step is to combine the conditions into the final result. This is done by selecting the conditions and the logical operator *AND* from the Tool Box. In figure 5 the final result of the query specification process is shown. Two other possibilities to express the same query without using the special *'all'*-operator are shown in figures 6 and 7. The figures give an impression how complex queries are represented in our Visual Query Interface. The reader might easily imagine the specification process of these queries.

2.3 Presentation of results

An important aspect of a visual user interface for multimedia database systems is the presentation of the results. The question is how to present a huge amount of multimedia objects. The problem is that, unlike conventional attributes, multimedia objects may have a time and space dimension.

To solve these problems we choose a combined form and list oriented approach. Generally, the results are presented as a list. In place of the media values only buttons are displayed which the user may select in order to see and/or hear the corresponding media object. Another way to see an object in more detail is to point to the line containing the desired tuple to get the tuple displayed in its form representation. Forms allow users to see more attributes (including media attributes) than available in a list; however, in contrast to lists, only one tuple at a time is displayed. By using the list representation the user can easily scan a huge amount of data but at any step he has also the possibility to get the more detailed form version of a media object. When specifying a query automatically a new form is created including spaces for the values of all attributes involved. With the help of a graphical design tool the user can rearrange the form according to his needs and store it under a different name. In future queries the user can choose an already defined form when dealing with a similar query. In figure 8 the results of our sample query are shown using the customized form *withDescription*.

The combined list and form oriented approach only solves the space problem. It is highly desirable to have an

influence on the time dimension of multimedia objects, too. Nobody wants to see a whole video in order to identify it as the desired one. Each time, a media object with time dimension (e.g. video or sound) is played, the user should have the possibility to stop, skip a part, go back, etc. In a special window all possible options should be presented as buttons so that the user can choose one of them using the mouse. A precondition for this kind of handling time dependant media objects is random access to their storage representation. In our MDBMS prototype system random access to media objects with time dimension is not supported yet and therefore we do not provide these features for time dependant objects. Other desirable options for time dependant media objects are the possibility to see a text version of a sound object (e.g. possible for speech or songs), the possibility to define index points which are directly accessible without linearly scanning the media object and the possibility to define synchronization points (for composed media objects).

3. Other database operations: schema definition, insert, update and delete

In this section we will give an overview of the other database operations that are supported by our visual user interface. The operations to be described are Schema Definition, Insert, Update and Delete.

For the schema definition we have chosen a rather simple table-oriented approach. The system designer defines a new relation by identifying name, type, width and key of all the attributes. The possible datatypes including media datatypes are presented in a menu. More important at this stage, however, is the possibility to predefine joins allowing an easier query specification by the end user.

Insert, Delete and *Update* are performed using a **form-based** approach. When creating a new table, automatically a new form is created. The spaces for the attribute values reflect the possible length of values to be inserted or updated. As mentioned before, the user is able to rearrange the form according to his needs.

The **insert** operation is performed by filling a form with data. After specifying the attribute values for a tuple the user selects the *'Insert'* button to trigger the actual insert. During the insertion process also an *'Erase All'* button to erase all fields is available. After inserting a tuple the user remains in the form to insert other tuples. To quit the insertion mode the user has to use the *'Quit'* button.

The first step of the **update** operation is similar to the retrieval operation because it is necessary to identify the tuples desired for update by specifying a selection condition. The condition, a simple or complex one, is specified using the query specification window as described in section 2.2. However, only attributes from one table may be in the *Result List*. The result for the specified query is presented as a list and by clicking to one row of the list a tuple is caused to be displayed in a form. To change the tuple the user simply edits the values in the form and uses the *'Update'* button. Other buttons available in the form are the *'Next Tuple'*, *'Previous Tuple'*, *'Update All'* and, of course, the *'Quit'* button. By using the *'Next Tuple'* or *'Previous Tuple'* button the user gets the next or previous tuple found by the user given selection condition, letting the displayed tuple unchanged. When using the *'Update All'* button an empty form is provided which the user fills to change all tuples found using the user given selection condition. Figure 9 shows an example for the update process.

Like update, **delete** is a two step operation. First, tuples must be retrieved according to a specified selection condition. In contrast to update, no *Result List* is necessary because tuples cannot be deleted partially. The second step, the actual deletion, is performed using the resulting list or a form. Both, list and form provide buttons for deleting the tuples one-by-one or all selected tuples at once.

Another important issue is how the **media datatypes** are integrated into forms because e.g. a sound cannot be displayed in a form and other difficulties arise for images. For the **sound** type two fields are necessary, one for the path of the sound file and one for the description. Furthermore a *'Play Sound'* button is available for each attribute of type sound to play the sound after inserting the path. For attributes of type **image** a frame, two text fields for the path and the description and a *'Display Image'* button to display the image after inserting or updating the path are provided (see figure 9). The frame for the image can be of an arbitrary size making it necessary to zoom the image in or out when it is displayed. By distinguishing between raw and registration data we efficiently support many different input/output formats including pixrect format with RMT_none (black and white) or RMT_EQUAL-RGB (color), Brown Univ.'s ALV format or Utah runlength format (URLE).

4. Concluding remarks

A major problem faced in today's database systems is the lack of a natural way to specify complex queries. It is caused by the gap between the user's way of thinking and the query languages used in most systems. Basically these systems are still linear in syntax. Such languages have neither made use of the visual aspect of human senses nor the natural process of the human minds to process information. Although a lot of work has been done in the area of user interfaces for database systems, no query language comes close to the natural query specification process used by humans.

Our contribution exploited in this paper is a visual, graphical database interface supporting a natural query specification process. We combine an extended relational DBMS with an easy-to-use visual interface allowing direct access to standard as well as multimedia data. It narrows the gap between the user's way of thinking and formal query lan-

guages by using visual user interaction. In our system, we support an incremental query specification and predefined joins to make the query specification process user friendly. The user is guided as much as possible allowing a quick, convenient and useful query specification. Further research is necessary to come even closer to the user's way of query specification e.g. by allowing the user to directly communicate with the system in natural language when appropriate.

We believe that our system provides a simple and elegant approach to the retrieval of multimedia data. The simplicity of our user interface lies in the natural way of query specification being directly obtained from queries expressed in natural language. We also believe that our approach is a general one that can be readily applied to most database query interfaces (e.g. relational systems and extensions hereof).

Future work include the integration of other media datatypes like video and mixed media data with the need to support the presentation of time dependent data.

References

[AGRA90] Agrawal R. et al. "OdeView: The Graphical Interface to Ode", Proc. ACM-SIGMOD 1990 Int. Conf. on Management of Data, Atlantic City, 1990.

[ALLE87] Allen J. " Natural Language Understanding", Benjamin/Cummings Publishing Company, Inc., 1987.

[BRYC86] Bryce D. and Hull R. "SNAP: A Graphics Based Schema Manager", Proc. IEEE 2nd Int. Conf. on Data Engineering, Los Angeles, 1986.

[FOGG84] Fogg D. "Leesons from a 'Living in a Database' Graphical User Interface", Proc. ACM-SIGMOD Int. Conf. on Management of Data, 1984.

[GOLD85] Goldman K. J. et al. "ISIS: Interface for a Semantic Information System", Proc. ACM-SIGMOD Int. Conf. on Management of Data, Austin, TX, 1985.

[GROS87] Grosz B.J. et. al. "TEAM: An experiment in the Design of Transportable Natural Language Interfaces", Artificial Intelligence, No. 32, 1987.

[GUGL92] Guglielmo E. "Natural Language Processing of Captions for Retrieving Multimedia Data", Proc. of Conf. on Applied Natural Language Processing, Trento, Italy, 1992.

[KEIM91] Keim D.A. and Lum V. "A Graphical Database Interface for a Multimedia DBMS", Tech. Report NPSCS-91-013, Naval Postgraduate School, Monterey, CA, 1991.

[KING84] King R. and Melville S. "Ski: A Semantics-Knowledgeable Interface", Proc. Int. Conf. on Very Large Data Bases, Singapore, 1984.

[LUM89] Lum V. and Meyer-Wegener K. "A Multimedia Database Management System Supporting Content Search in Media Data", Tech. Report NPSCS-89-020, CS Depart., Naval Postgraduate School, Monterey, CA, 1989.

[LUM92] Lum V., Keim D. A. and Kim K.-C.: "Intelligent Natural Language Processing for Media Data Query", Proc. Int. Golden West Conf. on Intelligent Systems, Reno, NEV., 1992.

[ROGE88] Rogers T. R. and Cattell R. G. G. " Entity-Relationship Database User Interfaces" in Readings in Database Systems, edited by M. Stonebraker, 1988.

[ROWE91] Rowe N. and Guglielmo E. "Exploiting Captions for Access to Multimedia Databases", Tech. Report NPSCS-91-012, Naval Postgraduate School, Monterey, CA, 1991.

[WONG82] Wong H. K. T. and Kuo I. " GUIDE: Graphic User Interface for Database Exploration", Proc. of the Int. Conf. on Very Large Data Bases, Mexico City, 1982.

[ZLOO77] Zloof M. M. "Query-By-Example: A Data Base Language", IBM Systems Journal, No. 4, 1977.

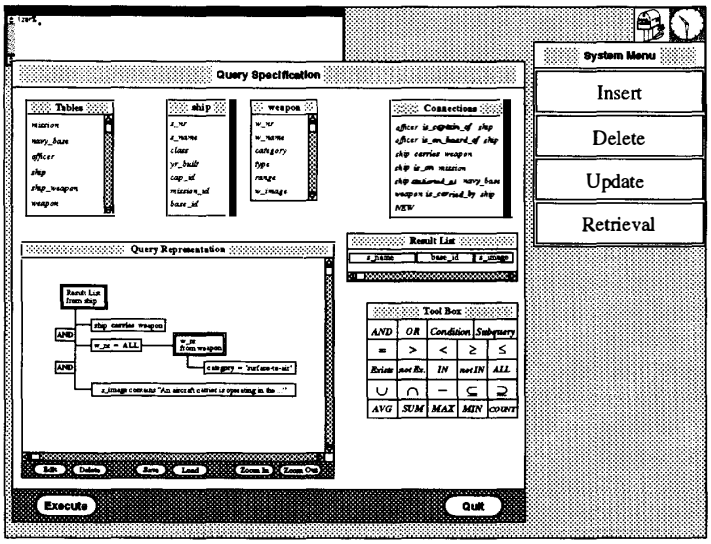

Figure 5: Query Specification (Example 1)

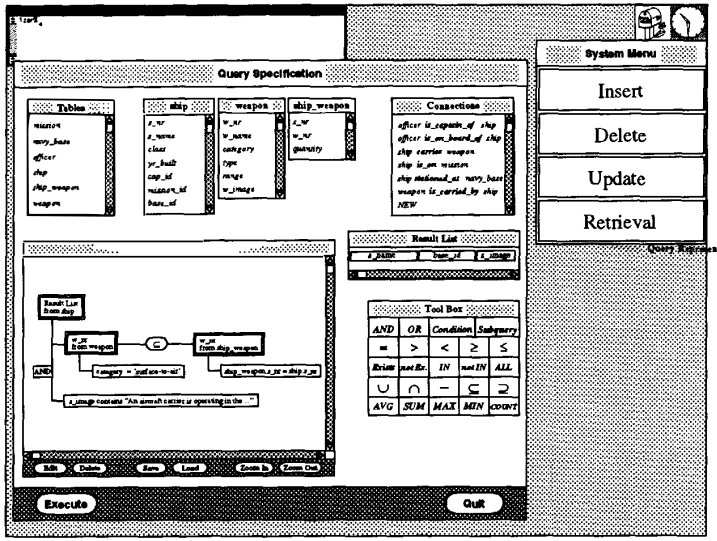

Figure 6: Query Specification (Example 2)

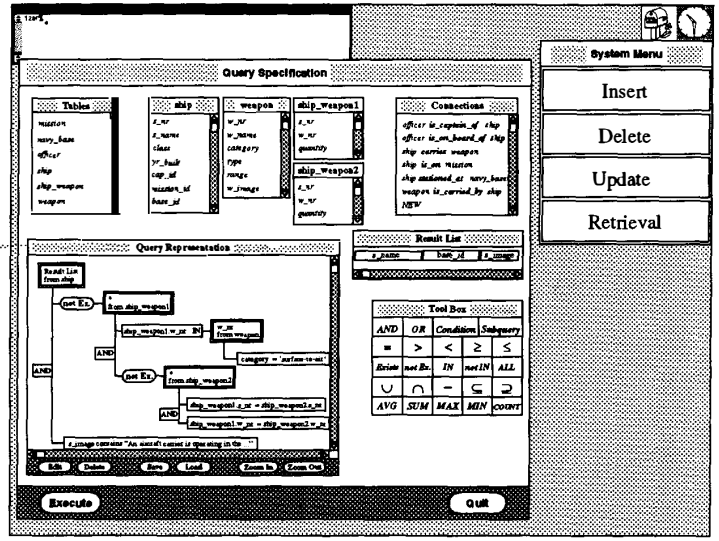

Figure 7: Query Specification (Example 3)

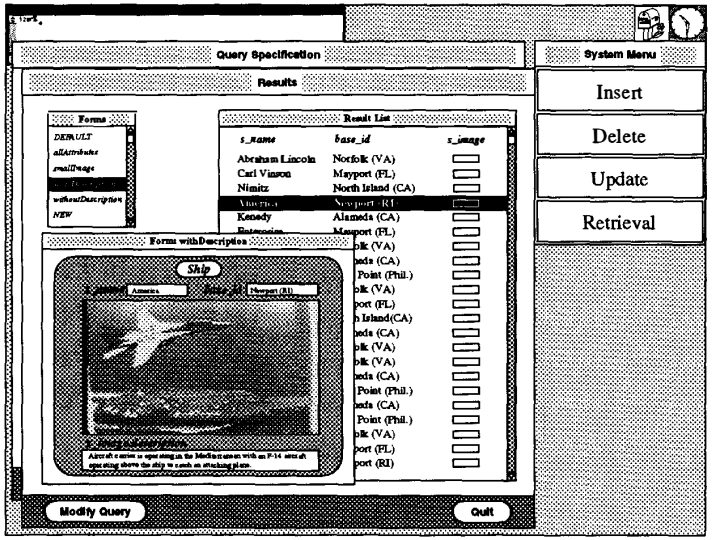

Figure 8: Presentation of Results

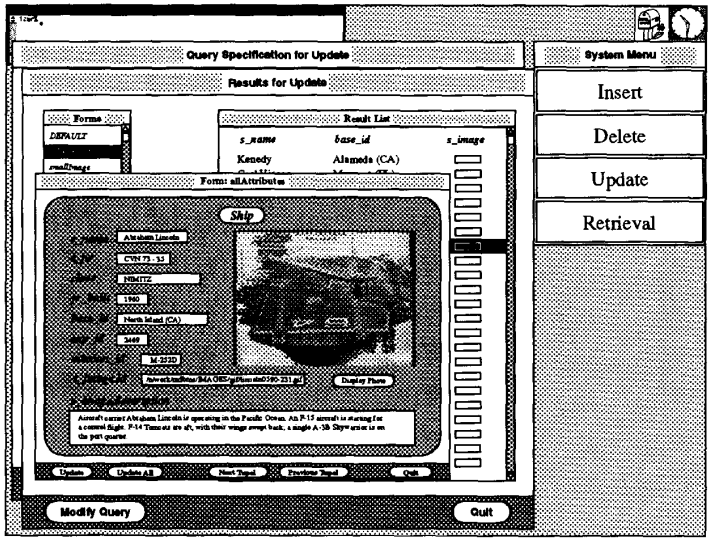

Figure 9: Update Process