Progressive Querying and Result Visualization in Logical and VR Spaces

Shi-Kuo Chang*, Maria F. Costabile+ and Stefano Levialdi++

*Dept. of Computer Science, University of Pittsburgh, Pittsburgh, PA. 15260, U.S.A.

+Dip. di Informatica, Universita' di Bari, Via Orabona 4, 70126 Bari - Italy

++Dip. di Scienze dell'Informazione, Universita' di Roma "La Sapienza", Via Salaria 113, 00198 Roma - Italy

ABSTRACT

A software tool called VQRH (Visual Querying and Result Hypercube) is presented. It enables the database users to visualize both database queries and retrieval results in many different representations. Due to the lack of knowledge about the database, a user may not be able to formulate a query whose result fully satisfies his or her needs in a single attempt. Using the VQRH progressive querying tool, the user interacts with the database by means of a sequence of partial queries, each displayed together with the corresponding result as one slice of the VQR Hypercube. This tool allows the user to change both query and result representations. It also presents the query history in a 3D perspective, so that a particular partial query on a slice may be brought to the front of the Hypercube for further refinement. Preliminary experimental results on using the VQRH tool for scientific databases are described. The combination of logical paradigms and virtual reality paradigms in progressive querying is discussed.

1. INTRODUCTION

Many studies have been performed in the area of human-computer interaction (HCI) to allow a fruitful and friendly use of complex programs. The design of a good interface is the crucial point for a broader diffusion of the computer technology. In order to improve the communication between users and computers, several approaches have recently been proposed. Among them, the visual approach, i.e. the use of drawings and images to convey information in a concise and intuitive way, is advantageous due to the reduction of the mental load. Based on this approach, programming environments, languages and systems have been developed in recent years.

Visual communication is quite appropriate for interacting with databases, particularly in the data retrieval phase. Although databases are designed, created and modified by professional people, there are several kinds of persons whose job requires access to databases specifically for extracting information. With this aim, special purpose languages called *query languages* have been developed. However, the most widely used languages for database querying are actually programming languages [10], therefore requiring the user to have: a) knowledge on the language syntax, b) technical background, c) information about the system, both on the application domain and on the interaction mechanisms. Recently, the steady increase of database users, including more and more non-expert and casual users, motivated the development of easier-to-use query languages. In particular, visual query languages (VQLs) have been proposed, which are part of more friendly visual interfaces to database systems [2]. These languages use visual representations to present the domain of interest and the retrieval requests.

The non-expert user and the casual user, who need to extract information from a database, are generally not able to directly formulate a query whose result fully satisfies the user's needs, at least in his or her first attempt. This is due to the following reasons: a) the user does not know exactly what data are contained in the database and how such data are structured; b) he or she is not familiar with the query language and prefers to ask simple queries. *Therefore, the user may prefer to formulate a complex query progressively, i.e. step by step, by first asking general questions, obtaining preliminary results, and then revisiting such outcomes to further refine the query in order to extract the result he or she is interested in.* A non-monotone query progression should be allowed. During this process of progressive querying, an appropriate visualization of the obtained preliminary result could provide a significant feedback to the user. The above described advantages of performing a progressive query through visual interaction, also displaying, in a suitable modality, the partial results in each step, has lead us to propose the Visual Querying and Result Hypercube (VQRH), which is a new tool for visualizing progressive querying and results when interacting with a database.

In this paper we will describe the VQRH tool and its use as query interface for a medical database. A prototype of VQRH has been developed and some experiments are reported. The paper is organized in the following way. The next section briefly reports on our previous research on multiparadigm interaction with databases. This research is important for the current work, because the progressive querying described in this paper exploits the multiparadigm approach [7]. Section 3 discusses the advantages of multiple-representation visualization. The structure of the VQR Hypercube is described in Section 4. The operational aspects of the VQR Hypercube will be explained in Section 5 by showing a working example of the formulation of a progressive query. In Section 6 a formal model of the progressive VQRH system is presented. In Section 7 some preliminary experimental results on using the implemented prototype are described, and the combination of logical paradigms and virtual reality (VR) paradigms in progressive querying are discussed. Section 8 concludes the paper by presenting further research issues.

2. VISUAL QUERYING OF DATABASES

In recent years, many *Visual Query Systems* (VQSs), i.e. query systems implementing VQLs, have been proposed in the literature [2]. They adopt a range of different visual representations and interaction strategies, depending on the visualization formalism primarily used. Some VQSs employ hybrid representations based on a combination of the visual formalisms. However, the existing VQSs generally restrict the human-computer communication to only one kind of interaction paradigm. It is our hypothesis that the presence of several paradigms, each one with different characteristics and advantages, will help both naive and experienced users to interact with the system. For instance, icons may well evoke the objects present in the database, while relationships among them may be better expressed through the edges of a graph, and collections of instances may be easily arranged into a form, etc. The way in which the query is expressed also depends on the chosen visual representation. In the existing VQSs, queries on diagrammatic representations are mainly expressed by following links, forms are often filled with prototypical values, and iconic queries are usually constructed by spatially composing primitive icons. In [7] a system is proposed, whose interface can offer to the user different interaction mechanisms for expressing a query, depending on both the experience of the user and the kind of query itself. The selection of the appropriate interaction paradigm can be made with reference to a user model that describes the user's interests and skills [8]. Such a model should be dynamically maintained according to the history of the interactions, including both queries and user's reactions to system messages. The multiparadigm interface introduced in [7] for formulating database queries is the starting point for the VQR Hypercube described in this paper.

3. VISUAL PRESENTATION OF INFORMATION

To present information expressively and effectively is a difficult task. Different graphical techniques are appropriate for various situations. An interface designer should consider all situations that may arise, design the appropriate presentations, and then decide which is the most effective for each situation. In order to relieve this responsibility from the interface designer, Mackinlay has proposed a system called APT (A Presentation Tool), that automatically designs graphical presentations of information [18]. When some information is extracted from a database, APT synthesizes a graphical design and then renders an image that presents such information. APT focuses on the design of 2D static presentations of relational information, i.e. it considers bar charts, scatter plots and connected graphs, while ignoring icons and other figurative encoding techniques. It also does not consider 3D presentations, nor temporal encoding techniques, i.e. animation.

Besides Mackinlay's work, little research on the automatic presentation of information has been published, especially on design variation issues. A system similar to APT is BARHAT [16], which is based on a simple design algorithm for choosing a pie chart, a bar chart or a line chart to represent a single unary function. However, its range of representations was limited. A preliminary theory to provide a scientific foundation to the development of graphical methods for data presentation is presented in [9]. Other works primarily focus on the automatic determination of the presentation content, by adding or removing pieces of information to generate an effective presentation or sequences of related information [14, 15]. Casner worked on automated presentation from task description. His approach is the most sophisticated one developed so far [5, 6]. For graphic design issues, the reader is referred to [3, 4]. Our only concern in the current design of VQR Hypercube is to exploit the advantages of generating different information representations, so that the user can choose which one to visualize. A desirable improvement of the VQR Hypercube would be the automatic identification of the most appropriate presentation for the specific task and user.

4. THE VQR HYPERCUBE

In this section we describe the structure of the VQR Hypercube as depicted in Figure 1. The VQR Hypercube is a tool for visually representing a progressive query and the results obtained step by step. Both queries and results can be displayed by any of the available visual representations. As shown in Figure 1, the main structure of the VQR Hypercube is a cube in which we distinguish the front face and a set of slices parallel to the front face. The front face is divided in two panes: the left one (query area) is a working area in which the user formulates the query in any of the available interaction paradigms; the right one is used to display the query result (when it exists) in any of the available presentation modalities. The slices parallel to the front face represent the query *history* (a sequence of query-result system states), because each slice contains in the query area a query formulated at a certain time, and in the result area the result of that query. Subsequences of these slices represent the steps in the formulation of progressive queries. Since both query and result representations can be multidimensional, we call this visualization tool the VQR Hypercube.

To select the interaction paradigm for formulating the query, the menu at the bottom of the query area is used. Similarly, the menu at the bottom of the result area is used for selecting the result representation. The current query paradigm and result paradigm are shown in the bar area below each pane.

In the VQR Hypercube, the subsequence of slices Q1, Q2,...,Qn represents a progressive query. Each slice Qi refers to the step in which the partial query Qi is being formulated. The query can be formulated in any of the interaction paradigms allowed by the system. As mentioned in the previous section, the system provides form-based, diagrammatic, iconic and the hybrid paradigm, as indicated by the four possibilities in the menu below the query area. The result can also be shown with icons or other graphics representations, including forms, pie-charts and plots, as indicated by the four possibilities in the menu below the result area. For example, in Figure 1 the query on the front slice is: "Display a list of patients". The query is formulated in the query area with the iconic paradigm, but any other paradigm could be used. The result of the query shows the existence of 253 instances of patients. This result can also be shown in some other representation.

Ideally, the system chooses the representation paradigm for both query and result on the basis of the knowledge it has about the user, the task and the application domain, as described in [7]. Of course, the user always has the freedom to change the paradigm by clicking on the menus below the Hypercube.

The Hypercube slices are used to keep track of the query history, i.e. the queries formulated during the interaction session. By clicking on the slice containing the Qi query, the user can bring it to the front slice of the Hypercube and start from there a new progressive query. For example, from the slices for the progressive query Q1,Q2,...,Qn, the user brings up query Q2 and starts from there a new progressive query B1,B2,...,Bm. Then, by clicking on query Q3 in the interaction history, the user may visualize it again on the front slice. The commands to bring up and *show* the history, or *hide* the current history, are in the pull-down menu for *history* in the menu bar on top of the screen.

In Figure 2, in the upper left corner of the screen there are three icons. The top icon, the *VQRH book*, represents the VQR hypercube. When the user clicks on the VQRH book icon, a new slice of VQRH is created and the user can then enter a new query. The second icon, the *opened box*, is used to display and select the permissable classes of database objects. As shown in Figure 2, by clicking on *LoadLib* in the SIL menu bar, a window with icons representing the existing databases appears. Once a database is selected, by clicking on the *opened box* icon, a menu with icons representing the classes of objects of the database appears. Such icons can be dragged into the query area to build an iconic query. The bottom icon, the *tool* icon, represents the premissable operations. By clicking on the tool icon, a menu (the tool box) with the permissable operation icons appear, as shown in Figure 3. The permissable database operations include *which_one*, *how_many*, *zoom_in*, *zoom_out* and *relation*. With *which_one* and *relation*, any join-query for a relational database can be formulated. The *zoom_in* and *zoom_out* can be used to navigate in a class hierarchy. Finally, the *how_many* operation gives the cardinality (size) of a class of objects. These database operations can be used to construct a query. The details will be described in the next section.

5. PERFORMING A PROGRESSIVE QUERY

When a person is looking for a specific information in a new town, he or she may look at a map, perhaps refer to a park (large green area) or to a railroad station (long, parallel lines indicating the rails) and then, gradually converge to the wanted address (road, house number, etc.). That person may then backtrack and look for a bus station or an underground stop to see whether a public transport may help in reaching that specific place. Conversely a parking lot may be searched to decide whether to go by car or public transport. Finally some other important landmarks may be searched: whether it is a shopping area to buy a present, etc. The typical querying process we have called *progressive querying* is a similar process, i.e. one in which there might be an initial goal (specific information to be obtained). Such goal may then be changed in the process of querying either by backtracking and focusing on some other information, or simply by looking at a different but related information as the new goal. This approach differs from the one in database theory, where the query is gradually refined and, at each step of the progression, the possible candidates for the result are both reduced in number and more specialized. An example from the relational database area can be found in [11]. In our case, a non-monotone query progression is allowed. In order to illustrate this concept of progressive querying, it may be useful to show examples of how the user interacts with the system when querying the medical database.

In Figures 1-5 we illustrate a progressive query. For the sake of simplicity, only the front slice of the VQR Hypercube is shown in Figures 2-5. Let us suppose that a clinical researcher needs information about the lab tests a patient has already undertaken. Since the clinical researcher is not yet familiar with the system, she will not know much about the data stored in the database and how to interact with the system. Therefore, it is more useful for her to ask some simpler queries on both patients and lab tests to become familiar with the system and obtain some idea on how to obtain a result for her original query. The suggested interaction paradigm is the iconic paradigm, as indicated by the system at the bottom of the query area (Figure 1). By looking at the icons representing the classes of objects stored in the database (Figure 2), she selects the icon representing patient and drags it in the query area, in order to know how many instances of patient are in the database. This simple query is performed by clicking on the *how_many* operation icon which is the second one in the Tool menu on the left side of the screen (Figure 3). The icon for this operation is that one with the big X on a deck of icons. This icon is then highlighted while, in the meantime, to give a feedback of the operation being performed, the patient icon is transformed in the result area into a deck. Next, after a search in the database, the iconic representation of the result appears in the result area, namely a deck of the patient icon with the number 253 indicating the number of instances of patient in the database (Figure 1).

At this point, the user drags the *lab test* icon in the query area and clicks again on the same *how_many* operation icon to find out how many instances of both patient and lab tests are in the database (Figure 3). As displayed in the result area, there are 3057 instances of lab tests and still 253 instances of patient, as in the result of the previous query. Since the user knows that in the database there are instances of both patient and lab test, the user could proceed by asking her specific query, namely "what are the lab tests that the patient Wang already undertook". In the diagrammatic mode this query is performed by connecting both icons for patient and lab test with an icon representing a specific relation between these two entities, i.e. the patient has undertaken a lab test.

If the user is not very familiar with the system, which is our hypothesis, it would not be easy for her to identify and select such a relation from the menu. Therefore the user constructs the query progressively, with the help of the system. In fact, having both the patient icon and the lab test icon in the query area, the user can ask the system if there is any relationship between these two entities. This query is performed by clicking on the operation icon for *relation*, which is at the bottom of the *tool* menu (Figure 3). As result of this query, a deck of the relationships connecting patient and lab test appears in a pop-up window in the result area (Figure 4). This means that there are more relationships between the two entities. The user inspects all of them and then selects the appropriate one for performing the original query. In our example, there are three relationships between patient and lab tests, "Test to be done", "All test" and "Test taken". The relation "Test taken" is the one appropriate for the user's original query and she selects it. Finally, the user, i.e. the clinical researcher in our example, can complete her original query, which is performed as shown in Figure 5. By clicking on the *which_one* icon (the big X at the top of the *tool* menu), the user can enter the patient's name. In the result area the form-based representation of the lab tests undertaken by patient Wang appears.

It is worth noting that for showing the results of the above query the user has selected the form-based representation because this is appropriate for such kind of query. In fact, in a medical laboratory the usual way for presenting most lab test results is to list the resulting values. If other representations are meaningful for the results, the system will provide them and the user can choose to visualize these alternative representations in pop-up windows above the result pane.

In the *tool* menu there are other operation icons for general-purpose queries. For instance, the third icon from the top represents a *zoom_out* operation, which is used to show the parent class of a class of objects. When applied to the class of objects, the parent class is shown in the result area. Conversely, the fourth icon represents a *zoom_in* operation to select a subclass of objects.

Last but not least, the tool menu has two useful indicators: the *green light* to the right of the upper tool icons, and the *red light* to the right of the lower tool icons (Figure 1). Since the figures are black-and-white, when the light is *on* its *color* will have a darker shade. A subquery Q is *admissable*, if it can be transformed into another representation. This is indicated by the *green light* of the tool menu. If the query is inadmissable, the *red light* is on. Whenever the *green light* is on, the user can manually select a different query representation, or the system can make a suggestion based upon the system's understanding of the user's querying characteristics. The experimental system currently supports the first option. To further extend the system's ability in suggesting an interaction paradigm, we need both a user model [8] and a formal model of the VQRH system, which will be explained in the next section.

6. A FORMAL MODEL OF THE VQRH SYSTEM

There are two major requirements for a formal model of a progressive query system. The first requirement is that the model must explicitly express the formulation of admissable subqueries. The second requirement is that the model must be capable of expressing the selection of different interaction paradigms.

The two major requirements for the modeling of a progressive query system stem from the fact that the user may create many subqueries in the course of formulating a query. However, each subquery must be admissable [7] in the sense that the subquery can be given a unique meaningful interpretation. If the subquery is inadmissable, it cannot be interpreted and therefore the system cannot translate it into another representation, or provide partial query results. On the other hand, if the subquery is admissable, the system can translate the subqueries into different representations, select a different interaction paradigm, and generate consistent query results.

The progressive visual query system can be modeled by a generalized Petri net called G-Net [12] in a way that is intuitively appealing. The G-Net differs from ordinary Petri nets, in that it can invoke other G-Nets. Therefore, the G-Net is a high-level net with a powerful abstraction mechanism. A G-Net may contain special places called Instantiated Switch Places (ISPs). When a token flows into the ISP of a net G1, the execution is "switched" to another net G2. More precisely, a token is deposited in the corresponding Generic Switch Place (GSP) of the net G2. An initial marking for G2 is defined, and G2 starts its execution. When a goal place of G2 is reached, the execution terminates. Now the token reappears in the original ISP, and the execution of G1 resumes. More details about G-Nets as a means to model complex information systems are presented in [12,13]. The object-oriented flavor of G-Net should also be noted.

Because of the power to invoke subnets, the G-Net is very suitable to model a progressive query system using the ISP/GSP mechanism. As shown in Figure 6, from the initial place INIT, depending upon the user preference, we can enter one of the four ISPs: ISPform, ISPicon, ISPdiagram and ISPhybrid, corresponding to one of the four GSPs: GSPform, GSPicon, GSPdiagram and GSPhybrid. There are four net types: Qform, Qicon, Qdiagram and Qhybrid, corresponding to the four query paradigms. The user preference is represented by the places (user states): UPform, UPicon, UPdiagram and UPhybrid.

Once the net Qx is entered, the user starts to formulate a subquery SQ. The goal place GOALx of the net Qx is reached, when the subquery becomes admissable, i.e., when the subquery can be meaningfully interpreted. When the goal place of Qx is reached, we exit the net Qx and return to the corresponding ISP that invokes the net Qx. Each net Qx can be modeled in much greater detail. However, from the global modeling point of view, this is unnecessary. At this point, an admissable subquery has been constructed in one of the four query paradigms. By definition, this admissable subquery can be translated into any other query paradigm [7]. We then enter the display place, DISPLAY, to display the alternate admissable queries according to the user preference. Notice we will display in paradigm y, in DISPLAYy, even if the query is in paradigm x. Now we go to the output place OUTPUT, to output the query results. After that, the token flows to EXIT place. If the user desires to continue refining the query, we return to the initial place INIT. Otherwise, we exit from the system.

In the preceding transition rule, GOAL is the final goal place. In the above formal model, whenever a transition rule involving UPx (user preference place) as the input place is fired, a token should be returned to that UPx, except for the "quit" preference UPquit.

The question arises as to which states should become historical events. In the above model, whenever we return to the INIT state, or enter the GOAL state, a snapshot of the G-Net is taken, which becomes a slice of history in the VQR Hypercube. Another way is to provide an explicit "snapshot" user preference, so that a slice of history is stored, only if the user has a token in the snapshot place.

The detailed subnets Qform, Qicon, Qdiagram and Qhybrid are not specified here. Such details are not needed for the overall understanding of the progressive VQS system.

The model of the progressive VQR system is shown in Figure 6. Further simplification is possible, if we exploit the ISP/GSP mechanism as follows. The UPquery has token carrying the attribute Aquery with value: form, icon, diagram or hybrid; the UPoutput has token carrying the attribute Aoutput with value: bar, pie, iconic or plot. They can invoke a specific method of a G-Net. The places are combined as follows: (1) ISPform, ISPicon, ISPdiagram and ISPhybrid are combined into ISPquery, with four methods corresponding to the four paradigms; (2) DISPLAYform, DISPLAYicon, DISPLAYdiagram, and DISPLAYhybrid are combined into ISPdisplay, with four methods corresponding to the four paradigms; (3) OUTPUTbar, OUTPUTpie, OUTPUTiconic and OUTPUTplot are combined into ISPoutput, with four methods corresponding to the four output modalities. After this simplification, the resultant model is illustrated in Figure 7. With this simplified model, it is also very natural to model the switching among interaction paradigms. The switching among visual query paradigms is determined by the Aquery attribute of the token in the place UPquery, where the attribute Aquery has value: form, icon, diagram or hybrid.

7. PRELIMINARY EXPERIMENTAL RESULTS AND EXTENSION TO VR PARADIGMS

The VQR Hypercube tool has been implemented on a PC using Visual Basic. It runs under the Windows NT environment and can be easily interfaced with different commercial database systems. In our experiment, we use dBase IV, although other commercial database systems can also be used. The figures, Figure 1-6, are actual screen-dumps produced by this experimental system.

We are currently experimenting with information retrieval using the VQR Hypercube tool in two application domains: (1) the medical databases, and (2) the library databases. The subjects are students with no previous experience in using VQRH. The preliminary experiment indicates that they have little difficulty in learning VQRH, and they can formulate queries after half an hour of interaction. They generally like the idea of progressive querying, and find it useful to be able to recall any past query-and-result slice. However, they sometimes do not understand the meaning of some icons.

As mentioned before, the possible operation icons to be used in performing the queries are included in the *tool* menu, and object icons and relations are provided in other menus. But the user unfamiliar with the system may still have difficulty in identifying the object icons and relations he or she needs. We discovered this problem after a test we conducted with some users. Therefore, we introduced the *relation* icon to show the relationships between a pair of objects classes. It was used in the query in Figure 4. The user who wants to know about the tests performed by a patient, but does not know which relationship to use, can click on the *relation* icon to ask the system to indicate such relations. New operation icons similar to the *relation* icon can be added to the *tool* menu, so that the *intensional* relationships among object classes can be explored by the user [21]. The user should be able to ask questions such as: "What are the object classes that are related by a specific relation?", "What are the relations that relate specific object classes?", "What object classes are related to a specific object class via a specific relation?", etc. Such questions can be represented by new operation icons or iconic sentences [21].

>From the preliminary experimentation, it is already clear that the visualization of the retrieval result is very important for the success of this approach. As a further improvement, we can use 3D features to present the results in a virtual reality (VR) setting. For example, the *physical location* of medical records can be indicated in a (simplified) 3D presentation of the *Virtual Medical Laboratory* by blinking icons. For the *Virtual Library*, the *physical location* of books can be indicated by blinking icons in a 3D presentation of the book stacks of the library. What the user sees on the screen will be the same (after simplification) as what can be experienced in the real world.

Furthermore, VR such as the Virtual Library or the Virtual Medical Laboratory can become a new query paradigm. For example, the user can select a book by picking it from the shelf, just like in the real world. Thus we can add VR both as a query paradigm and as a result paradigm, something similar to the Microsoft Bookshelf.

The admissability conditions to switch between a *logical paradigm* (our previous paradigms are all logical paradigms) and a *VR paradigm* (such as the Virtual Library) can be defined as follows. For a logical paradigm, a *VR-admissable query* is an admissable query whose retrieval target object is also an object in VR. For example, the VR for the Virtual Library contains stacks of books, and a VR-admissable query could be any admissable query about books, because the result of that query can be indicated by blinking book icons in the Virtual Library. Conversely, for a VR paradigm, an *LQ-admissable query* is a VR where there is a single marked VR object that is also a database object, and the marking is achieved by an operation icon such as *similar_to* (find objects similar to this object), *near* (find objects near this object), *above* (find objects above this object), *below* (find objects below this object), and other spatial operators. For example, in the VR for the Virtual Library, a book marked by the operation icon *similar_to* is LQ-admissable and can be translated into the following query: "find all books similar to this book."

An example of a VR-admissable logical query is illustrated in Figure 8. The query is to find books on bicycles. The result is presented as marked objects in a Virtual Library. The user can then navigate in this Virtual Library, and switch to the VR query paradigm. Figure 9 illustrates an LQ-admissable query. The query is to find books similar to a specific book about bicycles that has been marked by the user. The result is again rendered as marked objects in a Virtual Library. If we switch to a form-based representation, the result could also be rendered as items in a form. This example illustrates progressive querying can be accomplished with greater flexibility by combining the logical paradigms and the VR paradigms. The experimental VQRH system has been extended to support VR paradigms, but the similarity function must be supplied for the problem domain.

8. CONCLUSION

In this paper we described the Visual Query and Result Hypercube which is a tool for visualizing a progressive query for a database. The main features of our system include: a) the mediating of the user-to-database interaction by a progressive query that is built on a sequence of queries (the query process may sometimes backtrack to a previous query and be redirected towards a different goal); b) the availability of multiple representations of both the query and the result; and c) the presentation of the query history in a 3D perspective, so that any particular step of the process may be selected and retrieved to aid in focusing on the expected result.

An important research issue is to develop algorithms for the system to automatically suggest the switching between querying paradigms. Previous theoretical results indicate when the query satisfies the admissability condition, switching is possible [7]. The experimental VQRH system already provides the support for paradigm switching. For the advanced VQRH system, whenever the *green light* is on, the system may also suggest the most appropriate paradigm. This switching condition could be specified, using the methodology described in [8], to determine how the usage history can be abstracted into a user model to guide the switching among query paradigms. In actual implementation, it can be done by table look-up.

It is often easier to find something if we know in what spatial relationship the objects are placed. The VR paradigm provides such topological help -- finding a book on a bookshelf of a known library, booking a bed in a ward, reserving a seat in a theatre where one may see the stalls, etc. The combination of logical paradigms and VR paradigms also leads to the possibility of using the VQRH system as a means to *explore* new information, which will be of special importance for scientific databases. It is an interesting research issue to design other operation icons to explore the *extensional* relationships among objects for both the logical paradigms and the VR paradigms. Finally, more experimentation is needed to evaluate users' acceptance of the proposed approach.

ACKNOWLEDGEMENT: The authors wish to thank Mr. Ssu-Ping Ko, who implemented the experimental VQRH system in Visual Basic.

REFERENCES:

[1] Arens, Y., Miller, L., and Sondheimer, N. Presentation Design Using an Integrated Knowledge Base. In Intelligent User Interfaces, Sullivan, J. W. and Tyler, S. W., Eds., ACM Press, New York, 1991, pp. 241-258.

[2] Batini, C., Catarci, T., Costabile, M. F., and Levialdi, S. Visual Query Systems. Technical Report 04.91, Dipartimento di Informatica e Sistemistica, di Roma "La Sapienza", Roma, Italy, 1991 (revised in 1993).

[3] Bertin, J. Graphics and Graphic Information Processing. Walter de Gruyter & Co., Berlin, 1981.

[4] Bertin, J. Semiology of Graphics. The University of Wisconsin Press, 1983.

[5] Casner, S.M. A Task-analytic approach to the Automated Design of graphic presentations. ACM Transactions on Graphics. 10 (2), 1990, pp. 11-151.

[6] Casner, S.M. Task-Analytic Design of Graphic Presentations. PhD Thesis, University of Pittsburgh, Ann Arbor: University Microfilms International, No. 9109427, 1990.

[7] Catarci, T., Chang, S. K, Costabile, M. F., Levialdi, S., and Santucci, G. "A Graph-based Framework for Multiparadigmatic Visual Access to Databases", to appear in IEEE Transactions on Knowledge and Data Engineering.

[8] Chang, S. K, Costabile, M. F., and Levialdi, S. Modeling Users in an Adaptive Visual Interface for Database Systems. Journal of Visual Languages and Computing, Vol 4, N. 2, 1993, pp. 143-159.

[9] Cleveland, W. S., and McGill, R. Graphical Perception: Theory, Experimentation, and Application to the Development of Graphical Methods. Journal of the American Statistical Association, Vol. 79, N.J387, 1984, pp.531-554.

[10] Date, C.J. An Introduction to Database Systems. Vol.I. Addison-Wesley, 1987.

[11] D'Atri, A., Di Felice, P., and Moscarini, M. Dynamic Query Interpretation in Relational Databases. Information Systems, Vol. 14, N. 3, 195-204, 1988.

[12] Deng, Y., and Chang, S. K. A G-Net Model for Knowledge Representation and Reasoning", IEEE Trans. on Knowledge and Data Engineering, September 1990.

[13] Deng, Y., Perkusich, A., Figueiredo, J. and Chang, S. K., "Integrating Software Engineering Methods and Petri Nets for the Specification and Prototyping of Complex Information Systems", Proceedings of 14th International Conf on Application and Theory of Petri Nets, Chicago, June 21-25, (M. Marsan, Ed.), Springer-Verlag, 206-223, 1993.

[14] Feiner, S. APEX: An Experiment in the Automated Creation of Pictorial Explanations. IEEE Computer Graphics and Applications, Vol. 5, N. 11, 1985, pp. 29-37.

[15] Friedell, M. Automatic Graphics Environment Synthesis. PhD Thesis, Case Western Reserve University, 1983.

[16] Gnanamgari, S. Information Presentation through Default Displays. PhD Thesis, University of Pennsylvania, 1981.

[17] Hauptmann, A. G. From Syntax to Meaning in Natural Language Processing. Proc. of the 9th Nat. Conf. on Artificial Intelligence, 1991, pp. 125-130.

[18] Mackinlay, J. D. Automatic Design of Graphic Presentations. PhD thesis, Stanford University, 1986.

[19] Mackinlay, J. D. Search Architectures for the Automatic Design of Graphical Presentations. In Intelligent User Interfaces, Sullivan, J. W. and Tyler, S. W., Eds., ACM Press, New York, 1991, pp. 281-292.

[20] Mandelkern, D. GUIs The Next Generation. Comm. of ACM, Vol. 36, April 1993, pp. 37-39.

[21] A. Massari, S. Pavani and S. K. Chang, "An Iconic Query System with Intensional Feedback Capabilities", Proc. of 1993 Visual Languages Conference, Bergen, Norway, August 1993, 386-388.

[22] Shneiderman, B. The Future of Interactive Systems and The Emergence of Direct Manipulation. Behavior and Information Technology, Vol. 1, 1982, pp. 237-256.
