A “Graphical Notebook” as Interaction Metaphor for Querying Databases

Sonia Fernandes Silva
Departamento de Engenharia Elétrica, Universidade Federal da Paraíba
C.P.10053 Campina Grande PB, Brasil
silva@dis.uniroma1.it / sonia@dee.ufpb.br

Tiziana Catarci
Dipartimento di Informatica e Sistemistica, Università di Roma “La Sapienza”
Via Salaria, 113 I-00198, Rome, Italy, Tel/Fax: +39-6-49918331
catarci@dis.uniroma1.it / tic@cs.brown.edu

Ulrich Schiel
Departamento de Sistemas e Computação, Universidade Federal da Paraíba
C.P.10106 Campina Grande PB, Brasil
ulrich@dsc.ufpb.br

Abstract

Recent database applications are typically oriented towards a large set of non-expert users, and therefore, they need to be equipped with suitable interfaces facilitating the interaction with the system. Within this context, we propose the design and implementation of a new user-oriented visual query system, called TVQE, that supports a diagrammatic representation of the database schema (including historical classes and relationships) and a “graphical notebook” as interaction metaphor in order to provide a more intuitive form of expressing conventional and historical queries. Hence, users are released from syntactical difficulties which are typical of textual query languages. This paper concentrates on the overall view of the system architecture and the detailed description of how queries are visually expressed by using the new interaction metaphor. Implementation issues of TVQE are also discussed.

1 Introduction

The availability of graphical devices at low cost and the advent of direct manipulation paradigm [Shn83] have given rise in the last years to a large diffusion of interfaces using visual techniques.

Concerning the development of information systems, databases are designed, created and possibly modified by database experts, but there are different kind of users whose job
requires access to databases specifically for extracting information. So, visual interfaces for databases, in particular, the so-called Visual Query Systems - VQSs (see [Cea97] for a survey), have arisen as alternatives to traditional query languages, such as SQL. VQSs can be seen as an evolution of traditional query languages provided by DBMSs. They are characterized by several notable features, such as:

- The use of icons and visual metaphors, which attracts the user attention and stimulates her/his curiosity, instead of text.
- The fact that the user does not need to have previously knowledge of the database schema and to know a query language.
- The availability of interactive mechanisms to support the typical process of query formulation. In fact, such a process can be seen as constituted by three phases [CSA93]: the user selects the part of the database s/he wants to operate on (location phase); s/he defines the relations within the selected part in order to produce the query result (manipulation phase); s/he operates on the query result (visualization phase).

Within this context, we have initially developed a VQS described in [FSSC97b] as an effort to put in an easy-to-use visual form the task of formulating queries on databases (including temporal databases). Then, we have extended the original VQS with new features, such as hierarchical visualization of complex schemata; use of a familiar metaphor in order to facilitate the query specification; intensional visualization of the query result; visualization of the query result by using the dynamic query approach [AS94], [Shn94]; and processing of data about the user’s interaction within the query environment (preliminary results can be seen in [FSC98]).

The extended version of the system has been called Temporal Visual Query Environment (TVQE). The main idea of the system is to provide the user with a simple visual query environment. In such an environment the different query specification activities are performed in a homogeneous way, through elementary graphical operations, exploiting a model-independent visual metaphor. In this way, non-expert users do not need to understand neither the underlying data model nor the syntax and semantics of a textual query language. Moreover, the visual interaction mechanisms of TVQE are provided with precise syntax and semantics founded on a formally defined graph-based data model, namely TGM (Temporal Graph Model), and a minimal set of temporal graphical primitives (see [FSSC97b], [FSC99a], [FSC99b]).

In this paper we concentrate on the overall view of the system architecture and the detailed description of how queries are visually expressed by using the interface. Moreover, some implementation issues are discussed.

This paper is organized as follows. Section 2 describes the system architecture. Section 3 summarizes VQS classification criteria. Section 4 sketches the concrete scenario of TVQE. Section 5 is about implementation issues of TVQE. Finally, Section 6 draws the conclusions.
2 The System Architecture

TVQE has been developed in the context of a global system architecture [CSC97] which allows one to visually interact with heterogeneous databases through an adaptive visual interface. Each query can be stated through different interaction modalities (switching among different visual representations can be done during the query formulation) and its formal semantics is given in terms of the TGM query primitives. Translation algorithms are provided for expressing the query in terms of the query language of the various underlying DBMSs. Hence, the end-user is not conscious of the existence of several heterogeneous databases whenever accessing them.

More specifically, the global system architecture is illustrated in Figure 1, with the following basic functionality: based upon the user model provided by the User Model Manager, the Visual Interface Manager selects the visual representation most appropriate for the user. The User Model is responsible for collecting data and maintaining a knowledge base of the user model components, namely the class stereotype, the user signature, and the system model. The Visual Interface Manager and the User Model Manager are described in more detail in [CCS94].

![Figure 1: TVQE architecture within a global system architecture](image)

At the bottom of the figure, different databases structured according to several data models are shown. Each database is translated into a Temporal Graph Model Database, TGMDB, through the TGMDB & Query Manager, using the mappings described in [CSC97]. It is up to the TGMDB & Query Manager to manage such mappings and to translate the visual queries into queries that can be executed by the appropriate DBMS.

The outstanding components constitute the TVQE architecture and provide the following functionality: the user visually specifies a query (conventional or historical) through the TVQE interface. The graphical schema (TGM schema) is the basic instrument on which the user formulates a query. Actually, the query is represented as a subschema of the initial schema. The Translator module converts the subschema of interest into an internal representation that the DBMS can process. The Data Visualization module ana-
lyses the query result (How will the data be visualized? What are their types?) and then presents the user with an interactive visualization (time-oriented or not).

It is worth noting that the Data Visualization module is out of the scope of this paper. The issues related to the user model management and the interaction with heterogeneous databases are also out of the scope of this paper.

3 Classification Criteria for VQSs

The most significant classification criteria, among those introduced in [Cea97], are visual representation and interaction strategy. Concerning the visual representation, the systems are organized into four classes, depending on the adopted visual formalism [Har88], namely form, diagram, icon or a combination of them.

Form-based representations are the first attempt to provide the user with friendly interfaces for data manipulation; they are usually proposed in the framework of the relational model, where forms are actually tables. Their main characteristic consists in visualizing prototypical forms where queries are formulated by filling corresponding fields. Diagrammatic representations are those most used in existing VQSs. They represent the various concepts available in a model by specific geometric elements. They also adopt as typical query operators the selection of elements, the traversal on adjacent elements and the creation of a bridge among disconnected elements. The iconic representation uses set of icons to denote both the objects of the database and the operations to be performed on them. The hybrid representation uses an arbitrary combination of the above approaches, either offering to the user various alternative representations of databases and queries, or combining different visual structures into a single representation.

The Interaction strategies are provided to perform the following activities: understanding the reality of interest, where the goal is the precise definition of the fragment of the schema involved in the query, namely query subschema; and formulating the query, where the query subschema can be manipulated in several ways, according to the available query operators. The goal of query formulation is to formally express the operands involved in the query, with their related operators.

A classification of VQSs based on the strategies for understanding the reality of interest results into the following classes: top-down, where general aspects of the reality are first perceived, and then specific details may be viewed; and browsing, where the user examines a concept and, through its neighborhood, a new element can be selected to be the current one, and its neighborhood is also shown (this process proceeds iteratively).

The strategies for formulating the query are classified as follows: by schema navigation, which has the characteristics of concentrating on a concept (or a group of concepts) and moving forward in order to reach other concepts of interest, on which further conditions may be specified; by sub-queries, where the query is formulated by composing partial results; by matching, which is based on the idea of presenting the structure of a possible answer that is matched against the stored data; and by range selection, allowing a search conditioned by a given range on multi-key data sets to be performed, where the dynamic query approach [AS94], [Shn94] is an implementation of this technique.

Based on the criteria described above, TVQE adopts the diagrammatic and iconic representations of the database schema, which exploits a “graphical notebook” metaphor.
The choice of the iconic representation derives from the suggestions of users who have participated in a preliminard experiment with the TVQE prototype. All users were non-expert in databases (in this paper, we do not report this experiment with more details). The specification of some queries by searching indices in a graphical notebook (see the next section) has been enjoyed by the users more than the direct manipulation on the diagrammatic representation, since s/he prefers to interact with something more familiar. Moreover, icons have a significant metaphorical power and it is shown that iconic VQSs are mainly addressed to users who are not familiar with the concept of data models and may find it difficult to interpret even an E-R diagram.

However, the visual representation of a schema as a notebook is in some sense less rich than the diagrammatic one, since a diagram favour the visualization of relationships between concepts. So, we decided to integrate the diagrammatic and iconic representations, emphasizing the usage of the iconic representation as the interaction media.

Finally, TVQE adopts the top-down strategy to locate the schema of interest in a context tree (see the next section), the browsing strategy to determine the subschema of interest, and the schema navigation strategy for formulating the query. In the visualization of the query result, TVQE will adopt the range selection strategy.

4 The TVQE Environment

We have been developing a VQS for historical databases, namely TVQE, which includes a set of non-temporal and temporal graphical primitives, formally described in [FSC99a],[FSC99b]. In this section, we will show a detailed example of a user-system interaction, which refers to a historical database concerning employment agencies (we will use this example throughout the paper).

We are interested in queries that encompass the possible combinations of current and temporal selection, and current and temporal projection over time and data (temporal selection is a logical condition, based on a predicate that involves the time associated with facts, and temporal projection returns the time values associated with data derived from temporal selection [Jea98]). In this work, the generic notion of temporal selection/projection is specialized into valid-time selection/projection, in order to be applied only in the valid time dimension [Sno87] (the valid time of a fact is the time when the fact is true in the modeled reality). For example, the query “Retrieve the level history of each employee currently working on the project “Interface for TDBs”, only when her/his salary was greater than 5,000”, corresponds to a valid-time projection (level) on a current (project) and valid-time (salary) selections.

The TVQE Interface presents to the user a main window containing some buttons for accessing a database schema plus the following icons: Where? icon, which represents the selection on data; When? icon, which represents the valid-time selection/projection; Data Vis and Time Vis icons, where the objects which are current and historical instances of a temporal class or relationship will be visualized in the Data Visualization window (see below), respectively. The TVQE interface contains three windows: Schema Window, Interaction Window and Data Visualization Window. The three-window arrangement comes from the typical structure of the query specification process constituted by three interaction phases location, manipulation and visualization [CSA93].
In this paper, we concentrate on the Schema and Interaction windows, the Data Visualization window is under development.

### 4.1 The Schema and Interaction windows

The *Schema window* is basically constituted by a workspace on which the database schema is visually presented in three different ways: as a top-down context tree, as a graph (the conceptual schema of a context), and as a subgraph of the query (the three representations share the same workspace). Through the Schema window, the user has a global view of the classes of the schema and their interrelationships.

Since visualizing all classes of a complex schema (which involves hundreds or thousands of concepts) in a single visual structure may be cumbersome, the Schema window displays the database schema as a top-down tree, a so-called *context tree*.

The context tree is based on the top-down refinements of an *E-R* conceptual schema during its design process, by using one of the top-down primitives of [BCN92], which splits an entity *E* into a set of unrelated new entities *E*₁, ..., *E*ₙ. In our approach, we consider such an entity *E* as a *context* *C*. A context represents an abstract class, since it does not contain real-world objects as instances. It can be seen as an abstraction of a set of classes. *C* is refined into a set *C*₁,...,*C*ₙ, where each *C*ᵢ can represent either a more specific context or a class. If *C*ᵢ is in turn a context, it must be refined again.

Hence, the Schema window displays the conceptual schema as a top-down tree, having a context as root and its leaf nodes representing other contexts or classes, as shown in Figures 2 and 3 (left). Note that contexts, classes and temporal classes are visually represented as square nodes, oval nodes and shadowed oval nodes respectively.

---

Figure 2: The Schema and Interaction Windows
The Schema window also displays the database schema either as a graph, called *Graph Schema*, as shown in Figure 4 (left), or as a subgraph, called *Query Schema*, which represents the subgraph of interest comprising only the classes and relationships selected by the user for his/her query. In the graph schema, we have square nodes representing classes of objects (e.g. *PERSON, CAR*), rounded square nodes representing attributes (e.g. *NAME, AGE*), oval nodes representing the relationships between classes (e.g. *LIVES, OWNS*). If these nodes are *shadowed*, they represent the temporal classes (e.g. *EMPLOYEE*), temporal attributes (e.g. *LEVEL*) and temporal relationships (e.g. *JOB*). Moreover, *is-a* relationships are expressed through the edge label “is-a” (e.g. *EMPLOYEE is-a PERSON*) (see [FSSC97b], [FSSC97a], [FSC98]).

In the location and manipulation phases of a query specification, the user acts through the Interaction window. The *Interaction window* is basically composed by a workspace on which a database schema is visually displayed as a graphical notebook, as shown in Figures 2, 3 and 4 (right), where the notebook sheet represents either a context or a class. Whenever the notebook sheet represents a context, the notebook indices represent other contexts or classes, whereas whenever the notebook sheet represent a class, the notebook indices represents all its properties (attributes and relationships).

![Figure 3: The selected nodes EMPLOYMENT-CONTEXT and EMPLOYEE](image)

The user interacts with the graphical notebook by selecting its indices. The effects of the user’s actions are automatically reflected as changes in the global schema in the Schema window. Through the Interaction window, the user selects a class as the *trigger* class of the query [Kim89]. Then, the schema is visually represented as a graph (Graph Schema) in the Schema window.
4.2 Example

We show the interface “look and feel” by an example of a temporal query (the query contains only one trigger class). Assuming that the user is interested in knowing “Which salaries did the employees earn when they changed their level for the last time”. First, s/he accesses a database schema (selecting the Open item in the menu). The database schema is visually represented as a context tree and as a graphical notebook. The notebook indices are two contexts, namely, PERSONAL-CONTEXT and EMPLOYMENT-CONTEXT, which represent information about personal data and employment data, respectively.

Since the user does not visualize the desired class in the first refinement of the schema, s/he may expand a context, just by clicking on it (more precisely, by clicking on the index which represents such context). As a consequence, a new sheet appears with the refinement of the selected context, as shown in Figure 3, where the user has selected the context EMPLOYMENT-CONTEXT. Note that the two windows are synchronized: for each selected index, the corresponding node in the Schema window is also selected. After the selection of a context, the user may switch to a different context (or class). S/he may also return to the previous refinement or to the first refinement of the schema, by clicking on the back or begin buttons.

Since the user has selected the class EMPLOYEE as trigger class of the query, it appears as a sheet and the indices represent its properties, as shown in Figure 4 (note that the system automatically displays the Graph Schema in the Schema window).

![Figure 4: The selected and displayed nodes in the query](image)

The user may expand a relationship in order to let appear the classes and attributes directly reachable from the class EMPLOYEE through that relationship. A related class may be expanded in terms of its properties. The user can continuously change the status of
a node (unselected for the query, selected for the query and displayed for the query result) by clicking on it (clicking on the corresponding index). In our example, JOB and LEVEL are the selected nodes, while SALARY, EMP-ID and NAME are displayed.

At next, by selecting the Create View item, the system generates the appropriate sub-schema. The user can either save the corresponding sub-schema for further manipulation (selecting the Save As item in the menu) or immediately use it in the manipulation phase. The user may create several views by selecting other trigger classes, and relate them in the same query. This approach facilitates the specification of a complex query (with several trigger classes), which can be partitioned into several views.

Our query involves a valid-time selection with a temporal reference to another data, since it compares the history of certain data (salary) with the history of other data (level). So, the user specifies it by selecting the When? icon and the temporal attribute LEVEL. Next, a dialog box with the all history, instant, period options and a temporal reference to.. check box appear (see Figure 5).

Since the history of levels of each employee may contain several time intervals, the user may filter them, by selecting the first, nth-interval or last interval and by specifying whether the selection concerns the begin, end instants, or the duration of each interval. If the duration is selected, the system offers a menu of aggregate functions, min, max, count, avg and sum, which applies to the set of instances in the temporal class or relationship. In the example, the all history option and the begin of the last interval have been chosen, as shown in Figure 5.

![Figure 5: Await condition of a temporal query](image-url)
All sliders of the dialog box are used to specify the temporal condition. The first six sliders are used for specifying either an instant or a period. The number of active sliders depends on the granularity previously selected by the user (due to space limitations we do not address the time granularity issues in this paper). The next slider is used when the user selects the instant option. Let \( t \) be a instant. The begin and end operators retrieve only the instances whose lifespan starts and finishes at \( t \), whereas with at, \( t \) must be in between the lifespan of the object.

The next two sliders are used when the user selects the period option, which contains the temporal predicates between time intervals, defined by Allen [All83] (e.g. before). After the user has specified the start and the end of the desired period, s/he may use either the slider which contains the nine operators or the other slider with three operators. The order of the temporal operators in the first slider is based on the neighbors temporal primitives concept, introduced in [Fre92]. For example, Figure 6 illustrates the temporal conditions During and Finishes 08/26/1996 to 05/19/1998. Note that the specified period appears above the corresponding slider. Since the operators finished-by, contains and started-by are not part of such an order, they were included in the second slider.

Figure 6: Two slider movements used in a period

Figure 7 (left) also shows the effects of the user’s selection of the When? icon applied to the temporal relationship JOB. In the example, the user selects the instant option. S/he also selects the temporal reference to... check box and the LEVEL option in the menu of temporal classes, which can be used as temporal reference. As a consequence, the label “temporal reference” appears above the instant slider, as shown in Figure 7 (left). Note that in Figure 6 a new “temporal aggregation?” check box and a list of classes which are related through JOB appear. This happens because JOB relates more than two classes. With this option, the user may collapse some classes from the list, in order to gather the instances of classes which were not collapsed (see [FSC98] for more details). In the example, s/he collapses the class PROJECT.

At this point, the temporal condition has been completely specified, and the user can see the subgraph of interest in the query graph workspace, as show in Figure 7 (right).

Finally, temporal queries generate huge amount of (temporal) information, which obviously does not lend itself to tabular display [KG95]. Through the Data Vis and Time Vis icons, the objects which are instances of a temporal class or role are visualized in the Data Visualization window. The user visualizes the extensional part of the database in a time-oriented two- or three-dimensional visual representation. Concerning the interaction aspects, we use the dynamic query approach [AS94], [Shn94] in order to provide an interactive navigation over time.
5 Implementation Issues

For portability, flexibility and efficiency issues, the prototype TVQE has been implemented in the JAVA object-oriented language (JDK 1.1.6 / JFC Swing), using JDBC as interface to the DBMS. The overall system is actually under implementation, specifically we are currently implementing the Data Visualization Window, based on the dynamic query approach, which supports the display of the lifespan of a class or relationship at different granularities.

In this section, we describe the implementation issues of TVQE, considering the structure of the application classes.

5.1 Classes Structure

The classes structure of TVQE is partitioned in four modules, as shown in Figure 8:

1. **Internal Structure**: this module contains the classes which compose the internal structure (graph) of the conceptual schema;

2. **Visualization**: this module contains the classes which visually represent a conceptual schema;

3. **Temporal Selection (When?)**: this module contains the classes which compose a dialog box used in the specification of a temporal condition;

4. **Conventional Selection (Where?)**: this module contains the classes which compose a dialog box used in the specification of a conventional condition (non-temporal).
5.2 Internal Structure Module

We use a graph as internal and homogeneous structure, in order to facilitate the visualization of the conceptual schema as a top-down tree, as a schema graph, as a query schema, and as a “graphical notebook”, where the nodes represent the classes and the arcs represent the relationship between classes (this structure is maintained throughout the execution of the application).

The internal structure of the conceptual schema comprises the following structural components:

- An array of nodes, where each node contains information about a class, such as: the node identification number; the class label; the class description; the class type (context, class, temporal class, attribute, temporal attribute, relationship, temporal relationship); the class status (unselected, selected, displayed); a boolean value indicating if the node was visited in the navigation; the level of class within the context tree; the ordinal position of class within the level described above; the coordinates X and Y in order to visualize the class in the graph schema; a boolean value indicating if the node was selected (used only in the graphical notebook); a structure representing the temporal condition; a structure representing the conventional condition.

- An adjacency list which represents the arcs of a graph. The adjacency list is an array of linked lists, where each individual list represents the adjacent nodes of a node (the Figure 9 illustrates a graph and the corresponding adjacency list). An arc contains the following information: the origin node; the arc type (normal, heritage, class-context, subclass-class, component-aggregate); an arc pointer.
• The stack used to store the navigation paths and the queue structure, which is used in the *Breadth-First Search* algorithm, in order to traverse the nodes of the graph.

• Information on selected nodes.

![Figure 9: A Graph and its adjacency list](image)

Due to space limitations, we do not report the other modules in this paper, the interested reader can refer to [FSC99b].

## 6 Conclusion

This paper is an attempt to put in an easy-to-use visual form the task of formulating queries on databases. The different kinds of conventional and historical queries are shown in a homogeneous way, inside a global visual environment. This approach facilitates the user in expressing the query, since s/he has not to learn any complex syntax and can incrementally formulate the query, also receiving immediate graphical feedback.

Future work will aim at extending the TVQE with new features, such as: insertion and update of data; multiple visual representations (form-based, iconic) and accessing mechanisms; diverse kinds of visualization of temporal information in the query result, storage of data about the history of interaction, to access and dynamically maintain them with the same mechanisms as the application data, as an effort towards clearly establishing a link between user interaction and modeling and data modeling and querying, also allowing an evaluation of TVQE in terms of usability.

We also plan to increase the expressive power of the graphical primitives on which TVQE is based on, in particular by allowing: the specification of queries which involves negation; the explicit specification of quantifiers; the visual specification of recursive queries.

## References


