CAR-TR-562 July, 1992
CS-TR-2702 Revised March, 1993

A GRAPHICAL QUERY INTERFACE BASED ON AGGREGATION/GENERALIZATION HIERARCHIES

William J. Weiland
Ben Shneiderman *

Department of Computer Science,
Human-Computer Interaction Laboratory &
Institute for Systems Research
University of Maryland
College Park, MD 20742

Abstract

In order for automated information systems to be used effectively, they must be made easily accessible to a wide range of users and with short training periods. This work proposes a method of organizing documents based on the concepts of aggregation and generalization hierarchies. We propose a graphical user interface to provide a more intuitive form of Boolean query. This design is based on mapping the nodes of the aggregation hierarchy to Boolean intersection operations, mapping the nodes of the generalization hierarchy to Boolean union operations, and providing a concrete, graphical, manipulable representation of both of these node types. Finally, a working prototype interface was constructed and evaluated experimentally against a classical command-line Boolean query interface. In this formative evaluation with sixteen subjects, the graphical interface produced less than one-tenth the errors of the textual interface, on average. Significant differences in time spent specifying queries were not found. Observations and comments provide guidance for designers.

* address correspondence to Ben Shneiderman

1. INTRODUCTION

1.1 Introduction

The problems with Boolean query languages have been well documented (Welty, 1985; Borgman, 1986; Reisner, 1988; Greene, Devlin, Cannata, and Gomez, 1990). Providing precisely formulated Boolean queries is syntactically and semantically demanding and unintuitive, and therefore not well-suited to untrained and infrequent users.

This empirical study compares the utility of a prototype graphical query interface with a traditional Boolean query language for users with little training. Our aim is to provide guidance for designers and establish whether such a graphical interface can alleviate problems novice users experience, by reducing the permissible complexity of queries, selecting rather than typing attribute names and values, specifying numeric ranges graphically, and the possibility of making indirect inferences.

1.2 Traditional textual Boolean retrieval

Boolean query systems evaluate set-algebraic expressions, and present the result set to users. This model is certainly functional, since it covers a large class of retrievals. The problems crop up in the areas of learning and use, even for experienced users.

The set algebra is quite simple in principle, based on the designation of sets by keywords (i.e., each keyword names a set of items) and operating on these sets by union, intersection, and complementation. However, the notations for these expressions are often intricate, involving the semantics of the operators and precedence rules that apply when simple expressions are composed into complex ones. These need to be taught carefully, and may require lengthy training (the speed of this learning process varies widely among individuals). In use, Boolean query expressions can be difficult to comprehend — fully understanding them can require users to “play computer,” taxing human processing capacity.

Even when Boolean operators are well understood, they may be confusing, owing to the looseness with which the terms “and” and “or” are applied in everyday language. When negation is included in a Boolean expression, its interactions with union and intersection can be confusing as well. Frequently, in writing Boolean expressions, users make syntax errors (unbalanced parentheses, misspelled keywords, etc.), which must be detected and corrected.

The long time spent formulating queries and frequent errors have led many researchers to seek graphical representations for binary relational queries (Senko 1977), relational joins (Kim, Korth, and Silberschatz, 1988) and variations on the Entity-Relationship model (Czejdo, Elmasri, Embley, and Rusinkiewicz,1990). Attempts to deal with graphical boolean expressions are increasing (Michard, 1982) but no widely accepted system has emerged. Even commercial systems such as IBM’s Data Interpretation System (DIS) provide some form of graphic query formulation.

1.3 A Graphical Query System for novices

In the Graphical Query System (GQS) developed here, ease of learning and use were traded-off against functionality. Rather than provide the full range of queries specifiable by Boolean languages, we provided a useful, but restricted, range of possible queries equivalent to a tree of conjunct of disjuncts (groups of “or”s connected by “and”s). Negation was not supported.

To reduce learning, the GQS was designed to take advantage of "common sense" conceptual models of the world and apply a graphical, direct-manipulation appraoch. At a conceptual level, GQS relies on users' intuitions about the differences between categories (classes) and attributes to decide whether union or intersection is an appropriate operator in a particular context (so, fundamentally, the system is still based on Boolean queries, but automates much of the decision making involved in formulating a query).

The usability of GQS is enhanced by its reliance on visual recognition of items of interest, rather than recall of applicable keywords and the syntax of the query language. Query formulation is a refinement process; as each "operation" is performed, its effect is displayed immediately, and the operation can be undone. Unlike textual Boolean query languages, there are no syntactic errors a user can make (and consequently, no error messages for users to struggle with). The current state of a query is represented in its entirety on-screen, so that there is no need to remember what operations were performed in reaching the current state. Finally, the interface supports classical searching, where the objects of interest can be described via search terms, as well as browsing, which relies more on exploring the information base.

Wong and Kuo (1982), in their description of the GUIDE system, list five points they believe "are the major reasons for the difficulty in using and understanding query languages":

1. Excessive reliance on user recall of names, data formats, and units.

2. Data models that require users to describe implicit relationships via explicit queries (e.g., joins in relational systems).

3. Lack of feedback in the process of formulating queries.

4. No facility for suppressing unnecessary detail.

5. Lack of overview or browsing facility for meta-data (the schema).

GQS addresses the first, third, and fourth points directly. All interaction is via pointing and selecting, rather than recall and data entry. Numeric quantities are represented visually, so that correct units need not be remembered, and scales have meaningful labels. For example, a timeline may be labeled in years, but marked with historical events as reference points. A cost scale may be labeled in dollars, pounds sterling, and francs, avoiding the need for conversions. Feedback is continuous as queries are developed. The database is represented as a hierarchical structure that can be browsed from the top level, with lower levels expanded only on request. Since the data model used in this system is simple, the issue of implicit relationships (second point) is not addressed. However, the emphasis on graphics (such as maps) makes many implicit relationships readily apparent, and in fact makes available relationships that could not have been foreseen in developing a database schema. The fifth point is addressed insofar as the hierarchical structure is the meta-data, and can be browsed.

2. DESIGN OF THE GRAPHICAL QUERY SYSTEM

We begin with an example of the working system to offer readers an understanding of our goals.

2.1 An example

A hypertext currently in use, GOVA (Guide to Opportunities in Volunteer Archaeology) is an ideal candidate for use with this type of query interface (Plaisant, 1990). It is well structured, and contains a great deal of information in the form of dates, costs (of participation in various projects), and geographic information (currently embedded in maps). It consists of almost 300 articles about particular archaeological projects, periods of history, civilizations, and miscellaneous information.

Figure 1 shows the starting screen configuration for the GQS as it appears on a Macintosh II. The topmost node in the hierarchy, an aggregation node, is represented by the box labeled *Sites* in the upper left corner. The attributes of this aggregation node are *Location*, *Peoples*, *Period of History*, *Cost to Attend*, and *Excavation Dates*. The result box, also labeled *Sites* to reinforce its relationship to the source of the result list, appears on the right hand side. Nothing is selected in the *Sites* box in the upper left. The value of a null query is considered to be everything contained within the node, in this case, the entire database. Thus, the result list contains the names of all articles in the *Sites* database.

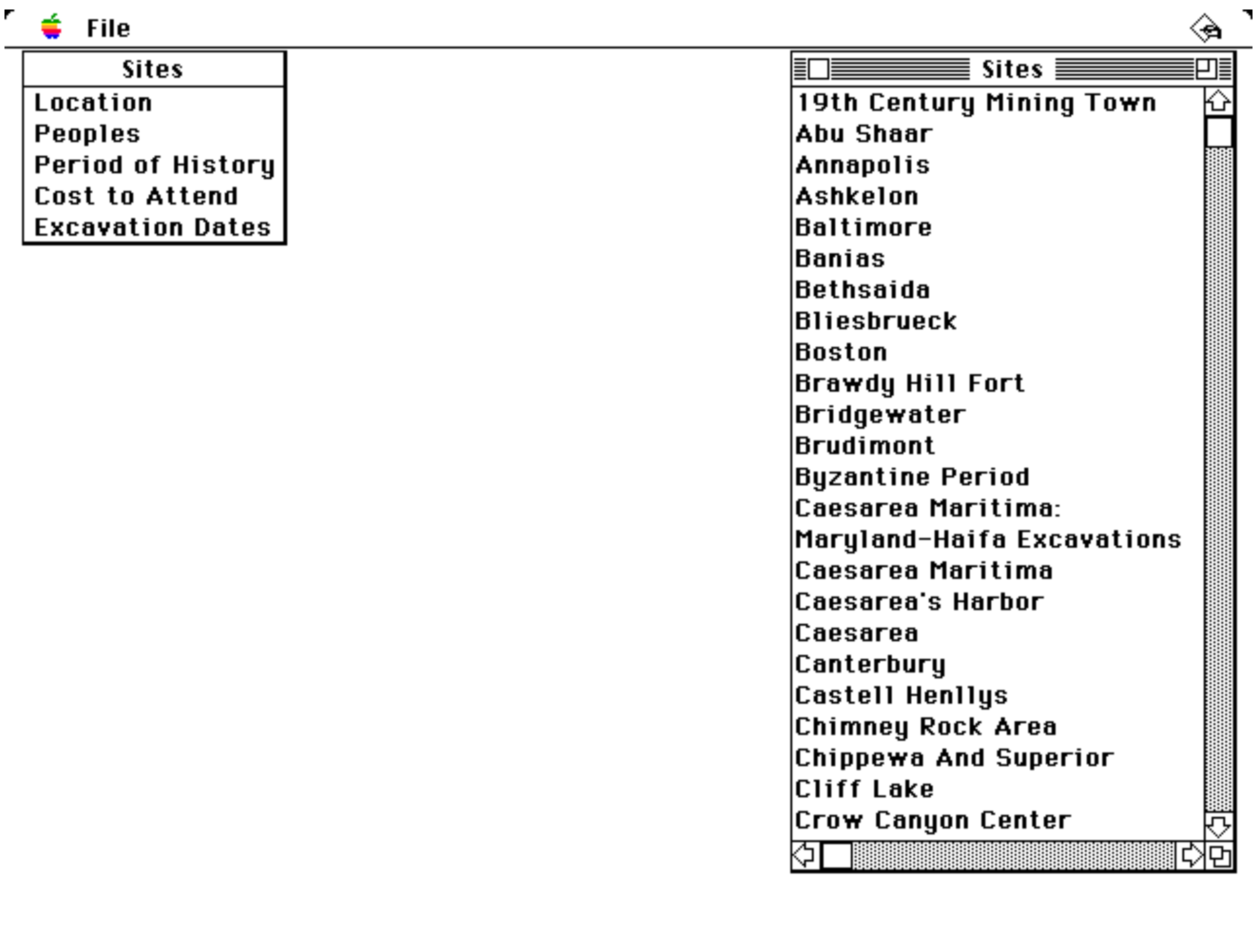

Figure 1. Initial Screen Configuration

Figure 2 shows the selection of the *Location* attribute from the *Sites* box which has caused *The World*, a map of the world, to appear. The domain of this box is not numeric; it displays discrete selectable elements that are regions of the world. *Mediterranean* has been selected and other elements such as *Middle-Eastern Sites* are not selected. This has immediately reduced the result list to four items: the four Mediterranean sites represented in this database. *The World* is a generalization box, where the classes represent rough geographic partitions of sites. Any elements selected in this box will be unioned.

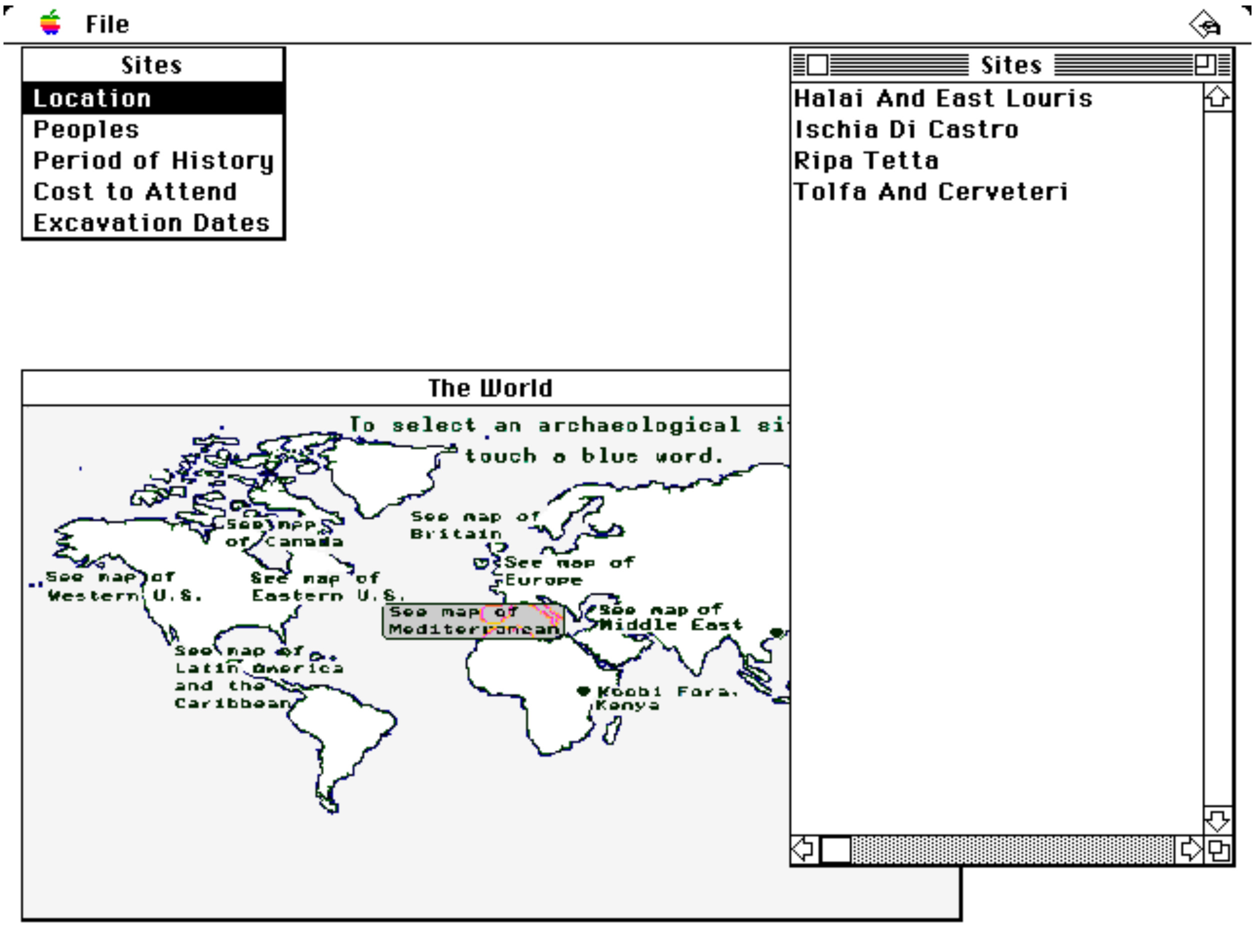

Figure 2. Mediterranean Sites Selected

In Figure 3, the Middle Eastern sites have been selected as well (by clicking the associated label). The result list has lengthened considerably, to contain all of the sites that are either Mediterranean or Middle Eastern. In addition, *Periods of History*, an attribute of *Sites*, has also been selected and opened, to produce another generalization box. This selector box, labeled *Sites by Period*, contains a list of periods of history associated with individual sites. *Hellenistic* and *Roman* have been selected. The query currently represented corresponds to the Boolean query: (*Mediterranean* OR *Middle-Eastern*) AND (*Hellenistic* OR *Roman*).

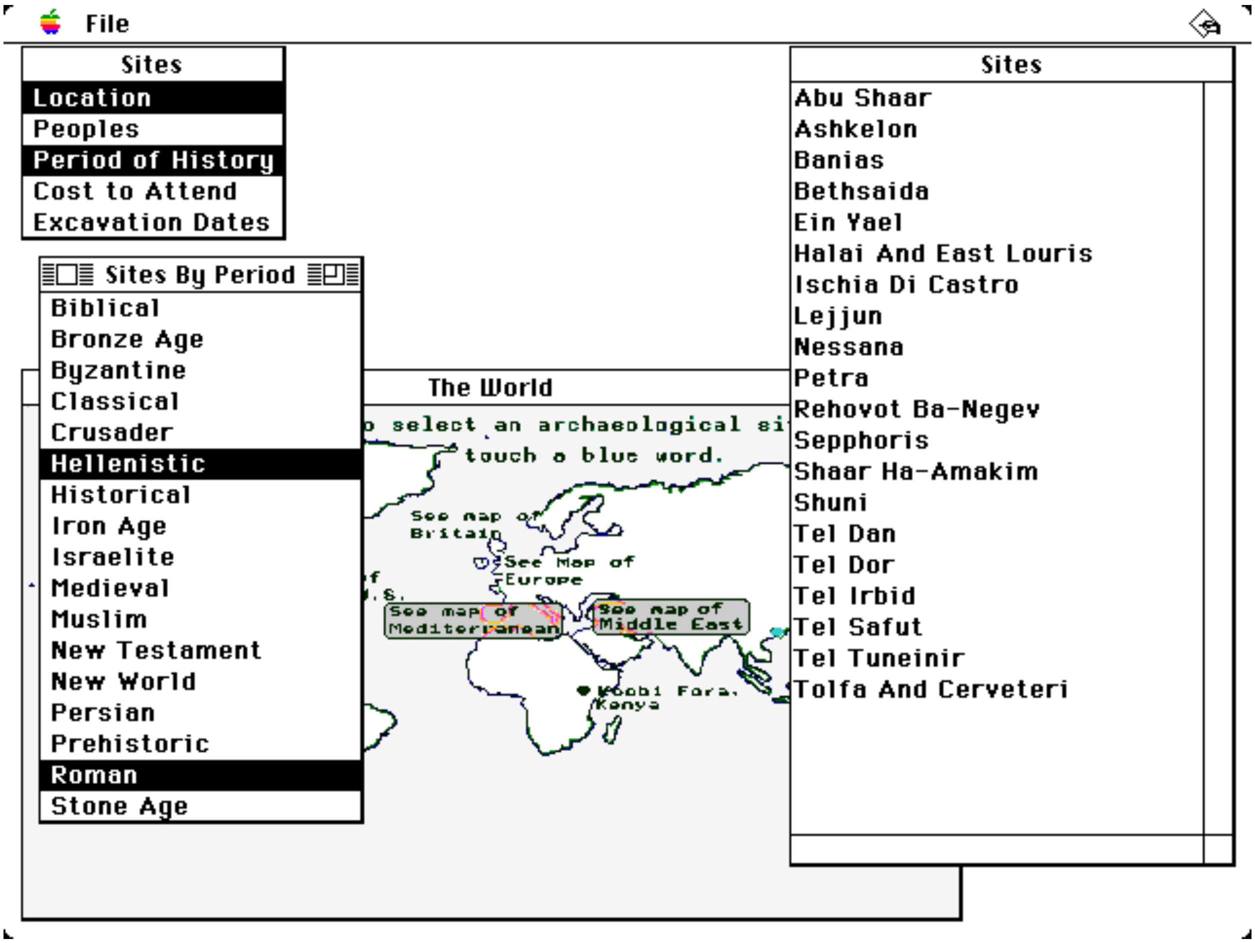

Figure 3. Hellenistic and Roman Sites intersected with Mediterranean and Middle Eastern Sites

In Figure 4, the *Cost to Attend* attribute of *Sites* has been selected, producing a selection box, *Cost*. This box displays dollar quantities along a vertical bar, from which ranges can be selected. The range of approximately $1,000 to $2,500 has been selected, by dragging the mouse from one end of the desired range to the other. This now expresses the textual query: (*Mediterranean* OR *Middle-Eastern*) AND (*Hellenistic* OR *Roman*) AND (1,000 ≤ *Cost* ≤ 2,500).

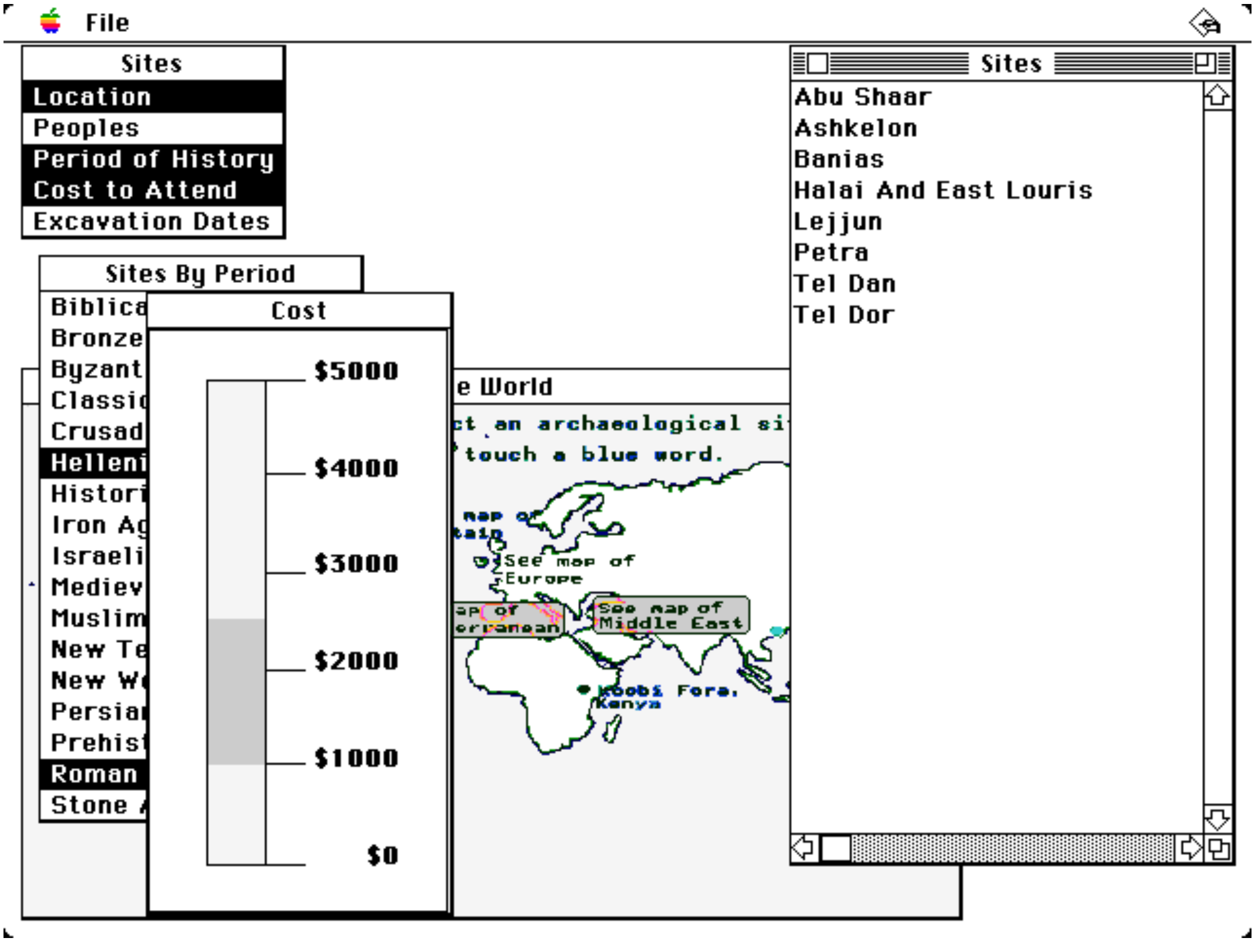

Figure 4. Selection of Cost Range

In Figure 5, the *Excavation Dates* attribute has been opened, and the months April and May selected. The imposition of a further constraint has caused the result set to shrink again, listing all sites that are available in April or May. The query expressed is now: (*Mediterranean* OR *Middle-Eastern*) AND (*Hellenistic* OR *Roman*) AND (1,000 ≤ *Cost* ≤ 2,500) AND (*April* OR *May*).

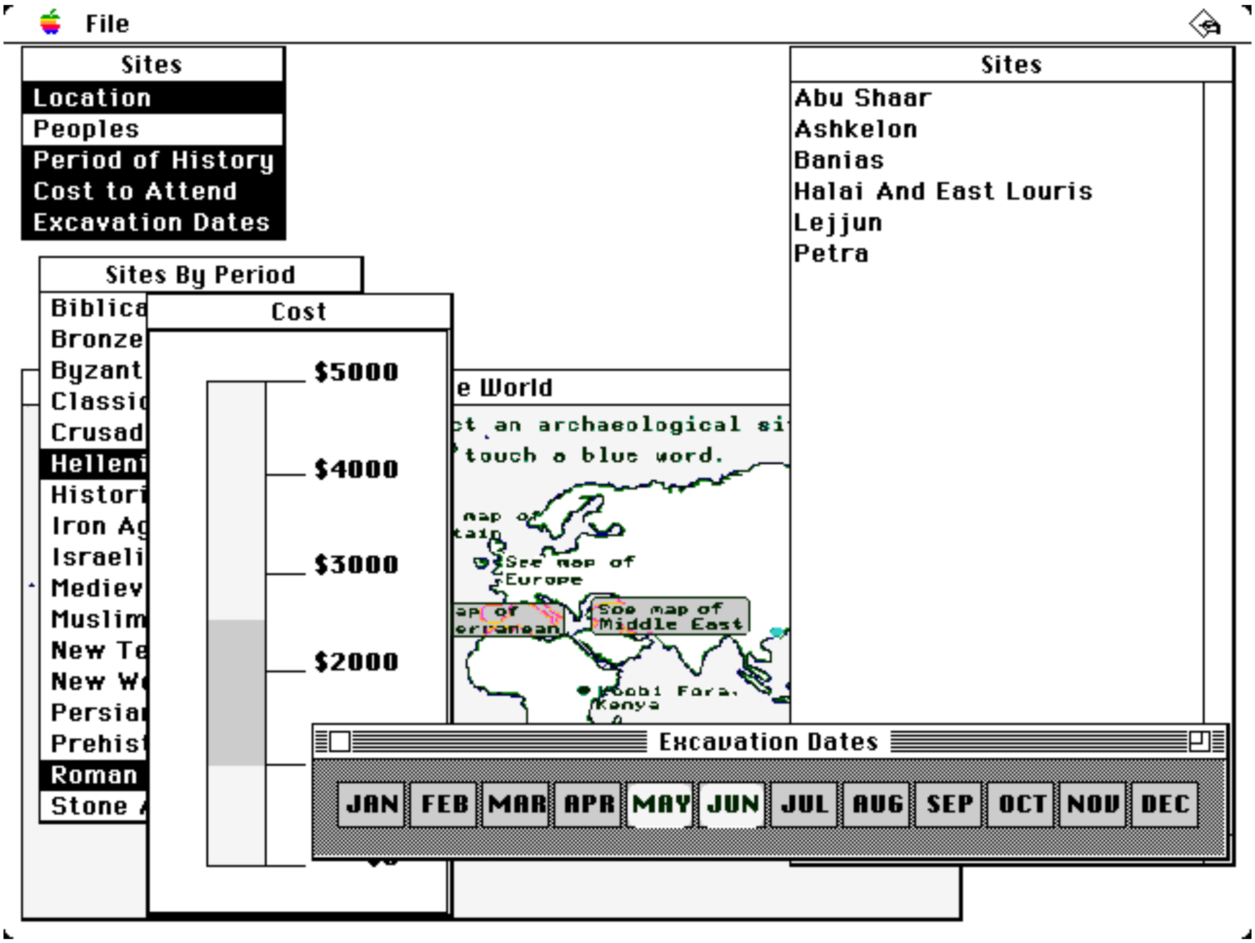

Figure 5. Excavation Dates Selected

2.2 Data Model

The relational model provides an elegant approach to representing structured data. By expanding on Query-by-Example (QBE) (Zloof, 1975), a simple yet powerful query interface could be constructed by displaying attributes, providing graphical selection among them, and specifying logical connections. Our approach is based on a object oriented class hierarchy of relations and attributes. We associate implicit Boolean connectives with this hierarchy, permitting an intuitive yet powerful user interface to be developed. We adopted Smith and Smith's (1977, 1980) high-level view of data modeling that describes the relationships between objects to be modeled in terms of a pair of orthogonal hierarchies, aggregation and generalization. These concepts, now popularized in object-oriented data models (Kim, 1990; Bertino & Martino, 1991), are in harmony with human factors principles (Chen, 1992).

An *aggregation* represents a relationship between objects as a higher-level object. This in fact is the usual way that relational databases are structured. For example, a parts shipment may consist of a supplier, part number, quantity, etc. These objects, known as *components*, are composed into a higher level object — the shipment.

A *generalization* represents a combining of types. The types *dogs*, *cats*, *pigs*, etc. can be generalized to *animals*. Two types within a generalization must be disjoint (something that is a dog cannot be a cat).

The basic mechanism of aggregation already exists in relational systems, but the generalization mechanism has to be constructed. The approach taken by Smith and Smith involves having a particular set of relation names be the domain for an attribute of an aggregate structure. The relations for any one of these domains are required to be disjoint — they represent disjoint subclasses. It is possible to provide separate subclass domains in an aggregate that are not disjoint. An example of this object oriented approach would be division of vehicles by propulsion type (human, jet, internal combustion engine, etc.) and by medium type (air, water, land, amphibious). Every vehicle should appear in both of these categories, but each in only one of the subtypes of a category.

The reason this structure was deemed desirable is that it provides a traversable tree of aggregations and generalizations (the term *hierarchy* is used here to mean that the structure may take the form of a semi-lattice, where branches can merge back together). This tree has the property that there are two types of nodes: aggregation and generalization. The aggregation nodes have a list of attributes that may be atomic (their domains are numeric or textual) or have subtype domains. The generalization nodes have lists of disjoint subtypes. The connections between aggregation nodes and generalization nodes occur on these subtype attributes. In addition, the hierarchy nodes are used to define implicit Boolean operations on their children. The aggregation nodes define intersections; the generalization nodes define unions.

The aggregation nodes correspond to conjunctive query terms. It makes sense to find all vehicles that are red *and* cost less than $20,000. For the case of aggregation nodes, it would also make sense to find all vehicles that are red *or* cost less than $20,000; however, this appears less likely to be useful. To provide some justification for this statement, the constraints users place on one attribute are usually firm, and are not superseded by constraints on another independent attribute. If someone can spend no more than $20,000 on a car, the fact of its being red does not (usually) resolve that constraint. A design decision made in the development of this system is that aggregation nodes are conjunctive. This is intended to provide a plausible simplification of the conceptual model. Although this is a limitation, it will be shown later that this is not a serious one. Through a slight complication of the interface, it can be extended to permit disjunctive queries across attributes of an aggregation node, although for the intended users of this system, it is not likely to be necessary.

There are some important subtleties that further reinforce the choices. One of the purposes of the Smith and Smith work was to define an abstraction that could capture aggregate objects, generic objects, and individual objects (instances) in a uniform way (so that generic objects could participate in relationships just as individuals do). This was accomplished by making relations themselves (in practice, relation names) elements of attribute domains. In the previous example, the domain of attribute *propulsion category* was the set of relations containing instances of vehicles of particular propulsion types. So, the domain of propulsion category might be the set of relations {*human-powered vehicles*, *jet-powered vehicles*, *internal-combustion-engine-powered vehicles*}.

This puts relations on an equal footing with primitive data elements, like numbers, in that both can be members of attribute domains. Primitive attributes are selected disjunctively (that is, when elements of an attribute domain are selected, the selection is disjunctive). If for example, the range of costs from $1,000 to $2,000 is selected in a query, this represents the union of all possible cost values in this range. Generalization nodes are discrete attribute domains. To be analogous to the continuous domains, selection from these should also be disjunctive. Thus, if human-powered and jet-powered vehicles are selected, this would represent the union of these two choices.

The object oriented scheme chosen offers a number of advantages. It permits the display of a small, meaningful view of the database at a suitable level of abstraction for a user's purposes; generic objects can be viewed at an abstract level, or examined at more detailed levels. It supports more graceful extension (by adding subclasses), an important consideration for textual information bases, which may grow in unpredictable ways.

One last point about generalization nodes is that for the purposes described here, the classes need not be totally disjoint (although this violates Smith and Smith's rule). The purpose of doing disjunction on generalization nodes is that it usually makes more intuitive sense to do so when there are classes that appear disjoint to users. This occurs in the Guide to Opportunities in Volunteer Archaeology. Site articles are classified by their period of history; for the most part, these are disjoint, although some sites do span several historical periods.

2.3 Query Interface

The GQS interface avoids abstruse formalisms and permits exploration of query options, with immediate feedback of results, and the possibility of easily retracting mistakes (in the sense of producing undesired results; ideally, mistakes of syntax should be impossible). We believe GQS helps control of complexity by conveying a suitable, concise conceptual model (Norman, 1983, 1986).

The GQS conceptual model has been mapped into a display ("system image") by associating individual, small, freely-overlapping windows (henceforth referred to as "boxes") with the nodes of the aggregation/generalization hierarchy. These boxes can be opened and closed to reveal or suppress parts of the hierarchy (allowing for complexity control). The nodes of the hierarchy, that correspond to query terms, are depicted within these boxes as diagrams or lists representing the domain of the query term in a meaningful context (e.g., location domains are represented on maps; numeric domains are represented as sliders; discrete domains are represented as lists of textual items or selectable elements on a graphical image).

Choices are made within these boxes (on screen, with a pointing device), that permit traversing the two-dimensional aggregation/generalization hierarchy, until leaf nodes of the hierarchy are reached. The collection of boxes on screen completely determines the current query, regardless of the order in which they were brought up. The result of the current query is maintained in a result box, as a list of the articles titles.

GQS is procedural in the sense that queries are developed in a step-by-step process, with immediate feedback as to the results of each step. As items in a box are selected, their contribution is immediately reflected in the result list; such items can be deselected, to retract the last change, or opened up to show their associated query boxes, to examine lower levels of the hierarchy. This provides for an environment in which exploration is rapid, easy, and safe. Any action taken can be undone; users are never presented with error messages, because there are no illegal actions. While the boxes themselves represent formal constructs, users deal with an extremely concrete representation and have no need to translate back-and-forth between formal language concepts and a textual representation, as is the case with text-oriented retrieval languages. The design relies solely on recognition, rather than recall (of keywords, formal syntax, or the current state of the query) — salient information is represented explicitly in the display.

2.4 Interface features

The version of GQS that was tested is described here, and illustrated in the example that follows. Originally, a more elaborate interface had been conceived, which supported negation, disjunction on aggregation nodes, and a limited form of join operation; this will be described later.

There are three types of query boxes — aggregation, generalization, and selection. The aggregation boxes present the name of a relation, followed by a list of its attributes (analogous to QBE, except vertically instead of horizontally). Individual elements of this list may be selected (and deselected, at will). When an "open" command is given (via a menu selection, command-key sequence, or mouse double-click), selected attributes produce the associated generalization boxes and/or selection boxes.

When the domain of the selected attribute is a relation name, it represents a set of (mutually exclusive) sub-classes of the previous relation, and thus brings up a generalization box, which contains the list of sub-classes of the selected attribute. For other domains, a selection box appears, which contains a representation of the domain (e.g., a map for a domain of coordinates, a timeline for a domain of years, or possibly just a list of discrete items). These boxes provide users with a graphical way of specifying range queries. As mentioned above, subclasses are just another domain, which may be represented in many ways (including graphically). For this reason, generalization and selection boxes are almost the same; selections in either are treated disjunctively. Individual boxes may be deactivated by deselecting the associated term from the parent box (they remain on screen, but don't contribute to the query), closed (they disappear from the display), as well as being dragged about to organize the screen space.

In a query, boxes represent either range selections on an attribute or the union of individual selections (as from a list). The value associated with each box is the set of items it selects. The value of the parent aggregation box of a set of selection (or generalization) boxes is the conjunction of their values. The parent generalization box of a set of aggregation boxes represents the disjunction of their values. This proceeds up the tree to the root. The value of the root box is presented in the result box, as a list. Queries are conjuncts of disjuncts; additional features such as built-in functions or matching of attributes to produce joins is not currenty supported. Query nesting and result saving are not supported, but since queries are easily modified and rapidly applied, nesting or saving are not required except for the potential execution efficiencies.

In GQS each box, like each node in the hierarchy, represents a relation. The value of a selection box is a selection of tuples from its relation, according to the selected values. As results are passed up the hierarchy through generalization nodes, the extraneous attributes (those not part of the generic object) are projected away, to make them union-compatible. The root relation simply contains all of the items (seen as generic items, whose only attribute is a name), and fields for one or more subclass domains. Each domain in the database, including each of the subclass domains, are associated with "selectors" used to display them and provide for selection; thus a distinct domain must be used for each different selector desired — this can be accomplished by imposing some kind of type structure on the domains, and providing for procedural attachment.

2.5 Interface conventions

The GQS prototype was implemented on an Apple Macintosh computer, with a thirteen-inch color display. It closely obeys the standard Apple interface conventions, except with regard to the selection of multiple items. Opening and closing boxes (windows) can be accomplished by options from the "File" menu (as in Figure 1), by means of command-key equivalents (Command-O for "open", Command-W for "close", as per the standard) — commands issued this way apply to the topmost ("active") window. In the figures below, the active window is the one which shows a patterned title bar, and obscures any other windows it overlaps. Windows may also be closed by a mouse click in the "close box" in their upper left corner. Two rapid clicks in succession (a "double click") on a selected item will cause it to open (show its associated box); double-clicking an unselected item will select and then open it. When multiple items have been selected, a double-click will open all of them.

The selection of items is non-standard. Normally, multiple selections are accomplished by clicking while holding the shift key; in GQS, clicking an item once selects it, clicking it again deselects it, without affecting any other selected items. This was done because multiple selections are expected to be the norm, rather than the exception. On maps, multiple selections can be accomplished by clicking and dragging; a rubber-band rectangle follows the mouse cursor. Anything within the rectangle will then be selected. Deselection is accomplished by clicking a single item. Additionally, some displays allow continuous range selection; this is accomplished by the same click-drag motion. One additional departure from convention regards active versus inactive windows. Normally, a window must be clicked once to activate it, before other mouse actions can be registered by it. In GQS, to facilitate selections in more than one window, the activating click will also select items.

3. EXPERIMENTAL EVALUATION

3.1 Hypotheses

The Graphical Query System has been designed to provide a query formalism that matches users' conceptual models of information seeking, eliminate the need to remember (or guess at) keywords, provide a direct mapping of numerical value ranges to users' mental models, and to make use of the human capacity to extract visually contextual data (from maps, etc.) that is not readily represented in a data model. Achievement of these design goals would be apparent from improvements in performance *vis à vis* textual Boolean query interfaces in information seeking activities. *Error rates* correspond closely to the match between users' conceptions of the functioning of a query mechanism and its actual functioning. The time to formulate a query — *think time* — provides a measure of the cognitive load imposed on users, together with the overall ease-of-use of the interface.

Query types of particular interest in assessing how well these design goals have been met are: *complex queries*, requiring significant understanding of the query formalism; *range queries*, where the search is to be constrained by the values of certain numerical attributes; and *indirect queries*, where information is sought that is present, but not explicitly represented, in the database (e.g., "Find all sites in countries bordering on France"). We conjecture that GQS users should do especially well with these types of queries.

The Graphical Query System would be expected to provide higher accuracy than the text interface for complex queries, owing to the difficulties users have with the precise meanings of Boolean connectives, as well as syntactic issues like operator precedence — these difficulties should have a greater effect as the number of connectives required to specify a query increases. GQS's graphical interface should provide shorter think time for range queries, since it requires visual selection rather than formulating and typing queries, and higher accuracy because there is no opportunity for a user to provide the wrong connective (e.g., for "find the sites costing between $1000 and $2000," entering (cost >= 1000) *or* (cost <= 2000) instead of (cost >= 1000) *and* (cost <= 2000)).

The graphical interface should also improve accuracy and think time for indirect queries, because for the text interface, these require drawing on prior knowledge (of geography in the GOVA example), which may be incomplete or difficult to recall. The text interface may have an advantage in direct queries, which can often be formulated on the basis of a purely syntactic understanding of the problem (e.g., finding all of the articles about Zambia using a sequence of maps would be difficult for someone with no idea where Zambia is, but would be easy using Zambia as a keyword in a textual query language). The text interface could also be expected to have an advantage for simple logical queries, because it can be simpler to type the terms used in such queries rather than locate them by traversing the hierarchical structure of the information base. However, in real-world queries, it is frequently the case that the terms to be used are not known, even in simple queries — in such cases, the hierarchy would provide a quick way of locating them.

3.2 Experiment Design

The independent variables in this study are the type of interface (graphic or text), the type of query (direct or indirect, logical or range query, as described previously), and the level of difficulty ("simple," requiring at most one connective, or "complex," requiring two or more connectives). The dependent variables of interest here are the time to formulate a query for a given problem and the number of queries formulated incorrectly. The time taken to translate a question posed in English to a query provides a rough measure of the cumbersomeness of the interface mechanism and the difficulty of comprehending the formalism used. The experiment was a counterbalanced, within-subjects design, with half of the subjects using the graphical interface first, the other half, the text interface. In addition to the interface type, there are three additional within-subjects factors: difficulty of query (simple, complex), type of selection (logical or range), and type of inference (direct or indirect).

3.3 Subjects

The desired subject population was one relatively naive of computer science and formal query languages, yet not so technically unsophisticated as to be unable to use a computer keyboard or mouse. In order to achieve this goal, they were screened on the basis of computer experience (those with programming experience, coursework in computer science, or database management/information retrieval experience were excluded). Furthermore, it was decided that subjects who demonstrated significant difficulty with the mechanics of using the computer during the training session were to be removed from the study, although this did not occur. The sixteen subjects ultimately chosen were all undergraduates at the University of Maryland, and were drawn from introductory Psychology courses.

3.4 Method

The apparatus used consisted of an Apple Macintosh II computer with 13-inch color monitor, and the standard keyboard/mouse configuration. The THINK C object-oriented development environment was used to construct an experimental platform. This platform consists of a graphical browser, incorporating the GQS interface features, and a simple textual Boolean query interface, both of which can search a database of articles from the Guide to Opportunities in Volunteer Archaeology. The platform also provides a mechanism for controlling the experiment (prompting the experimenter with a randomly generated sequence of query numbers) and logging the presentation and completion of queries, as well as the results generated by the queries (and in the case of text queries, the actual text strings entered by the subject).

The textual query interface implements the significant features of most Boolean query systems: conjunction, disjunction, and negation over relational expressions (the name of a numeric attribute, a relational operator, and a scalar (e.g. cost > 1000), or the reverse order). Keywords and relational expressions are atomic and represent sets of items either associated with the keyword or having the given attribute within the specified range. Boolean expressions are formed by applying the connectives & (and), | (or), and ~ (not) to these atoms or (recursively) to other expressions. These connectives perform the operations of set intersection, union, and complementation (with respect to the universe). Their precedence conforms to the standard convention — from highest to lowest precedence, negation, conjunction, then disjunction — and may be controlled by the use of parentheses. Lexically, letter case and spaces are ignored (except within <=, >=, and <> operators). A typical query (CLD from Table 1) was typed as: (Israel | Egypt) & (April | May).

The textual query interface resembles a standard command line interface (as might be found on an alphanumeric terminal), but makes use of the standard Macintosh conventions for selecting, cutting, copying, pasting, and deleting text. Although not strictly necessary for the tasks given here, this provides somewhat greater flexibility in editing or correcting queries. When a syntactically incorrect expression is typed, or one that refers to non-existent keywords/attributes, the system automatically selects and highlights the offending text, which speeds the process of identifying and correcting errors.

As described before, there are four query types and two complexity levels; one example of each query-type/complexity-level was tested, resulting in a total of eight queries for each interface type. For each interface type, the queries were presented in random order. For the two interface types, the same queries were presented. The original intent was to disguise queries through changes in keywords used, but it proved to be difficult to provide queries of identical difficulty this way, and subjects appeared to have no significant improvement when encountering a query the second time.

Training was provided prior to using each interface type, and consisted of a brief introduction to the use of each interface type and working through four examples (identical for each interface) with the assistance of the experimenter. Table 1 provides a list of the test questions used. Since the experiment involved queries to a geographic database, there were some minor problems with lack of geographic knowledge by the subjects, but no attempt was made to compensate for this — in the intended applications of these systems, a certain amount of ignorance of the subject matter is virtually guaranteed, and the ability to locate information under such circumstances is a desirable feature of such a system. The problem here is analogous to that of finding in a dictionary a word one can’t spell. The ideal is for the system to make this possible.

SLD: Find all sites in France.

SLI: Find all sites in countries bordering on France.

SRD: Find all sites that cost less than $1000.

SRI: Find the ten most expensive sites.

CLD: Find all sites in Israel or Egypt that are available in April or May.

CLI: Find all non-Native American sites on the West Coast of the United States.

CRD: Find all sites costing between $1000 and $2000 that are available in June.

CRI: Find a month with more than 10 sites that cost less than $1000.

Table 1. Test Questions

An answer was graded correct if and only if the list of documents retrieved matched the correct answer. The documents themselves were not available for inspection, in order to make sure that only the available keys and attributes were used in locating the desired information, rather than some type of browsing approach to search. In both the graphical and text cases, the list of documents retrieved was shown to the subjects as they formulated their queries, to provide feedback for query refinement. Subjects were permitted an unlimited number of attempts at formulating queries with the text interface, until they were satisfied their answer was correct (or they gave up). Together with the feedback mechanism, this approximated the progressive refinement approach integral to the graphical query interface. In itself, query by progressive refinement appears to be a powerful technique for improving performance in formulating queries, whatever the style of interface used (Oddy, 1977; Welty and Stemple, 1981; Williams, 1984). Since it could not be eliminated from the graphical interface, it was in effect added to the text interface to avoid having the effects of this feature swamp the effects of interface style, which was of greater interest in this study. The response time was measured from presentation of the question to the time the subject indicated that the query being formulated was complete.

3.5 Results

The mean times and error rates for the two interfaces, grouped by order of presentation are given in Table 2 (standard deviations appear in parentheses). *N* in this table refers to the number of subjects participating in each treatment, which is half of the total number of subjects. Each subject answered eight questions for each treatment. Standard deviations are in parentheses.

Treatment (N=8)	time (sec)		errors (per query)	
Text-first	114	(71)	0.42	(0.50)
Text-second	95	(43)	0.61	(0.49)
Graphic-first	101	(55)	0.03	(0.16)
Graphic-second	88	(47)	0.09	(0.29)

Table 2. Mean Response Times and Error Rates, by Interface Type and Presentation Order

As can be seen, the mean response times and error rates for graphics were lower than those for text; however, as will be shown later, the time differences are not statistically significant at the .05 level, while the error rate differences are.

In analyzing the experimental results, it was first necessary to determine whether the presentation-order effects were significant. A four-way analysis of variance (ANOVA) was run, with interface type and the three query factors as within-subjects factors, on the time and error variables. Partial results of this ANOVA, presented in Table 3, show a highly significant order effect. Rather than risk distorting the results, it was decided to analyze the experiment as a split-plot design, by discarding the data for the second trial of each interface. The interface type was then treated as a between-subjects factor, with the three query factors within-subjects.

An ANOVA was run with the new design, which resulted in the values shown in Table 4, for time, and Table 5, for errors. Significant P-values (at alpha = .05) are indicated in boldface. The effects are labeled TG for interface type (text-graphic), SC for query difficulty (simple-complex), LR for selection type (logical-range), and DI for inference type (direct-indirect). The use of ANOVA for dichotomous dependent variables (such as the error variable here) is supported by the literature. According to Lunney (1970), ANOVA is sufficiently robust to analyze this type of data (SS=sum of squares, DF=degrees of freedom, MS=multiples squares, F= F statistic value, P=probability):

Effect	SS	DF	MS	F	P
Time:					
Order	17161.0	1	17161.0	6.58	**0.011**
Error:					
Order	1.0	1	1.0	7.23	**0.008**

Table 3. ANOVA Results for Time and Error Order Effects

As can be seen, the main effects of interface type on the time variable were not statistically significant. There were however, a number of statistically significant query-type effects and interaction effects.

The interface type showed a highly significant main effect for error; as Table 2 shows, the error rate for the graphical interface was lower overall than that for the textual interface; the main effect for interface type (TG) of Table 4 indicates this difference to be significant.

Table 4 shows a number of significant main and interaction effects for the time dependent variable. The significant main effects are for complexity (SC), selection type (LR), and inference type (DI). Additionally, complexity and selection type (SC x LR) show a significant interaction, as well as all three query factors together (SC x LR x DI). The interface type participates in no significant interactions.

Table 5 shows several significant main and interaction effects for the error dependent variable. In addition to the main interface-type effect, mentioned previously, there is a significant main effect of query complexity (SC) on errors. There are also interaction effects of complexity and inference-type (SC x DI), complexity, inference-type and interface-type (TG x SC x DI), the three query factors (SC x LR x DI), and all independent factors (TG x SC x LR x DI).

Effect	SS	DF	MS	F	P
TG	40100	1	40100	0.467	0.505
SC	316125	1	316125	18.350	**0.001**
LR	85703	1	85703	5.112	**0.040**
DI	171189	1	171189	6.901	**0.020**

TG x SC	3	1	3	0.000	0.990
TG x LR	1106	1	1106	0.066	0.801
TG x DI	77145	1	77145	3.110	0.100
SC x LR	100648	1	100648	5.585	**0.033**
SC x DI	30888	1	30888	1.621	0.224
LR x DI	588	1	588	0.049	0.828
TG x SC x LR	33581	1	33581	1.864	0.194
TG x SC x DI	23793	1	23793	1.248	0.283
TG x LR x DI	315	1	315	0.026	0.874
SC x LR x DI	222548	1	222548	10.078	**0.007**
TG x SC x LR x DI	11610	1	11610	0.526	0.480

Table 4. ANOVA Results for Time

Effect	SS	DF	MS	F	P
TG	4.883	1	4.883	42.476	**0.001**
SC	0.633	1	0.633	5.505	**0.021**
LR	0.195	1	0.195	1.699	0.195
DI	0.070	1	0.070	0.612	0.436
TG x SC	0.195	1	0.195	1.699	0.195
TG x LR	0.195	1	0.195	1.699	0.195
TG x DI	0.070	1	0.070	0.612	0.436
SC x LR	0.008	1	0.008	0.068	0.795
SC x DI	0.633	1	0.633	5.505	**0.021**
LR x DI	0.070	1	0.070	0.612	0.436
TG x SC x LR	0.008	1	0.008	0.068	0.795
TG x SC x DI	0.633	1	0.633	5.505	**0.021**
TG x LR x DI	0.008	1	0.008	0.068	0.795
SC x LR x DI	0.633	1	0.633	5.505	**0.021**
TG x SC x LR x DI	1.320	1	1.320	11.485	**0.001**
ERROR	12.875	112	0.115		

Table 5. ANOVA Results for Error

3.6 Discussion

None of the hypotheses for the time variable can be confirmed from the results. The main effects of complexity (simple queries took significantly less time than complex ones), selection type (range queries took less time than logical ones), and inference type (direct queries took less time than indirect ones) suggest that the factors identified were interesting from the standpoint of response time. Query complexity had an especially strong effect; complex queries have been identified as the main “sticking point” in many studies involving query languages.

However,the time spent was certainly no worse than for the Boolean languages; in a real-world situation, users would not have been provided the kind of assistance with managing syntax errors and identifying suitable keywords that they were in this experiment. Had the latter been eliminated, it is possible that the graphical interface would have proven faster overall, because such assistance was not provided for (nor was relevant to) the graphical interface. The experiment was intended to test for an improvement because of the different *cognitive* styles of the two interfaces;

the well-known problems of unfamiliar syntax and keyword sets that plague Boolean query languages were not at issue.

The error rates provide more encouraging results. One of the main hypotheses of the experiment, that the graphical interface would result in lower error rates than the textual one, was confirmed at alpha = .05 ($P < .001$). As mentioned in the discussion of the time results, the experiment was intended to uncover differences for cognitive, rather than mechanical or syntactic, reasons. Errors in the textual interface were strictly the result of misunderstanding the formalism, not mistakes in remembering keywords, syntax, or typing.

The hypothesis that the graphical interface would produce fewer errors than the textual interface for complex queries was confirmed. Query complexity yielded the only other significant main effect on the error variable; in addition, it participates in all of the other significant interaction effects. Interface type was also involved in two of the four significant interaction effects.

4. CONCLUSIONS FROM THE EXPERIMENTAL DATA

The Graphical Query System resulted in greatly reduced error rates compared to commonly encountered Boolean textual query interfaces for novice users. The reports in the literature that Boolean textual query interfaces are difficult to use for other than the simplest queries were supported in this study.

It is interesting to examine the errors with each interface. Although it is difficult to determine precisely what caused errors with the graphical interface, observation of subjects suggests that these errors were due to items selected (or deselected) unintentionally. This was the result of stray clicks in windows during window manipulations (bringing to front or sliding) or extraneous items selected in the process of forming a query which went unnoticed later.

For the text interface, the errors tallied (i.e., scored as correct or incorrect answers) were never the result of syntax errors (because the subjects were instructed to continue until they corrected any such errors), but were exclusively problems of operator precedence and association (because of badly placed parentheses) , or confusing the meaning of union and intersection operators. There were also a few cases where the use of negation appeared to confuse the issue of which connectives to apply. These results are unsurprising; such problems with textual Boolean query languages have been extensively documented (Braine, 1978; Borgman, 1986; Reisner, 1988).

Minor errors were extremely common with the text interface, as observed in previous studies (Welty, 1985). In rough terms, the errors observed most frequently were misspellings of keywords, followed by parenthesization errors. There were occasional attempts to express range queries in the form “a < b < c,”, where the syntax required “a < b & b < c.”

Graphical interface users may have spent less time formulating queries, but these results are not statistically significant. A larger, perhaps more carefully controlled study, should be conducted to pursue this issue. In any case, the graphical interface was not slower than the textual interface.

5. QUALITATIVE FINDINGS

Progressive refinement of queries was again seen to be an important interface element (Welty and Stemple, 1981; Wong and Kuo, 1982). The effect of this cannot be induced from the data, since this facility was available for use with both interfaces. However, it appeared from observing the subjects that the immediate feedback of the results played an important part in the low error rates obtained with the graphical interface. Essentially, the result window provided a rough "sanity check" for the query; if a user were attempting a query involving, for example, sites in Israel, and "Mission San Juan Capistrano" appeared among the results, the user would know to check for incorrect query terms (in this example, California may have been selected inadvertently). Also, the number of items in the result would provide an indication as to the correctness of the query, if it departed greatly from what was expected. Since the result is updated after every selection or de-selection, it frequently becomes obvious what action caused the problem.

In the graphical interface, progressive refinement is forced on the user: if a change is made to the state of the query, the result changes immediately to reflect it. In the textual interface, the facility is available by merely depressing the "enter" key after constructing a partial query. Subjects took advantage of this infrequently, preferring instead to make large revisions when the query result did not match their expectations. What is interesting is that a sample problem used in training for the text interface involved taking a simple query (e.g., "France") and appending other terms (e.g., "| Italy") to produce more complex ones. A more cautious, piecewise approach might have improved error rates for the text interface.

Overall, comments from subjects having completed both trials favored the graphical version. Subjects seemed to prefer the concreteness of the information display, compared to the relatively abstract view of the information provided by the text version. The graphical system was also said to be more visually engaging, which is likely to have added to its appeal. Although it is a highly subjective impression, subjects also appeared more confident of the correctness of their results with the graphical interface.

In addition, the graphics made indirect queries (e.g., "Find all sites in countries bordering on France") far easier, because these queries require a great deal of contextual information. With graphical displays, users can find this information on screen; with a textual system, users must themselves remember, generate, or guess at such information.

One of the subjects had significant experience with formal logic; although he had a considerably easier time with the Boolean query language than most, it was interesting to note that he made the same types of mistakes in complex queries as did other subjects, and overall, did no better with this interface than with the graphical one.

Most of the subjects had little or no experience with the Macintosh. Observing novices attempt to grapple with the use of a mouse and overlapping windows proved somewhat disconcerting, in light of the purported "user -friendliness" of systems built around this interaction paradigm. It appears that these factors alone may have accounted for much of the time spent locating information. A number of subjects had difficulty maneuvering the mouse cursor, tending to turn the mouse sideways, which makes the movement of the cursor hard to control.

The problem was not one that could be remedied completely by practice with manipulating the interface. A great part of the difficulty came from the need to continually rearrange windows. For example, the world map display (see Figures 2-5) was a relatively large window, compared to the others. A selection from the map would send the map window to the front, covering the smaller windows. Adjusting the selections in these windows in turn would require sliding the map window to uncover the others. This attribute of the Macintosh interface could be overcome with a much larger screen space, thus eliminating the need for window manipulation, or a significant departure from the usual Macintosh interface conventions. The ability to iconify windows might reduce the clutter problem, but at the expense of having to remember more of the query state. The subjects were sometimes confused by these automatic front-to-back window shifts; they seemed to think the windows being obscured had completely disappeared.

6. DIRECTIONS FOR FUTURE RESEARCH

The fact that subjects had some difficulty adjusting to the use of the mouse as a selection device, and, for the graphical interface, spent inordinate amounts of time on the mechanics of the window interface, may account for the absence of significant time differences. A future study might use a large monitor to permit fixing the locations of all windows (which would eliminate the bulk of the mouse manipulations) and provide more training with mouse and window manipulations. This would separate the effects of the interaction devices from the more cognitive aspects of the interface.

Another effect on the time variable was the level of assistance provided for the textual interface. This could be better controlled by providing memorization training of the keywords, and extensive drills in the query language. In addition to the time and error measurements taken in this experiment, it might prove fruitful to follow the subjects' action sequences in answering questions; this might reveal to what extent they were relying on progressive refinement. In addition to the correctness of answers, subjects' confidence in the correctness of their answers would be useful to measure; this confidence will affect users' satisfaction and acceptance of the system.

6.1 Applications

Our graphical techniques have numerous applications in providing information retrieval to novice computer users, particularly in public-access information systems (bibliographic search systems in libraries), and educational systems (online encyclopedias). Graphical query interface can also benefit technically sophisticated users, whose acquisition of computer skills is secondary to the tasks they perform. Geographic and medical information systems are good examples of this category of applications (e.g., traffic control systems or the National Library of Medicine's Grateful Med), since information can be well organized around graphical displays.

In the case of database management systems, construction of a schema is a necessary part of developing a database. The schema can be used to provide the structure to be traversed and examined. For this reason, the GQS is readily adaptable as a query interface for DBMS's. Such a system would be somewhat analogous to QBE, except that selections of data values would be done graphically, rather than by typing selection expressions. Future research could consider mechanisms for specifying joins and aggregate functions (sum, maximum, minimum, etc.).

On the other hand, classical information retrieval systems (that is, systems for retrieving textual documents from a large collection) do not impose a structure on the database. In large-scale information retrieval systems, documents are usually indexed by terms appearing in the documents themselves — the effort of constructing and maintaining controlled vocabularies of index terms by hand is often prohibitive, and controlled vocabulary indexing is not as effective. This means that the index is essentially "flat," consisting of only a pairing of documents and terms. The aggregation/generalization hierarchy and associated indices used by GQS would have to be hand-crafted; thus, it would be difficult to adapt the GQS to work with such a large-scale information retrieval system. It is possible that techniques used to perform automatic document clustering (Salton, 1989) could be adapted to derive the necessary hierarchy from surface features of text documents, but this requires further research.

6.2 Query extensions

The GQS interface supports only union and intersection, and each only for restricted cases. However, the system was conceived initially to support a far greater range of query types. One obvious extension is negation. Since selection (by mouse clicking) is a relatively fast process and the hierarchical organization is designed to limit the number of choices to be made at any level, negation can be accomplished manually to some extent. In the example above, if it were desired to examine any sites except those in the Middle East, it is straightforward to select all of the other geographic regions (there are only nine regions in all). However, if the world map were organized to show each of the sites individually, it would be more cumbersome (even though selection of large areas can be accomplished by dragging the mouse).

It would be possible to add an explicit negation facility by adding a checkbox to the window border that would indicate that the complement of the box's value is to be used, or a menu command that would invert the current selection in the topmost window. One advantage of the latter technique is that it keeps the selected set explicitly visible at all times; if negation is merely a flag, it is easy for users to forget or overlook. As was mentioned before, negation operators in nested expressions containing unions and intersections can obscure their meaning. To illustrate this point, how obvious is it that ~(~x & ~y | ~z) is the same as (x | y) & z?

The way negation is implemented influences the range of queries supported. If negation is provided as an option on all boxes, then, even without removing the restrictions that generalization boxes are disjunctive and aggregation boxes are conjunctive, it is possible *in principle* to express any Boolean query that can be expressed with union, intersection, and complementation — the sequence of selections and negations may become rather tortuous, however. If negation is restricted to the selection boxes (the leaf nodes of the hierarchy), then not all such queries can be expressed. However, the power of the former approach is offset by its potential incomprehensibility.

A more difficult problem is to provide direct disjunction on aggregation nodes (or possibly even conjunction on generalization nodes). One way to accomplish this is to add the ability to open several copies ("clones") of a particular aggregation box. Since the values selected by these aggregation boxes are being passed up the hierarchy to a generalization node, which performs a set union, the effect of this will be to provide the disjunction of selections from those aggregation nodes. This will work if the topmost node or nodes are generalization nodes, which can be forced, if necessary, by supplying a degenerate root node with only one class. It would be possible to do something similar with generalization nodes, to supply conjunction.

The main problem with this scheme, aside from possible screen clutter, is that it would cause a problem with indicating the association of subtrees of the duplicate nodes. If a node is opened twice, different sets of descendants of the duplicate node may be opened; this case may still be relatively clear, because of the highlighting of the names of selected nodes. However, if in this case, the *same* descendants are opened from the duplicate nodes, but *different* selections are to be made from those descendants, it becomes ambiguous which descendent passes its value to which parent. This would require adding some means of associating the boxes visually. One way to do this is to connect them by lines, in effect drawing the hierarchy on the display; the drawback to doing so is that it can severely clutter the display, and that laying out the hierarchy visually becomes an important consideration. Another way would by textual labeling, although this would require careful examination of the node titles by the user, which begins to defeat the purpose of this type of interface.

An intermediate step between the simple interface and the extensions proposed for conjunction and disjunction would be to allow users to explicitly designate the type of a node (AND or OR), but still permit only one copy to appear. The union and intersection boxes could be made visually distinct, but switchable by means of a button. This scheme, however, is less general than the more complex one proposed above, but it has the merit of preserving a simpler interface style. In the "cloning" scheme above, an essentially arbitrary Boolean expression can be composed out of the hierarchy; in a sense, the hierarchy (as it appears on the display) resembles a parse tree for a textual Boolean expression. In the "switching" scheme described here, only one choice of operator would be available per node.

Several other interface features were considered. Since each query box represents the value of a piece of the query subhierarchy, the active window (topmost query box) was connected to the result list in that its value was always the one displayed. This meant that any piece of the query hierarchy could be inspected by making its parent box the active window. The title of the result box would change to reflect the query box whose value was being displayed. We believed that this would assist in "debugging" a query. Unfortunately, it became apparent in pilot studies that users were confused by this feature, partly because the active window is constantly changing (whenever a new selection is made in an inactive window, for example). This feature was inactivated (the result box is now fixed to the root of the hierarchy); it might be useful to restore this feature, however, if window activation could be made independent of selection, or if selection of the box appearing in the result list were made a separate, deliberate act (accomplish by menu selection, for example).

It became apparent during the experiments that screen clutter was a problem (the screen was simply too small). A menu or command-key option to iconify (or just miniaturize) individual windows, without removing them from the query, could be useful to reduce this problem.

Another feature worth considering would be to distinguish the two node types by means of their window frames. Although this implementation used the standard Macintosh window style, these could be customized to provide a stronger visual association between the appearance of a box and the type of the associated node.

6.3 Analysis of Query Capability

The following analysis is intended to assess the range of queries that may be formed given the interface as it was described above, without extensions. This means that aggregation boxes are conjunctive, generalization boxes are disjunctive, and negation is "manual." Clearly, the interface as described does not support relational joins, or the types of aggregate functions (meaning sums, averages, counts, etc.) that most full-fledged relational database query languages support (e.g., SQL). Instead, it is intended to be more like the types of query languages used in information retrieval (e.g., bibliographic) systems, which use union, intersection, and complementation on keyword or full text searches. These bibliographic systems often support special constraints on text searches, like one word occurring before or within n words of another. The latter functionality is not in GQS. However, GQS provides the ability to manage structured information, with greater dynamics (e.g., numeric data that is constantly updated).

As alluded to, the system is not relationally complete, because of its lack of join operators and quantification. GQS directly supports Boolean expressions consisting of the union and intersection of sets selected by discrete choices (types) or ranges of numeric values. It is not capable of expressing every possible Boolean expression involving these operators and terms; the range of expressions that can be generated is limited by the hierarchy used to represent the data. For an aggregation node with n attributes, there can be at most n terms combined by intersection; for a generalization node with m types, there can be at most m terms combined by union. Clearly, this is a limited subset of possible Boolean queries. However, for the generalization nodes, if the restriction on disjoint types is maintained, this is no loss, since intersection on those nodes would yield a null result.

Adding negation at the level of leaf nodes (the selection nodes) may improve the interface, but provides no extension of the query capability; users can achieve the same effect (with more effort) through the standard selection mechanism (they do the negation manually). Adding negation at all nodes, on the other hand, would provide the equivalent of any Boolean expression that could be constructed if any node in the hierarchy could be either disjunctive or conjunctive. This is true by virtue of DeMorgan's Laws, although as mentioned before, it would not provide much in the way of a *usable* increase in query capability, because the cognitive effort involved becomes too great. The same increase in power could be achieved by allowing users to select the operation at each node; in this case, it would also be more usable, since specifying the operation directly is cognitively straightforward.

To provide the full range of Boolean queries possible, a "cloning" scheme like that mentioned above would have to be adopted, so the same term could appear more than once in an expression. The design goals of this system are to provide a useful, intuitive query interface for novice users. Although it is difficult to judge *a priori* what range of query expressions is "useful" enough, it is apparent that fully general Boolean query expressions, whether expressed textually, or using an interface like that of the GQS, represent cognitive overload for most such users. The utility of this interface stems primarily from its use of a hierarchy to control the level of detail, and its concrete representation of the sets which participate in query expressions. The lack of full generality is traded-off for a much simpler, clearer user interface. Inspired by GQS, others have developed a novel graphical representation of complete Boolean queries using a metaphor based on water flowing through a network of filters (Young and Shneiderman, 1993). A second spinoff is dynamic queries which permit adjustment of query components such as sliders and buttons and rapid update of a visual display or text list (Williamson and Shneiderman, 1992).

Acknowledgements: We greatly appreciate the comments on early drafts by Bonnie Dorr, Leo Mark, Catherine Plaisant, and Charles Welty. We are grateful to Ara Kotchian for preparing excellent black and white versions of the color figures. Apple and NCR Corporation provided partial support for this project.

REFERENCES

Bertino, E., and Martino, L., "Object-oriented database management systems: concepts and issues," *IEEE Computer*, vol. 24, 4, pp. 33-47, 1991.

Borgman, C.L., "The user's mental model of an information retrieval system: an experiment on a prototype online catalog," *International Journal of Man-Machine Studies*, vol. 24, pp. 47-64, 1986.

Braine, M.D.S., "On the relationship between natural logic of reasoning and standard logic," *Psychological Review*, vol. 85, pp. 1-21, 1978.

Chen, Z., "Human aspects in object-oriented design: an assessment and a methodology, *Behaviour & Information Technology*, vol. 11, 5, pp. 256-261, 1992.

Czejdo, B., Elmasri, R., Rusinkiewicz, M., and Embley, D., "A graphical data manipulation language for an Extended Entity-Relationship Model", *IEEE Computer,* vol. 23, 3, pp. 26-36, March 1990.

Greene, S.L., S.J. Devlin, P.E. Cannata, and L.M. Gomez, “No IFs, ANDs, or ORs: A study of database querying,” *International Journal of Man-Machine Studies*, vol. 32, pp. 303-326, 1990.

Kim, W., “Object-oriented databases: Definition and research directions”, *IEEE Transactions on Knowledge and Data Engineering*, vol. 2, 3, pp. 327-441, September 1990.

Kim, H. J., Korth, H. F., and Silberschatz, A., “PICASSO: A graphical query language”, *Software: Practice and Experience*, vol. 18, 3, pp. 169-203, 1988.

Lunney, G.H., “Using analysis of variance with a dichotomous dependent variable: An empirical study,” *Journal of Educational Management*, vol. 7, pp. 263-269, 1970.

Michard, A., “A new database query language for non-professional users: Design principles and ergonomic evaluation.,” *Behavioral and Information Technology*, vol. 1, 3, pp. 279-288, July-September 1982.

Norman, D.A., “Some observations on mental models,” in *Mental Models*, ed. D. Gentner, A.L. Stevens, pp. 7-14, Lawrence Erlbaum Associates, Hillsdale, New Jersey, 1983.

Norman, D.A., “Cognitive Engineering,” in *User Centered System Design: New Perspectives on Human-Computer Interaction*, ed. D.A. Norman, S.W. Draper, pp. 31-61, Lawrence Erlbaum Associates, Hillsdale, New Jersey, 1986.

Oddy, R., “Information retrieval through man-machine dialogue”, *Journal of Documentation, 33*, pp. 1-14, 1977.

Plaisant, C., Guide to Opportunities in Volunteer Archaeology: Case study of the use of a hypertext system in a museum exhibit, In Berk, Emily and Devlin, Joseph, (Editor), *Hypertext/Hypermedia Handbook*, McGraw-Hill Publ., New York, NY, pp. 498-505, 1991.

Reisner, P., “Query languages,” in *Handbook of Human-Computer Interaction*, ed. M. Helander, pp. 257-279, Elsevier Science Publishers, Amsterdam, 1988.

Salton, G., *Automatic Text Processing: the Transformation, Analysis, and Retrieval of Information by Computer*, Addison-Wesley, Reading, MA, 1989.

Senko, M. E., DIAM II and FORAL LP: Making pointed queries with light pen, *Proc. IFIP Congress 77*, North Holland Publishers, Amsterdam, The Netherlands, 1977.

Smith, D.C.P. and J.M. Smith, “Conceptual database design,” *Infotech International, Ltd.*, Maidenhead, Berkshire, UK, 1980.

Smith, J.M. and D.C.P. Smith, “Database abstractions: Aggregation and generalization,” *ACM Transactions on Database Systems*, vol. 2, No. 2, pp. 105-133, June 1977.

Welty, C., “Correcting user errors in SQL,” *International Journal of Man-Machine Studies*, *22*, pp. 463-477, 1985.

Welty, C. and D.W. Stemple, “Human factors comparison of a procedural and a nonprocedural query language,” *ACM Transactions on Database Systems*, vol. 6(4), pp. 626-649, December 1981.

Williams, M., What makes RABBIT run?, *International Journal of Man-Machine Studies*, *21*, pp. 333-352, 1984.

Williamson, C. and Shneiderman, B., The Dynamic HomeFinder: Evaluating dynamic queries in a real-estate information exploration system, *Proc. ACM SIGIR Conference*, pp. 339-346, 1992.

Wong, H.K.T. and I. Kuo, "GUIDE: Graphical user interface for database exploration," *Proceedings of the 8th VLDB Conference*, 1982.

Young, D. and Shneiderman, B., A graphical filter/flow model for boolean queries: An implementation and experiment, *Journal of the American Society for Information Science*, 1993.

Zloof, M. Query-by-Example, *Proc. National Computer Conference, 1975*, AFIPS Press, Montvale, NJ.