Developing macro queries:
A comparison of text-based and visual approaches

Craig Standing
Department of Information Systems
Edith Cowan University
c.standing@cowan.edu.au

The paper examines the problems associated with using and teaching Decision Support System interfaces. In particular, it examines the specialised and often overlooked task of developing macro queries in software systems. It compares two common interface types for developing macros, a text-based command interface and a visual query language for retrieving information from a Geographical Information System. Although it is often automatically assumed that graphical interaction methods are superior to text-based command languages, the literature is divided on the issue.

Introduction

Decision support systems (DSS) are a varied group of software systems that aid users in making decisions of a complex nature. How the users interact with these systems is just as varied as the applications themselves. Interfaces to DSS range from the grid like spreadsheet, menus, command languages, to graphical or iconic systems. For an overview of interface styles see Hawryszkiewycz (1997). University programmes frequently use various forms of DSS as part of their courses. Graphical user interfaces are commonly thought of as being the most suitable and most effective interface style for these DSS, having superseded the text-based command languages within many teaching programs. Interestingly, results of evaluations in the literature of the superior effectiveness of graphical query languages over text-based command languages are far from being clear cut in favour of graphical query languages.

A related and yet often overlooked problem is that of designing software that aids the user in developing small programs of commands or macros. Database systems, spreadsheets, statistical packages, and geographical information systems for example all include macro languages that can be used to automate tasks and allow the creation of more sophisticated applications. Students often hit a wall in their learning of sophisticated software packages as few progress to what is perhaps considered the advanced skill of macro development. As a result, students may feel to be stuck at the beginner or intermediate level of user skill in a particular package and feel frustrated by the obstacles provided by macro development which has much in common with programming in the traditional sense. Teachers designing courses that involve complex software with macro languages may be worried by the text-based language that they are faced with teaching.

This paper examines the literature on graphical query languages and text-based command languages to determine if there is conclusive evidence that one is better than the other in terms of ease to learn and ease of use. Although the evaluation that is the focus of this paper is primarily concerned with macro query development; a discussion of systems that allow the execution of single operations (rather than macros) is still relevant. Two contrasting interface styles to Geographical Information Systems (GIS), a text-based command language and graphical query language, both used for macro development, are then presented. The results from evaluations of user performance on the two systems for macro query development are presented and discussed. Finally, the implications for educators and students are examined.

A review of command languages and GUIs

Graphical query languages appear to be the dominant method of interacting with database systems at present. It is assumed that graphical interfaces have a number of significant advantages over command languages for database interrogation. However, these advantages are rarely clearly articulated. Indeed, a number of studies that have compared graphical query languages with other interaction styles have not confirmed that users perform more favourably on graphical query languages. It is proposed here that
the issue has been oversimplified and that the topic therefore is still worthy of further investigation. Firstly, we examine the strengths and weaknesses of command languages and then examine the design features that make graphical languages effective.

**Command languages**

Command languages often have the advantage of being flexible as the user is free to string various commands together to interrogate a database. They can also be powerful in that for a short command with the relevant parameters, input can be minimised and so the query can be specified quickly. The command options can provide a number of shortcuts in retrieving information.

Some of the drawbacks of this dialogue strategy are:

- users have to memorise commands;
- they are syntactically fussy;
- developing expertise in macro development is often difficult;
- they tend to be procedural languages and so impose a strict order in query development.

For these reasons they are criticised as a dialogue type for beginners and less frequent users who often forget the commands (Burton & Steward, 1993; Kerry, 1990; Preece, 1994). Macro languages are frequently text-based. The development of macros involves expressing a sequence of commands, often with the objective of reusing the macro. Because of the tendency of macros to be more complex than expressing one or two individual commands the skills has been compared to computer programming (Nardi, 1993). In the section following Graphical Query Languages the results of experiments are reported that compared user performance on command languages and graphical languages.

**Graphical query languages (GUIs)**

This section reviews the literature and examines ‘what makes graphical interfaces effective?’ in order to understand the real contribution of graphical representations in the design of interfaces. The following discussion focuses on visual querying and visual programming languages.

Visual programming represents a departure from tradition and according to Shu (1988) is driven by the following premises:

- Pictures are more powerful than words as a method of communicating as they can often concisely convey more meaning.
- Pictures can help comprehension and remembering.
- Pictures may provide an encouragement for learning to program.
- Pictures, if designed properly, can work across cultures and languages.

Although these are common sense points they don’t offer us much help in designing graphical interfaces. The statements by Shu don’t inform us how pictures can be used to concisely convey more meaning. Also some would argue that some icon designs are instantly forgettable.

Lodding (1983) described the differences in the way that text and images are processed by humans; text being read in a sequential manner while images are processed in a parallel way. Iconic interfaces have the advantage of requiring a user to recognise and point rather than, as with command languages, remember and type (Smith et al., 1982). Larkin and Simon (1987, p.98) put forward the following reasons why a diagram can be superior to a verbal description for solving problems:

- Diagrams can group together information that is used together, and so cut down on the search time for information.
- Diagrams use location to group information.
- Diagrams automatically support a large number of perceptual inferences.

In other words, differences and similarities between objects can be conveyed in the icon design. For example, text documents could have a similar shape and colour but vary in size according to the size of the file in bytes. Lodding (1983) assumed that graphical languages are “automatically” superior to text-based languages. This is not correct since graphical systems require detailed design decisions to be made to achieve these perceptual differences. They are not an automatic by-product of choosing a graphical interface.

Marian Petre (1995) examined the significant features of graphical notations, in electronic circuit diagrams. The work generalises to graphical interfaces where information has to be presented precisely. She put forward the arguments that:

- Much of what contributes to the comprehension of a graphical language is due to a secondary notation of layout, typographical cues, graphical enhancements and use of white space. The presence of graphics in a notation does not guarantee clarity. In the experiments by Petre (1995) with novices and experts in electronic circuit design differences were found in way the two groups interpreted graphics and text in electronic schematics. It was found that the secondary notation in the diagrams were most important to aiding understanding of the diagram. This notation included: elements of adjacency, clustering, white space,
views that the secondary notation features any other mode of interaction. As time and error rates were concerned, which is superior. Catarci and Santucci comparisons of Structured Query Language (SQL) (1981) and other dialogue methods Scammell (1993) compared SQL and language developed by the authors, outperformed SQL in a graphical language and information in a graphical language and skills in interpreting a graphical language need to be developed, just as in any other mode of interaction.

**Comparisons**

From studies that have involved comparisons of Structured Query Language (SQL) (1981) and other dialogue methods the results are inconclusive in relation to which is superior. Catarci and Santucci (1995) found that QBD, a graphical language developed by the authors, outperformed SQL for most types of queries as far as time and error rates were concerned. However other studies, such as Yen and Scammell (1993) compared SQL and Query-By-Example (Zloof, 1980) and found that query language type had no effect on user performance in terms of correctness of queries. Roy (1992) found no significant difference in the performance improvement of users on menu systems and command languages for CAD applications. Davis (1989) in a comparison of first-time users of SQL and menus found that the SQL users' performed better than menu users' and generally considered it easier to use.

Although these studies have compared text-based command languages and graphical query languages there has been no work as far as the author is aware on the comparison of the two methods in relation to the development of macro queries. The term macro query is used here to denote the development of a query that involves a number of steps, either in commands or in graphical manipulations, to solve a problem. This macro solution could either have variables included or be modified to form a reusable query. Developing macros or small reusable sequences of commands is often seen as an advanced skill.

**GIS interfaces used in the study**

This research used two different interfaces to Geographical Information Systems as part of the study on the strengths and weaknesses of text-based and graphical macro languages. Geographical information systems are computer-assisted tools used to collect, analyse, manage and present spatially defined data. There are a wide variety of software systems available which facilitate these activities. The major differences between them are how they represent the data i.e. whether the GIS is a vector or raster system; and the type of operations they perform. Businesses, universities and government departments are increasingly using GIS across a diverse range of applications from land management to marketing and demographics. In some respects, the use of GIS is typical of many decision support systems in that many learn the basics of the software but few go on to be competent users (not necessarily experts).

There are a number of software components in a geographical information system. Not all systems have all of these components but to be considered a true GIS then there must be a core set of components consisting of:

- a spatial and attribute database;
- a cartographic display system;
- a map digitising system;
- a database management system;
- a geographic analysis system (Eastman, 1992).

In addition to these core elements, a GIS may provide components to analyse remotely sensed images and provide statistical analyses.

Traditionally, most GIS relied upon command-language interfaces, the commands being strung together to form macros by the more experienced user (Egenhofer & Herring, 1995). More recently, graphical user interfaces have been developed because of the problems faced by beginners and those that use the tool infrequently. When a user wishes to progress beyond the first user level stage, which may either be a simple graphical interface, menu system or even a subset of the same command language used by the expert, it is not difficult to appreciate the difficulties that the inexperienced user will get into (Burton & Steward, 1993; Ng, 1993). The skills required to become an efficient user of many GIS have some similarities to those required for writing small programs in languages that are part of spreadsheets, databases, CAD, and statistical packages (Nardi, 1993).

**Evaluation method**

This study attempted to compare the approach of developing queries in VFQL with developing macros in a command language. It assessed the usability of the interaction method in terms of error rates and time taken to complete tasks. The participants were grounded in GIS concepts and had acquired some basic skills on a GIS but were looking to move to a more sophisticated level of usage.

Eighteen final year undergraduates in environmental science from Edith Cowan University who had taken a unit on Spatial Information Systems were the subjects for the experiment. They had all taken a semester long course that included studying GIS on a conceptual and practical level using a GIS called Idrisi (Eastman, 1992) but they had not used ARC/INFO.
(ESRI, 1994). They had not developed macros in the command language but were familiar with executing a series of single commands at a system prompt. They had experience in both a practical and theoretical form with GIS. The subjects were divided into two balanced groups of nine according to their grade for the GIS unit they had taken. Both groups contained equal numbers of participants with A, B and C grades achieved in their GIS unit.

There are two main ways of conducting experiments: within group and between group. The within group method takes the subjects and uses them on both or all systems, sometimes forming into two groups and swapping over. The major criticism of this method is that it should not be used where there is some transfer of learning or knowledge from the first system experiment to subsequent system experiments. Between group experiments split the numbers into groups and each group is used with just one system and the results compared.

For the evaluation of VFQL and the command language a between group design was chosen. The rational for this being that if a within group design was chosen participants would learn from their first attempts with the query language and this would influence the second set of results. In this method there may be differential transfer of learning from one system to another. From the sample size of eighteen, nine were selected to work on VFQL and nine on the ARC/INFO macro command language using a basic text editor.

The participants were asked to develop queries for seven tasks which were classified from simple (T1, T2, T3), intermediate (T4) and complex (T5, T6, T7). They were timed and their solutions checked for errors. For each task the participants were given three attempts. Each attempt was checked for syntax and logical errors. The answers were graded in the following way based on the method used by Yen and Scamell (1993):

1 = minor data error or minor language error;
2 = arguments to a function wrong way around, more serious language error;
3 = wrong use of operation, or wrong operation to produce result, or missed operation.

Each attempt was given the score of its most severe error as an item score and repetitive errors were counted each time they occurred (each attempt), this is also in line with the study by Yen and Scamell (1993). The total error rate and total time was recorded at the end of the third attempt or whenever the query was successfully completed. For example, a participant could complete the task successfully at the first, second or third attempt, or could still have errors in the query after the third attempt. The feedback time (time between attempts) was not included in the data.

**Table 1: Mean number of errors and time to complete tasks**

<table>
<thead>
<tr>
<th>Task</th>
<th>VFQL Mean Errors (over 3 attempts max)</th>
<th>Command Language Mean Errors (over 3 attempts max)</th>
<th>VFQL Mean Time (s)</th>
<th>Command Language Mean Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>0.9</td>
<td>5.7</td>
<td>259</td>
<td>722</td>
</tr>
<tr>
<td>T2</td>
<td>0.1</td>
<td>2.4</td>
<td>137</td>
<td>295</td>
</tr>
<tr>
<td>T3</td>
<td>0.7</td>
<td>3.0</td>
<td>247</td>
<td>274</td>
</tr>
<tr>
<td>T4</td>
<td>2.3</td>
<td>3.9</td>
<td>531</td>
<td>677</td>
</tr>
<tr>
<td>T5</td>
<td>2.9</td>
<td>3.6</td>
<td>454</td>
<td>664</td>
</tr>
<tr>
<td>T6</td>
<td>4.6</td>
<td>2.6</td>
<td>784</td>
<td>922</td>
</tr>
<tr>
<td>T7</td>
<td>3.4</td>
<td>4.6</td>
<td>560</td>
<td>594</td>
</tr>
</tbody>
</table>

**Evaluation results**

In the experiment, users of VFQL made fewer errors and spent less time on developing solutions to simpler tasks. On more complex tasks users again made fewer errors and took less time than in the command language. However the margin of difference was less in percentage terms than for simpler tasks and the difference was not statistically significant in multivariate ANOVA tests.

From Table 1, it can be seen that users made fewer errors and completed tasks more quickly on VFQL than the users of the command language. The difference between the users on the earlier tasks is particularly marked.

Tasks 1, 2, and 3 were classed as simple tasks and tasks 5, 6, 7 as complex tasks. Task 4 was judged for the purposes of this study as an intermediate level task. When a comparison of classes of tasks is made (Table 2) it can be seen that the error rate was much greater in the command language than VFQL for simple tasks, six times greater in fact. The difference between the two systems for simple task error results was significant (p < .05) according to a multivariate ANOVA (or MANOVA) test (see Table 2). For complex tasks the error rate for the command language answers was about 50% greater, although this was not statistically significant due to internal group variation.
The answers were completed for VFQL more quickly than in the text-based command language. The difference between the two systems for the time taken for complex tasks was less in percentage terms than for the difference between the two systems for the simple tasks, however, this difference was not statistically significant (see Table 3). The difference between the two systems for the simple tasks was statistically significant for $p < 0.05$.

The results of the experiment that compared user performance in VFQL and the text-based command language can be summarised as follows. For simpler tasks, VFQL significantly outperformed the command language in terms of error rates and time taken to complete tasks. For more complex tasks the difference between the two systems was not statistically significant. As far as the time factor is concerned, the reason for this result may be that the intention (planning) and evaluation (assessing) periods (Norman, 1985) in the process of developing a complex solution are likely to be high in relation to the selection and execution of the tasks in the software. As for the error rates on complex tasks, again neither system had an edge on the other. In other words neither one of the systems was substantially better than the other in reducing the time and error rates on the complex tasks.

### Implications for education

The study revealed a number of implications for teachers and students in the educational sector and these are now discussed.

Teachers and students often assume that graphical interfaces to database systems are easier to use than text-based command languages. This is only partially true based on the results of this study. For simpler queries, graphical query languages seem to be more effective for users. The important point here is that the graphical query language must be well designed. It should take into account the secondary notation features mentioned earlier in the paper as well as offering features that are not available in an equivalent command language.

In the case of VFQL, it has no number of useful features that serve as an aid to users in constructing queries. It provides information to the user about the data sets and the operations and so greatly reduces the number of errors when building queries that are related to syntax and data types. It also provides a flexible method of building queries since users are not confined to working in a strictly procedural manner. The low-level operations can be replaced by higher order functions more suited to the solution of problems in the problem domain.

For more complex queries the interface strategy may not be as important in the overall solution of the problem as people think. The study revealed no significant difference between the two systems for complex tasks. We might think the graphical interface is helping us solve the complex problem but it is probably no more significant than a comparable text-based command language. In this respect the teachers and students must always be aware that solving the underlying logical problems is something that should be concentrated upon rather than focusing upon the software tool. The package interface will provide significant help in routine task solutions but overall is a less important in the solutions to complex problems. Teachers therefore should not be put off from designing courses that include the learning of text-based macro languages.

Even though software companies market many of their products as intuitive and easy to learn, in reality students and teachers must not underestimate the fact that graphical query languages require a significant amount of learning and effort to master and appropriate laboratory sessions should be organised. There should also be emphasis placed on the mapping of the logical solution to the sequence of constructs needed within the software.

Graphical query languages are often non-procedural in nature. As a result they can be flexible in terms of the order the user chooses to develop the components.

### Table 2: Mean errors for simple and complex tasks

<table>
<thead>
<tr>
<th></th>
<th>VFQL Errors</th>
<th>Command Language Errors</th>
<th>Multivariate Anova F Value</th>
<th>D.F.</th>
<th>Significance of F</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simple Tasks</td>
<td>0.6</td>
<td>3.7</td>
<td>5.16</td>
<td>(3,14)</td>
<td>0.02</td>
</tr>
<tr>
<td>Complex Tasks</td>
<td>3.6</td>
<td>5.3</td>
<td>0.78</td>
<td>(3,14)</td>
<td>ns</td>
</tr>
<tr>
<td>All Tasks</td>
<td>2.1</td>
<td>4.5</td>
<td>1.81</td>
<td>(7,10)</td>
<td>ns</td>
</tr>
</tbody>
</table>

### Table 3: Mean time in seconds to complete simple and complex tasks

<table>
<thead>
<tr>
<th></th>
<th>VFQL Errors</th>
<th>Command Language Errors</th>
<th>Multivariate Anova F Value</th>
<th>D.F.</th>
<th>Significance of F</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simple Tasks</td>
<td>0.6</td>
<td>3.7</td>
<td>5.16</td>
<td>(3,14)</td>
<td>0.02</td>
</tr>
<tr>
<td>Complex Tasks</td>
<td>3.6</td>
<td>5.3</td>
<td>0.78</td>
<td>(3,14)</td>
<td>ns</td>
</tr>
<tr>
<td>All Tasks</td>
<td>2.1</td>
<td>4.5</td>
<td>1.81</td>
<td>(7,10)</td>
<td>ns</td>
</tr>
</tbody>
</table>
of the solution. Whilst this is generally seen as advantageous to users and especially those learning the software, it can create problems. On complex problems non-procedural languages may not provide the user with a problem-solving sequence. Sequential command languages are generally used, as in this study, to create solutions step by step in a logical sequence. If this is not necessary because of the tool allowing greater flexibility then the user can start at any point in the problem, and even work backwards on the problem. This may be advantageous to some but may mean that others get into all sorts of difficulty because they lose their way in their problem-solving strategy.

This paper has attempted to clarify the issues related to interface styles in developing macro queries to database systems. The issues are not as clear cut as is often assumed. Results from the study indicate that graphical query languages can be more effective tools, but only if well designed. Also the interface strategy is less significant as the problems increase in complexity. Whilst the author does not recommend that students and teachers rush back to using command language interfaces, they should be aware that for complex problems the interface is probably only of limited significance.

### Table 4: ANOVA test for each task for errors and time

<table>
<thead>
<tr>
<th>Task</th>
<th>F(1,16) for errors</th>
<th>Value of p</th>
<th>F(1,16) for Time</th>
<th>Value of p</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>15.94</td>
<td>0.002</td>
<td>22.05</td>
<td>0.001</td>
</tr>
<tr>
<td>T2</td>
<td>7.18</td>
<td>0.020</td>
<td>3.77</td>
<td>0.076</td>
</tr>
<tr>
<td>T3</td>
<td>4.44</td>
<td>0.057</td>
<td>0.31</td>
<td>ns</td>
</tr>
<tr>
<td>T4</td>
<td>0.53</td>
<td>ns</td>
<td>1.20</td>
<td>ns</td>
</tr>
<tr>
<td>T5</td>
<td>1.71</td>
<td>ns</td>
<td>2.17</td>
<td>ns</td>
</tr>
<tr>
<td>T6</td>
<td>2.11</td>
<td>ns</td>
<td>1.04</td>
<td>ns</td>
</tr>
<tr>
<td>T7</td>
<td>0.49</td>
<td>ns</td>
<td>0.07</td>
<td>ns</td>
</tr>
</tbody>
</table>

### REFERENCES


