MODELLER : A DSS FOR EXPERIMENTAL STUDY OF HUMAN-COMPUTER INTERACTION

G. R. Finnie
University of Natal,
Pietermaritzburg, 3200

This paper describes the structure of a simple nonprocedural financial modelling system designed to collect a variety of statistics on aspects of novice interaction with decision support systems. Some findings on the use of HELP facilities by novices are discussed. Certain areas of difficulty experienced by novices are identified from an analysis of compilation and logical errors.

1. INTRODUCTION

Decision support systems (DSS) can play a significant role in the management of organisations, with the term covering a wide range of computer-based management and staff support tools. There is, however, some disagreement on whether the area is sufficiently novel or distinct to justify a new title. Naylor [14], for example, argues that the basic techniques of DSS have existed in the management information systems (MIS) area for a considerable time. In contrast, Bonczek et al [3:62] argue that, although both DSS and MIS rely on mechanisms for handling data, "... they differ in terms of the purpose for which data are used. For an MIS application, data are typically used in the context of repetitive, routine transactions and report generations. In contrast, a DSS uses data in responding to ad hoc, exploratory questions of a user who factors the responses into a decision making process".

A major focus in the DSS area has been the issue of "user-friendliness" and the structure of the human-computer interface. Many DSS incorporate features such as nonprocedural language to facilitate the rapid development of solutions. The active role of the DSS user suggested by Bonczek et al has to some extent introduced a new class of novice or occasional computer user as well as a new class of languages.

Financial modelling systems are possibly the most popular DSS in use at present. More precisely, these systems are in the class of DSS generators (Sprague & Carlson [19]) as they are used to generate specific models of business conditions. Although extensively used in a wide variety of organisations, little research has been done on what problems can arise in their use, how fourth generation features are used in practice and what factors can inhibit their acceptance.

MODELLER was developed primarily as a vehicle for the study of aspects of the human-computer interface in the use of financial modelling systems and of DSS in general. The system collects a variety of statistics as well as taking copies of models at various stages of development. To date it has been applied to the study of the use of nonprocedural sequencing and to the identification of problemm areas in the use of financial modelling. Although the study of novice programmers is an area that has received considerable research attention, little work appears to have as yet been done in the DSS area.

2. SOME STUDIES OF PROGRAMMING

A number of approaches have been applied to the study of programming and language constructs. Microlanguages which allow concentration on specific aspects of language have been used to compare language features [1, 16, 17] as have natural English commands [12]. Pinsky [15] used video cameras to study observable behaviors in analysing the actions of operators using a first system, as well as analysing the verbal protocols used. Most studies have, however, dealt with specific languages, both with a view to identifying problem areas or in using the language to investigate related concepts. Languages studied have included PASCAL [18], COBOL [20], SOLO [6, 11], BASIC [2], FORTRAN [4], and LOGO [5].

One technique which has been applied to identify problem areas in languages has been the analysis of errors. Moulton & Muller [13] used the DITRAN compiler to tally data on statement usage and error distribution in FORTRAN. Youngs [21] established a classification of general

programming constructs and evaluated the range of errors in each, for both novices and experts, to compare five commonly used languages. Gannon [8] used a measure of error severity based on the persistence of errors to compare TOPPS and TOPPS II. Sime et al [16, 17] investigated conditional constructs in microlanguages, with semantic error counts being one of the variables of interest. Little work appears to have been done recently, particularly from the approach of utilising the compiler to collect error statistics. One factor certainly is the cost and complexity of doing this work on full-scale programming languages but an additional consideration is that a compiler can only detect certain classes of error, whereas many important areas of difficulty in programming arise with logical errors.

An approach which appears to have potential for identifying problem areas in novice use of languages lies in monitoring the use of HELP facilities during program development. No references were found to any research being done here, possibly because computing facilities for novice student users have often been limited to batch access. In addition most studies have concentrated on "conventional" languages such as Pascal which generally do not incorporate HELP subsystems. Many of the so-called "fourth-generation" systems do however make extensive use of the HELP concept. Measuring the distribution of requests for HELP in such systems could provide some insight into those areas in which users experience difficulty.

3. THE STRUCTURE OF MODELLER

3.1 The Modeller Language

Financial models are essentially arrays of data used to represent a specific business scenario. The rows in the model define individual variables and the columns generally indicate time periods, although they may be used for other purposes, e.g. totals over a set of time periods. An example of a simple financial model (in MODELLER) with its associated data matrix is given in figure 1.

```
10      SALES = 55000, 50000, 60000, 40000 FOR 2
20      CASH_SALES_PCT = 0.6
30      CREDIT_PCT = 1 - CASH_SALES_PCT
40      COST_OF_SALES_PCT = 0.55
50      EXPENSES = 18000
60      CASH_IN = 0, CASH_SALES_PCT * SALES + CREDIT_PCT * LAST_SALES
70      CASH_OUT = 0, COST_OF_SALES_PCT * NEXT_SALES
80      CASH_ON_HAND = 10000, LAST CASH_ON_HAND + CASH_IN-CASH_OUT

SALES              55000.00  50000.00  60000.00  40000.00  40000.00
CASH_SALES_PCT         0.60      0.60      0.60      0.60      0.60
CREDIT_PCT             0.40      0.40      0.40      0.40      0.40
COST_OF_SALES_PCT      0.55      0.55      0.55      0.55      0.55
EXPENSES           18000.00  18000.00  18000.00  18000.00  18000.00
CASH_IN                0.00  52000.00  56000.00  48000.00  40000.00
CASH_OUT               0.00  33000.00  22000.00  22000.00      0.00
CASH ON HAND       10000.00  29000.00  63000.00  89000.00 129000.00
```

Figure 1 : Financial Model with Data Matrix

MODELLER was written in HP3000 Pascal and is a self-contained system, consisting of a compiler, interpreter, editor and simple filing system. The language for model definition is similar to that of the modelling subsystem of IFPS[10]. Models are restricted to a maximum of 70 rows and 10 columns, which is generally adequate for teaching purposes but can be extended if required. The language has facilities for expression repetition in columns (the FOR keyword), references to preceding or succeeding columns (the keywords LAST/NEXT) and the conditional assignment of expressions to variables (the keyword IF). Functions may be used as operands. In addition, column computation can be used, i.e. a reversal of the normal sequence of computation (within columns, down rows) to one in which the arithmetic is performed across the columns

(i.e., within rows, across columns).

A number of commands are available to control system operation. These include commands to evaluate models, request help, obtain hardcopy results, manipulate models to investigate different business scenarios and terminate the session. Editing commands can be used to insert new lines, modify existing lines or delete lines no longer required.

The filing system is intended to completely hide any system complexity from the user. Only one model may be used during a MODELLER session and this model is kept and retrieved from the disc filing system via the keywords SAVE and GET, with no model name required as a parameter. The restriction to one model is not a serious limitation in a teaching environment where students are generally working on a specific case study and it considerably simplifies the tracing of model development.

3.2 The Compiler and Interpreter

The compiler subsystem of MODELLER compiles each line of the model in line number order, producing output code in the form of Reverse Polish (RPN) strings. Since the language is nonprocedural, variables may be referenced in advance of their definition, e.g.

```
10    x = y + 20
.
.
.
50    y = 50
```

In order to fully evaluate x, values must have been established for y, i.e. the value of x depends on y and an evaluation sequence must be determined. A method based on the topological sorting of directed acyclic graphs was devised to extract the correct sequence [7].

The compiler operates in a manner similar to a two-pass assembler to detect valid forward references to variable names. The first pass extracts variable names, i.e. anything to the left of the assignment symbol. These are hashed to a symbol table to establish a name/matrix row number correspondence. Long lines (i.e. those that contain a continuation marker) must be detected in order to maintain correct numbering of the matrix rows. The second pass compiles the rest of each line, using the definitions established during the first pass.

The syntax analysis phase of the compiler is based on the use of transition tables (transition matrices). This approach relies on the concept of a current state of compilation which then defines the action to be taken when the next symbol is detected. The symbol and the current state then define the next state to be entered (see e.g. [9]). The technique has largely fallen into disuse in compilers for conventional programming languages due to the size of the transition matrix required for the analysis of a complete language. However, in this case the limited command set in the language means that the transition matrix remains fairly small. The technique is especially suitable for this type of study due to the precision of error detection and recovery allowed. Any error exit from the matrix can indicate the error type, the state of occurrence and the symbol type causing the error. In addition the technique is very fast (since no searching is required), which makes it suitable for interactive use. A major transition table was used for syntax analysis while smaller tables were used for token identification (lexical analysis) and for the identification of functions.

Once compilation is complete and an evaluation sequence established, control passes to the interpreter module. Each entry in the data matrix is evaluated using a stack for computation of Reverse Polish strings. Any run-time errors, e.g. division by zero, will halt the evaluation with a suitable error message.

4. DATA COLLECTION

MODELLER traces model development and collects statistics by creating a separate trace file for each user. The file is locked to prevent illegal access and is invaluable as a control mechanism in a student user enviroment.

The following data is collected :

(a) Date and time stamps for the beginning and end of each terminal session.

(b) Copies of all editing operations performed on the model.

(c) A copy of the initial model.

(d) A copy of the model at the termination of each terminal session, providing that the model has been permanently changed during the session.

(e) A record of all compilation errors at each evaluation of the model and a record of the use of HELP facilities prior to this evaluation. These are packed into a form allowing the error class (or HELP class) and the number of errors (or HELP requests) to be held in one 16-bit word of storage. Only non-zero error/HELP entries are retained.

(f) Data on the variable definition and reference structure in the nonprocedural model at each evaluation. This includes the number of forward references, the longest forward reference, average forward reference length, longest backward reference and average backward reference length.

(g) The time since the start of the session for each evaluation.

(h) A record of all model manipulation (the use of WHATIF statements).

The trace files for all participants in a specific experiment are consolidated into a single large file using the system editing facilities. A separate program has been written to process this file to extract and highlight individual results for error rates, use of HELP facilities, timing and sequencing data. A consolidated table is produced of the frequency distribution of error rates and HELP facility requests for the entire sample.

5. SOME EXPERIMENTAL RESULTS

5.1 Experimental Conditions

Two experiments have been performed using MODELLER, both of which required undergraduate commerce students to complete case studies involving financial models. Although the analysis of error rates and HELP requests provide some insights into the use of financial models by novices, several more experiments are planned to obtain a more comprehensive picture.

The first experiment involved 132 final year students, 85 of whom had some prior computing experience from having attended at least one university level computing course. The second experiment used a group of 142 second year students, only 17 of whom had prior experience in computing. Participants were provided with a copy of the MODELLER User Manual and were given three 45 minute lectures on financial modelling.

A major problem in student experiments of this type (and in teaching computing in general) lies in control of the subjects, i.e. in limiting the copying of models and transfer of solutions. To separate genuine participants in the exercise from the rest, a careful study was made of each subject's trace file to detect cases of copying or excessive assistance. This reduced the sample to 68 in the first experiment and 92 in the second.

5.2 Use of HELP Facilities

Usage of HELP facilities was monitored by recording the number of requests for explanation of model structure or MODELLER commands. It was felt that knowledge of the frequency and distribution of these requests for assistance would aid the identification of problem

areas in using the language.

Unfortunately, few subjects in fact attempted the HELP option, i.e. six out of 68 in the first experiment, and 21 of 92 subjects in the second.

98 requests were made for explanation. Of these, 33% were for editing commands, 22% were for filing commands and 30% for general system commands (Exit, Evaluate, Logging and Whatif). Only 15% were for explanation of the modelling language itself, the largest group (6%) being requests to explain the structure of variable names.

It is apparent, at least in these experiments, that monitoring of HELP usage provides little information of direct value for the identification of problem areas. Novices appear to have some difficulty in the editing of models but the small sample makes generalisation suspect. The apparent reluctance of novices to try this facility, coupled with the few requests for information about language structure, indicates that such users prefer to work from manuals in most cases. A popular view in the DSS field is that a system should be (virtually) capable of teaching itself. Although the quality of services available would obviously influence the general usage of these facilities, the small number of users who attempted access in these experiments indicates that conventional manuals and teaching methods remain of importance in the training of novices.

5.3 Analysis of Errors

The MODELLER system gathers error counts for each individual for each of the compilation error types. A separate program accumulates this data to generate an error frequency distribution. The most significant areas of difficulty in the use of financial models identified by the compilation errors relate to understanding the use of functions and, particularly for total novices, understanding of the underlying matrix structure. Knowledge of other language conventions which are incompatible with the modelling system form a fairly significant source of errors amongst the more experienced subjects but it is unlikely that these would persist.The collection of logical errors is no easy task as these are, of course, not detected during compilation. A sample of logical errors was extracted for each experiment by examination of the final models (i.e. for incorrect results) and careful examination of the trace files. In the latter case, the editing commands were studied to observe those situations where modifications were made to models which had correctly compiled. Logical errors were grouped into three broad classes, viz. arithmetic errors, language related errors and data structure (matrix) related errors.

The most common errors in the arithmetic class were incorrect use of the order of evaluation and an incorrect use of fractions as percentages. In the language related class the most difficulty was encountered in the use of the conditional statement, either relating to incorrect use or reversal of relational operators or misunderstanding of the effect. Another common error involved either including extraneous variables or leaving variables out of the range of the SUM function. Errors classified as data structure related were those which arose due to the subject's lack of understanding of the underlying data matrix. Common errors were in incorrect references to preceding/succeeding columns, misunderstanding of the cumulative effect of these references (i.e. implicit iteration) and unneccessary item repetition in statements.

6. CONCLUSION

As a general research technique, the use of self-contained systems for the experimental study of aspects of human-computer interaction appears to have considerable potential. By restricting the necessity to use "external" facilities, virtually all aspects of direct computer use can be monitored. Financial modelling systems, being relatively easy to use and based on a simple data structure, provide a good basis for such work. The technique also allows for comparative research on language constructs, e.g. different forms of conditional expressions could be used in separate experiments with the comparative error rates providing a measure of relative difficulty.

MODELLER has been used to investigate both the use of nonprocedural sequencing of variables and areas of difficulty in the use of nonprocedural financial modelling systems by novices, with the latter being briefly discussed in this paper. From these results, it is apparent that novices face a considerable conceptual barrier in understanding the role and use of the underlying data matrix. This structure is an integral part of all financial models and difficulty with

this concept could prove to be a significant inhibiting factor in their adoption. Training programs for novices should concentrate on bridging this gap at an early stage in the training process.

REFERENCES

1. Arblaster, A. (1982). Human Factors in the Design and Use of Computing Languages. *Int. J. Man-Machine Studies* 17, 211-224.
2. Bayman, P. and Mayer, R.E. (1983). A Diagnosis of Beginning Programmers Misconceptions of BASIC Programming Statements. *Comm. ACM.* 26, 677-679.
3. Bonczek, R.H., Holsapple, C.W. and Whinston, A.B. (1981). Foundations of Decision Support Systems. Academic Press, New York.
4. Coombs, M.J., Gibson, R. and Alty, J.L. (1981). Acquiring a first Computer Language : a study of Individual Differences. In M.J. Coombs and J.L. Alty (Eds.) ComputingSkills and the User Interface. Academic Press, London.
5. du Boulay, J.B., O'Shea, T . and Monk, J. (1981) . The Black Box inside the Glass Box : Presenting computing concepts to Novices. *Int. J. Man-Machine Studies* 14, 237-249.
6. Eisenstadt, M. (1983). A User-friendly Software Environment for the Novice Programmer. *Comm. ACM,* 26, 1058-1064.
7. Finnie, G.R. (1985). Sequencing Nonprocedural Financial Models. *Decision Support Systems* , 1, 247-250.
8. Gannon, J.D. (1976). An Experiment for the Evaluation of Language Features. *Int. J. Man-Machine Studies,* 8, 61-73.
9. Gries, D. (1971). Compiler Construction for Digital Computers. John Wiley & Sons, New York.
10. IFPS User's Manual (1984). Execucom Systems Corporation, Austin, Texas.
11. Kahney, H. (1983). Problem Solving by Novice Programmers in T. R. Green, S. J. Payne and G.C. van der Veer (Eds.). The Psychology of Computer Use. Academic Press, London.
12. Miller, L.A. (1974). Programming by Non-programmers. *Int. J. Man-Machine Studies,* 6, 237-260.
13 Moulton, P.G. and Muller, M.E. (1967). DITRAN - A Compiler Emphasizing Diagnostics. *Comm. ACM,* 10, 240-252.
14. Naylor, T.H. (1982). Decision Support Systems or Whatever Happened to MIS. *Interfaces,* 12, 92-94.
15 Pinsky, L. (1983). What kind of "Dialogue" is it when Working with a Computer? In T.R.G. Green, S.J. Payne and G.C. van der Veer (Eds.) The Psychology of Computer Use. Academic Press, London.
16 Sime, M.E., Green, T.R.G. and Guest, D.J. (1973). Psychological Evaluation of two Conditional Constructions used in Computer Languages. *Int. J. Man-Machine Studies* 5, 105-113.
17 Sime, M.E., Green, T.R.G. and Guest, D.J. (1977). Scope Markings in Computer Conditionals - a Psychological Evaluation. *Int. J. Man-Machine Studies,* 9, 107-118.
18 Soloway, E., Bonar, J. and Ehrlich, R. (1983). Cognitive Strategies and Looping Constructs : an Empirical Study. *Comm. ACM,* 26, 853-860.
19 Sprague, R.H. and Carlson, E.D. (1982). Building Effective Decision Support Systems. Prentice-Hall, Englewood Cliffs.
20 Volpano, D.M. and Dunsmore, M.E. (1984). Empirical Investigation of COBOL Features. *Information Processing and Management,* 20, 277-291.
21. Youngs, E.A. (1974). Human Errors in Programming. *Int. J. Man-Machine Studies,* 6, 361-376.