USER INTERFACES FOR DECISION SUPPORT SYSTEMS:

AN OVERVIEW

Edward A. Stohr

and

Norman H. White

September 1983

Center for Research on Information Systems
Computer Applications and Information Systems Area
Graduate School of Business Administration
New York University

Working Paper Series

CRIS #42

GBA #82-63(CR)

Published in: International Journal of Policy Analysis and Information Systems, Special Issue on Decision Support Systems, Vol. 6, 1982.

1. INTRODUCTION

Heavy involvement by managers in the use of computers for decision-making was an early dream of the MIS profession that has rarely been realized. Recently however VISICALC [1981] and a number of similar micro-computer-based accounting spread-sheet programs have defied the odds and been broadly used by managers. How did David succeed where Goliath failed? There can be little doubt that the answer lies in the 'user interface' provided by these programs--certainly they are not as powerful functionally as many software products available on mini or maxi-computers. The term 'user interface' covers all aspects of the communication between a user and a program including ergonomic, effectiveness and psychological factors. The study of user interfaces has recently emerged as an important area of concern for commercial software developers, computer scientists and researchers in the area of Decision Support Systems (DSS). Several books, (Mehlmann [1981], Shneiderman [1980]), and conferences (User Interfaces [1982] and Human Factors [1982]) highlight progress in this hitherto neglected area.

Our purpose in this paper is to discuss the user interface requirements for DSS software. In particular, we are interested in how generalized software systems can be designed to provide managers and their staffs with software tools that can be used to build DSS's that are pleasant and effective to use and readily modifiable to meet changing needs. Our discussion will concentrate on features that have been well-proven in a number of applications. However we will also mention some more futuristic possibilities to provide a longer range

perspective.

We will be concerned with the following roles:

Designers: the developers of the generalized DSS software.

Builders: the technical personnel within an organization who use the DSS generator to build databases and models.

Intermediaries: staff personnel who perform data retrievals and create models to support managers in their decision-making.

Managers: the decision-makers; managers who may or may not use the DSS in a hands-on mode.

In general we will adopt the point of view of the designers. Note that the builders, intermediaries and managers may be one and the same person or that group decision-making may be involved. The point here is that different users will have different levels of: (1) syntactic knowledge of the software (how to use it) and (2) semantic knowledge of the application (what should be done)--Shneiderman [1980]. A DSS should provide tools such as access to general purpose languages, and full-screen editors for the builder as well as menu-driven displays, on line documentation and a wide range of defaults for a managerial user.

We are interested in providing good interface characteristics in a general-purpose, transportable DSS software package. An architecture for such a package is outlined in Section 2. In Section 3 we describe some user interface alternatives together with a number of principles of good design that appear to have some empirical validity. In Section 4 we develop a set of detailed DSS interface requirements. Section 5 describes some generalized software for meeting these requirements. In Section 6 we provide an overview of

the role of languages in DSS; however we are mainly concerned in this paper with non-language aspects of user interfaces. Language facilities are covered in a companion paper (Stohr and White [1982]).

2. ARCHITECTURE FOR A DSS GENERATOR

Historically the first examples of DSS software were 'Specific DSS' that were built from scratch to provide computerized support in a particular decision situation. Many such systems are described in Alter [1980]. The disadvantages of this approach are that the systems were difficult to build and could not be used in other contexts. Recently however, a number of software systems have appeared with quite similar capabilities and structure. These 'DSS Generators' provide tools that can be used to build a wide variety of corporate DSS. Some commercial examples include EMPIRE [1982], EXPRESS [1982], IFPS [1982], PLATO [1982], SIMPLAN [1982], and XSIM [1982].

A list of different types of software that may be included in a DSS generator is given in Table 1. Obviously no current system contains all of these features. Nevertheless all of them have been used (or in some cases proposed). From Table 1 one can see that a versatile DSS generator requires a great diversity of software components. These might be invoked in any sequence and must be able to communicate with each other and with external data sources including the organization's operational information system. An even more important message of Figure 1 is that there are a large number of potential points of contact between users and the DSS. In fact, the interface software may comprise the greatest part of the system (over

GENERAL INTERFACE TOOLS		DATA-ORIENTED COMPONENTS	MODEL ORIENTED COMPONENTS	LANGUAGE TOOLS
Data-Entry Subsystem	Report Generator	Data Query Language	Model Command Language	Command Processor
Screen Forms Generator	Help/Training Aids	Data Definition Language	Model Definition Languages	High Level Language Compilers/ Interpreters
Graphics Input	Graphics Output	Data Base Management System	Model Management System	Parser-Generator Compiler-Compiler
Menu Generator	Full-Screen Editor	Data Conversion Facility (MIS Interface)	Model Library: - Data Analysis - Forecasting - Simulation - Financial Function - Mathematical Programming - Simultaneous Equations	
Voice Input	Voice Output	Data Dictionary	Model Dictionary	

TABLE 1

POSSIBLE SOFTWARE COMPONENTS FOR A

DSS GENERATOR

60% in the GADS system--Carlson and Sutton, [1974]).

There are two possible approaches to building a DSS generator with very general capabilities. The first strategy is exemplified by some commercial generators that focus on a particular function (say financial planning) but allow the user to link to other software when necessary. An extreme example of this approach is advocated by Donovan, [1976] whose GMIS system provides a framework in which many different software systems can be integrated using data base technology and a system of communicating virtual machines. In this way languages and modeling tools (APL, PL/1, TSP, EPLAN and Editors) were united with data base facilities (SEQUEL, IMS, QUERY-BY-EXAMPLE).

The second approach to building a general purpose DSS generator is to attempt to integrate a broad range of the capabilities shown in Table 1. This seems to be the approach of some commercial DSS packages such as XSIM, EXPRESS, and PLATO.

A design philosophy somewhere between these two extremes seems to us to be both desirable and feasible. Thus we believe that a DSS Generator should provide: (1) 'prepackaged software' in the form of modeling and data retrieval languages that can be used as--is by builders, intermediaries and managers; (2) tools that allow the builders to extend the capabilities of the generator and to interface it with other software as needed.

To focus our discussion we will use the framework depicted in Figure 1 (adopted from Ginzberg and Stohr [1981], see also Sprague and Watson [1976] and Bonczek et al [1980]). In a sense all DSS

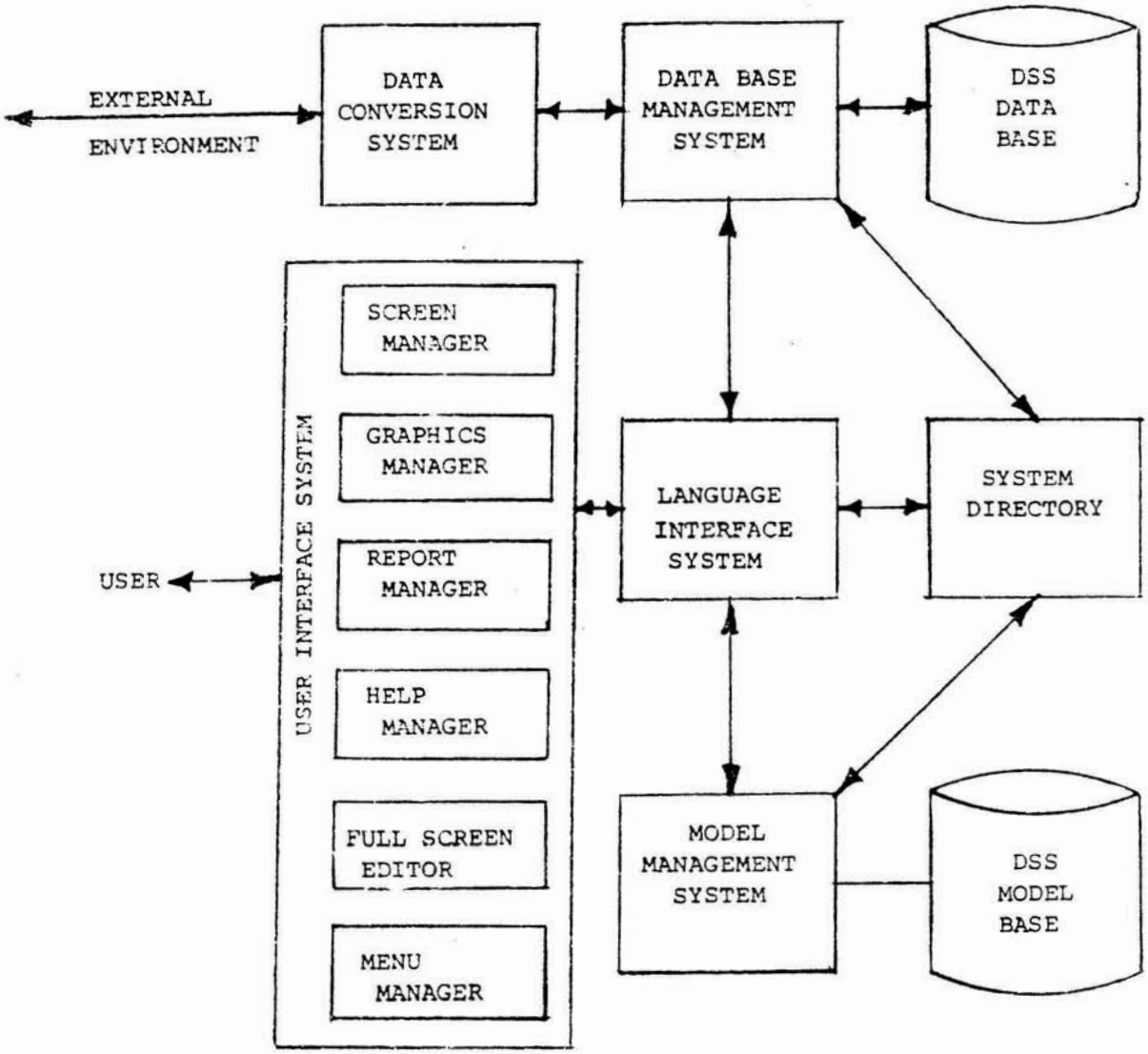

FIGURE 1

MAJOR COMPONENTS OF DSS GENERATORS

generators have the major areas of functionality shown in the figure to some degree. However our viewpoint is that each component is an independent software sub-system possessing a high degree of functionality. The Data Base Management System (DBMS), User Interface System (UIS), Language Interface System (LIS) and Model Management System (MMS) communicate with each other to integrate the different features in Table 1. Each of these four components can be regarded as an abstract machine that hides physical implementation details from the other components. Thus: the UIS is responsible for aspects of user-system communication; the DBMS manages logical and physical data access paths; the MMS manages the accessing and execution of procedures and programs that reside in the 'Model Base' library; the LIS consists of a number of language translators that interpret user requests directed to the DBMS, MMS and System Directory. The latter contains meta-knowledge about the data and procedures available within the planning system and their interrelationships. It provides information for the DBMS and MMS as well as online documentation, help and training aids for the user. Finally, the Data Conversion System allows communication between the resident data base of the DSS generator and external data bases and files.

Figure 1 does not necessarily represent a software architecture (for example the System Directory might reside in the DBMS and/or MMS). However it does represent a convenient functional grouping of the software that resembles some existing prototype systems. We will discuss the UIS and LIS in some detail here. For explanations of the roles played by the DBMS and MMS see Donovan [1976], Bonczek et al [1980] and Sprague and Carlson [1982].

3. INTERFACE DESIGN ISSUES

In this section we will discuss some general objectives for interface design, survey the available hardware and software design choices and finally describe research concerning how the design alternatives can be used to meet the design objectives. The discussion will be brief since we intend only to provide a background for the more detailed development of DSS interface requirements in the remainder of the paper.

Objectives for DSS Interface Design

In general a user interface should:

(1) Be easy to learn, use and remember

(2) Be helpful when problems arise

(3) Be suitable for both novice and expert use

(4) Be efficient in the use of system resources

(5) Promote efficient usage

(6) Promote effective usage--better decision-making

(7) Provide for a range of different media and interaction styles

More detailed lists of objectives for interactive interface design have been proposed by various authors and are reviewed in Shneiderman [1980], Ch.10. The major injunction for designers is 'know thy user'. The designed system should also know itself in the sense that it has sufficient knowledge to instruct the user and to prevent erroneous usage.

For DSS software the general objectives listed above can be sharpened somewhat. First, recalling Keen, [1980] we note that DSS is characterized by an evolutionary development process involving three dialogues between: the user and the builder, the user and the system, and the builder and the system. Since the objective of a DSS Generator is to speed this prototyping process we must support:

(1) The builder - by providing useful interfaces to programming and other development facilities. Since the builder will be a frequent user with high syntactic knowledge a compact and efficient style of discourse should be provided.

(2) The users - by providing help facilities and a range of interface styles. Managers may be infrequent users with low syntactic knowledge. They will need an easy-to-use 'big picture' interface that allows rapid access to important prepackaged information. Intermediaries will be frequent users: after a brief period of learning they will wish to progress to more complex applications and will be willing to learn more advanced system features and to stream-line their interaction.

Secondly, since a DSS is used to enhance problem solving processes it is important to provide support at the semantic level:

(1) A number of different external representations of the same information (e.g., tables and graphs).

(2) Methods for aggregating, grouping and filtering information, and for operating on grouped and aggregated objects.

(3) Short-term memory aids that help users know their current

position and the available alternatives.

(4) Longer-term memory aids that help users track and record intermediate calculations and trial results (an audit trail).

Finally, since the use of a DSS is often voluntary, we need to make the interface functionally useful and enhance the productivity and enjoyment of users. The latter requires an understanding of cognitive processes and of the determinants of user satisfaction. Empirical research (Zmud [1979]) has established the existence of persistant perceptual and cognitive patterns of problem solving among individuals. To support these different 'cognitive styles' we must provide interfaces that adapt rapidly to different information demands and sequences of processing steps.

Physical Design Alternatives

An initial physical design consideration involves the choice of a suitable combination of information mode, input/output device and mechanism for controlling operations. Some common alternatives are shown in Table 2. A DSS Generator may support several combinations such as (Character, Dumb CRT, Typing) and (Graphics, Graphics CRT, light-pen). A further hardware/software design consideration involves the number of 'information channels' (physical or logical terminals) available at any one site: (1) a single interface (2) multiple interfaces with only one interface active at any one time or (3) multiple simultaneously active interfaces. As an example of (2) one

Information Mode	Input/Output Device	Mechanism for Controlling Operations
Character (including numeric data)	Hard Copy Terminal Dumb CRT Semi-smart Smart CRT Large Screen Line Printer	Typing Function keys 'Picking' Device: Light-pen, touch-screen, mouse, joystick
Graphics	Graphics Terminal CRT and Remote Plotter	
Voice	Voice Synthesizer Voice Recognizer	Voice Command

TABLE 2

PHYSICAL INTERFACE DESIGN CHOICES

might have a graphics CRT and plotter; as examples of (3) one might have several 'windows' on one screen run by different application programs (Buneman et al [1977]) or several CRT's displaying spatial and other information at different levels of detail (Herot [1982]). Finally, for cooperative decision-making as in teleconferencing (Hiltz and Turoff [1981]), the design might involve multiple sites interacting using one or more information channels.

For a single information channel an interface design starts with the choice of one or more alternatives from within each of the three categories shown in Table 2. However many more design choices are required. Thus if the physical design involves (Character, dumb CRT, typing) the designer must still choose: the screen formats and interaction sequence for both input and output, the command language syntax, and so on. These alternatives will be considered in more detail in Section 4.

The range of technology available is very broad and expanding rapidly presenting interface designers with difficult challenges and interesting possibilities. For future reference, Tables 3A and 3B link the physical characteristics of various types of device to the interface applications that they enable.

Interface Design Principles

We turn now to a brief consideration of the major factors that should be considered in attempting to produce a cost-effective design. These are shown in Table 4 together with references to some related literature. They fall into four groups: user-oriented,

Device	Physical Attributes	Interface Styles[1]	Additional Feature
Hard-copy Terminal	80 to 132 characters Upper/Lower case	Query/Response, Command	
Dumb CRT	24 x 80 characters (up to 65 x 132)	Menu[2]	
Semi-smart CRT	Block Transmission Protected fields Insert/delete line/ character Clear Screen Cursor Positioning	Screen Forms[2] Full Screen Editing[2]	
	Scroll Up/Down		Split Screens
	Blinking, color, intensity, reverse video		Highlight important information
	Send modified fields only		Reduce transmission cost
	Printer port		Local hardcopy
	Video output		Drive monitors for peer presentations
	Multiple input ports		Connect to multiple systems simultaneously
Smart Programmable CRT	Down/Loadable Disk storage		Local edit checks Local storage of Forms
CRT Attachments	Light pen Digitzer Cross-hair Mouse		Picking, pointing alternatives to typing

1. Cumulative (earlier interface styles are also supported).
2. Transmission speeds > 1200 baud required.

TABLE 3A

CHARACTER-ORIENTED DEVICE CHARACTERISTICS

AND USER INTERFACE APPLICATIONS

Device	Physical Attributes	Additional Features
Black & White & Color Graphics CRT's	Bit-mapped raster	Allows hardware fill & other features
	Vector	Very fast for line drawing
	Segmentation and rotation	Rapid reorganization of a chart
	Zooming	Blow up portions of chart
	Color	16 + colors possible
CRT Attach-ments	Light pen Digitizer Cross hair Mouse	Direct manipulation systems possible
Black & White Graphics Printer	Matrix	Fast, inexpensive low resolution hard copy
	Electrostatic	Very fast, high resolution hard copy
Color Graphics Printer	Matrix	Medium fast, inex-pensive, low re-solution color
	Laser	Fast, expensive, medium resolution color
	Photographic	Very fast, expen-sive, slides and points (same res-solution as monitor)

TABLE 3B

GRAPHICS DEVICE CHARACTERISTICS

AND USER INTERFACE APPLICATIONS

CATEGORY	ISSUE/VARIABLE	REFERENCES
USER	Cognitive styles	Mason & Mitroff [1973] Zmud [1977]
	Naming conventions	Schneider [1982]
	Attitude - anxiety - enthusiasm	Walther & O'Neil [1974]
	Syntactic vs semantic knowledge	Shneiderman [1980]
TASK	Data retrieval	Vassiliou & Jarke [1982]
	Interactive problem solving	Meador & Ness [1974] Holloway & Mantey [1975]
	Multi-attribute decision-making	Carlson & Sutton [1974] Jacob & Sprague [1980]
	Intelligence, design, choice phases	Sprague & Carlson [1982]
EFFICIENCY	Response times	Miller [1973]
	Screen-forms for data entry	Mehlmann [1981]
	Screen-forms for data query	Greenblatt & Waxman[1978]
	Text editors: line vs full screen	Roberts [1982]
	Menu Drivers	Robertson et al [1981] Savage et al [1982]
	Prompt/response interface	Gaines [1982]
	Command languages vs menus	Gilfoil [1982]
EFFECTIVE-NESS	Graphic versus Tabular output	Remus [1982] Keen & Scott Morton [1978]

TABLE 4

SOME MAJOR DESIGN CONSIDERATIONS

task-oriented, efficiency and effectiveness. Although there does not seem to be an overall theory of interface design there are a number of design principles and empirical findings, some of which are shown in Table 5. However, many intriguing questions remain unanswered and there is as yet little guidance for cost-benefit analysis of the various interface choices such as color versus black-and-white graphics.

Before leaving this section we should mention the 'ROMC' methodology for DSS interface design (Sprague and Carlson [1982]). According to this technique, the design should proceed by finding (1) a suitable Representation (algebraic, tabular or graphic) of user concepts, (2) a set of Operations on those represention that are useful for problem-solving, (3) a set of Memory aids to overcome human memory limitations and (4) a set of Control mechanisms that allow the user considerable freedom in utilizing the three preceding sets of tools. The representations and operations part of this paradigm is similar to an observation by Shneiderman [1982] concerning the property of 'direct manipulation' that he feels characterizes good (and exciting) computer interfaces. Their basic characteristics are that the object of interest should be visible, that it should be manipulable directly rather than via a command language, and that all operations should be rapidly and easily reversible. As examples of good interface designs he cites the many successful video games that are now flooding the market.

General Principles:

- o Provide sufficient functionality and efficiency to make the system useful.
- o Reduce the number of actions and effort required to achieve a result (e.g., minimize keystrokes, eye movement and muscle fatigue).
- o Provide consistency of command formats and other conventions--announce rules to users.

User Control of the System:

- o Commands that change the state of the system should be easily reversible (e.g., allow users to back-out erroneous updates).
- o Provide immediate feedback—announce errors or confirm succesful actions.
- o Provide information on current state--position in a menu hierarchy, progress on a lengthy procedure execution.
- o Allow the user to stop processing at any response point without jeopardizing system integrity.

Assistance for Inexperienced Users:

- o Simple interfaces that don't require learning command language syntax.
- o Provide sensible defaults to avoid the necessity for learning and typing complex specifications (e.g., graphs and reports should have default formats).
- o Provide access to online 'help' documentation from all response points (e.g. by typing 'HELP').
- o Provide hard copy as well as online documentation.

Efficient Interfaces for Experienced Users

- o Provide short-cuts across menu-levels and in prompt/response interfaces.
- o Provide concise command languages with full functionality.
- o Allow abbreviations of commands.

Response-Times/Displays Rates

- o Tailor the efficiency of the interface to the task situation--reduce unpredictability.

TABLE 5

SOME PRINCIPLES OF INTERACTIVE INTERFACE DESIGN

We will return to these concepts of good design a number of times as we attempt to derive a set of software features that should be provided by a DSS generator.

4. REQUIREMENTS FOR A DSS INTERFACE

We first discuss three related topics: 'dialogue style', response-time/display rate requirements and help features. This is followed by more specialized treatments of each of the three information modes--character, graphics and voice that were originally introduced in Table 2.

Support for Dialogue Styles

The dialogue component of the interface is concerned with the two--way flow of control information that allows the user to direct the execution of the data base and model components, to receive help information and to perform data entry or programming tasks. The major styles of dialogue are shown in Table 6 together with the functions for which each appears most suited, the level of syntactic knowledge needed and the software and hardware requirements for effective usage. Useful discussions of the various techniques are contained in Martin, [1973] while Shneiderman [1980] summarizes some experimental results.

Prompt/Response - The computer prompts and users respond in a sequential conversational style. Some syntactic knowledge is required in order to choose an appropriate response. This can be provided conveniently by 'Help' features (e.g., the user types a question mark and receives a menu of permissible choices). This dialogue style is appropriate for inexperienced users who must be led through a number of different paths in order to specify their requirements. However, the large number of interactions makes a prompt-response interface cumbersome--especially if the user can anticipate the question path

	Dialogue Style				
				Command Languages	
Functions Performed	Prompt/ Response	Menu Selection	Screen Forms	Formal	Natural
Direct Operations	Simple Operations or Alternatives	Discrete Choices, Hierarchy Structures	Possible	Complex Operations	Very Simple Applications
Enter Data	Small Quantities	N.A.	Bulk Data Entry	Possible	N.A.
Retrieve Data	Low Selectivity	Low Selectivity	Good	Usual	Future
Define Data Base Schema Report Formats	Cumbersome	Possible for Simple Choices	Possible	Usual	N.A.
Define Models	N.A.	N.A.	N.A.	Usual	N.A.
Syntactic Knowledge Required	Medium	Low	Low	High	Low
Special Hardware Required	None	> 1200 Baud Transmission	Semi-Smart Terminal, > 1200 Baud	None	None
Special Software Required	None (Dialogue Manager Possible)	None (Menu Generator Possible)	Screen Forms Generator	Interpreter/ Compiler/ Parser	Natural Language Translators

TABLE 6
COMPARISON OF DIALOGUE STYLES

and the communication medium is slow. Prompt/Response interfaces can be made more useable by allowing users to "save" past request sequences so that the system does not repeat all the queries. The prompt/response style can be implemented without special equipment or software and can be used with both CRT and hardcopy terminals. However, to achieve uniformity of interface style and reduce application programming costs, special dialogue software has been proposed (e.g. Gaines [1981]).

Menu-Selection: Users are given a list of options from which to choose. Menus reduce user input to a minimum and show the user all possible responses. This makes them excellent interfaces for naive users, since the system is providing considerable structuring. Menu choices can also be made using light pens and other picking devices thereby eliminating keyboard input entirely. Unfortunately, a menu system involves a large amount of data transfer from the DSS system to the user's display device. Hence it requires CRTs and a high speed link to the computer system. Some DSS generators may soon use local micro-computers to store and display menus thereby reducing communication costs. Menus are often designed and managed as screen forms (see below). Special menu management software may also be used to provide a uniform interface to users across applications and to perform the special processing involved with hierarchies of menus (Robertson et al [1978]).

Screen Form: A complete screen of information is displayed in a 2-dimensional lay-out resembling a paper form. Users fill in blank fields using local terminal functions to move the cursor, edit, etc. Screen forms are suitable for both naive and expert users and offer speed advantages for tasks such as data entry. Although this is a standard interface in transaction processing applications it is not often provided by DSS Generators because they must be able to run in many different environments. Provision of a full-screen interface under these conditions requires 'Screen Manager' software to provide device independence. Most major computer manufacturers provide software for screen management (although this may be inaccessible from a DSS Generator). A high (at least 1200 baud) line speed and a 'semi-smart' terminal is required.

Command Language: Users enter language statements that direct the system to execute a specified sub-system, specify data to be retrieved, models to be run and so on. A common formal language syntax consists of a keyword followed by one or more parameters. Often this takes the form of a verb-noun pair (e.g. PRINT PROFIT). However more complex forms are also common especially for online data retrieval. Command languages are difficult for inexperienced users to learn and remember. They are appropriate for expert users because they minimize the amount of data transmitted. They are especially useful when the processing task is complex. Natural language command interfaces are currently under research and development. Users enter requests in a natural, English-like language. These are transformed into the appropriate DSS command and executed. The pitfalls in this approach include increased overhead, possible translation errors and lack of structure for the user. However natural language opens the door to two long-awaited possibilities: the use of the DSS by

non-trained personnel and the extension to voice input. Either or both of these would significantly change the way organizations use computers.

Table 6 shows that no single dialogue style dominates the others across all applications and all user classes. Command languages are best for expressing complex data retrievals or processing requirements. Menus are useful for helping users through a finite list of alternatives but can not handle complex requirements well. Futhermore there is a natural trade-off between the expressive power of the dialogue style and syntactic knowledge requirements. Gilfoil [1982] provides empirical evidence showing that user preferences migrate from menu interfaces towards command language interfaces over time. Combining these observations with the earlier prescription that a DSS should support different 'cognitive styles' we see that a DSS Generator should help the DSS builder tailor different interfaces to fit the various application areas and users.

Before being more specific about DSS Generator requirements however we note that the dialogue styles of Table 6 are generally intermixed. Thus, menus are often designed as screen forms and conversely screen forms are often designed with a menu section that shows how to select special functions to be performed and to move from screen to screen. Again, a formal command language interface is often overlaid upon query-response (Gaines, [1978]) and menu selection (Mehlmann, [1980]) interfaces to speed the interaction process for more experienced users. This allows users to move directly through a series of prompts or down several levels of menus by typing multiple responses on a single line in a command language style.

In summary, a DSS Generator should provide a number of different dialogue styles and the ability to switch between them. These capabilities can be designed into the 'prepackaged software'. However the builder must also be able to use the language facilities and/or procedures of the DSS Generator to construct special interfaces for each DSS application. It should at least be possible to build prompt/response and menu selection interfaces. Screen forms software would seem to be the next most desirable feature in a DSS generator because of its desirable interface properties and because the ability to rapidly generate formatted screens aids the DSS prototyping process.

Response Time and Display Rate Requirements

Response time requirements vary with the nature of the task. Command and Control systems may require instant response. For DSS applications greater variability is permissible. Typing and cursor control commands require 0.1 second response times; for simple data retrievals response times in the range of 0.2 to 3 seconds are reasonable; major computations and file loadings may take longer (up to 15 seconds) without disturbing users (R. Miller [1968]). It has been found that predictability of response time is an important design goal (L. Miller [1977]). Users need predictability to organize their time optimally between thinking and doing. Some experiments involving problem-solving tasks show that the total time to solve a problem may be fairly invariant to changes in response times over low ranges but may increase after that (Shneiderman [1980], Ch.10).

We have already indicated that faster display rates (>1200 baud) enable effective use of full screen interfaces. Faster display rates are also beneficial for scanning(rather than reading) text and for graphics applications.

Help Features

A user interface should be informative but not verbose, friendly but not 'human' and forgiving but not unmindful of human errors. In this section we briefly survey interface features that can help convey this impression.

We begin with the informative aspect. Users need assistance both in learning a system (what can I do?, how can I do it?) and in using it (where have I been?, where am I now?, what can I do next?). Learning can be facilitated by online documentation (although hardcopy manuals can not be dispensed with - Dunsmore [1980]). Some features that might be supplied are now described.

Online System Documentation: 'Help Files' describing major system concepts, facilities and commands should be accessible from all command levels of the DSS. In some systems users type 'HELP name' in response to a prompt to receive instructions about the named object. Omitting the name gives a list of the objects for which help can be requested. Other systems provide menu access to documentation or special help screens. Hierarchical documentation in which successive requests for help produce more detailed information provides less detailed information for more experienced users who may need only a reminder but also accomodates the more detailed requirements of novice users.

Online Data and Model Dictionaries: Users must be encouraged to document their data and models by supplying labels and explanations. The system should automatically supply contextual information such as time and date and user identification. At a minimum, variable and data item lists for currently running models and attached files are

supplied on demand by most current systems. More comprehensive 'knowledge bases' utilizing artificial intelligence techniques are being developed (Elam, et al [1980]).

Naming Conventions: Human factors research has shown that command language naming conventions can affect learning and retention. For example, 'congruent' command pairs such as ('get' and 'put') rather than ('get' and 'store') are helpful (Schneider [1982]). The modeling language should also allow the builder to use meaningful names.

Current Status Information: An online interface can be confusing - especially when used for exploratory problem solving tasks as in DSS applications. It is important to keep users informed of their position in menu hierarchies and on the status of currently executing commands. The latter can be done by supplying progress reports for long operations and confirmatory messages for all non-trivial operations.

Defaults: Useful defaults should be provided for most operations to reduce the need for learning and typing detailed specifications. Experienced users should be able to override thes defaults easily.

We turn now to the friendly (lending-a-hand) features that assist users in executing commands. The following techniques can aid short-term memory and reduce the impact of clerical errors:

Command Syntax Prompts: Users can 'discover' the syntax for a command line by typing a part of the command followed by a '?'; the system responds with a list of options from which the user chooses. This process can then be repeated for the next part of the command.

Recognition: Users type part of a command or data name and press an 'escape' key; if the typed portion uniquely identifies an item that is valid in the current context the system responds by typing the rest of its name. A combination of syntax prompts and recognition can be a particularly useful feature.

Editing and Saving Command Strings: It should be possible to edit command language statements both prior to submission for execution and subsequently if a mistake was made or if a similar command is to be typed. The current command string should always be saved for this purpose. Users should also be given the option to name and save commands for future use - thereby building their own private language. Automated editing by the DSS using flexible parsers and a spelling checker is also very useful.

Finally, a system should be forgiving and helpful when mistakes are made. Some useful system features are:

Error Messages: These should have a uniform format, should state what went wrong and, more importantly they should indicate how to correct the error. The tone of the messages should be neutral and informative rather than hostile or jocular (Shneiderman [1980]).

Reversible Actions: The DSS must protect the user against severe mistakes such as destroying a file. Automatic file back-up is one technique. Providing temporary workspaces during user sessions can help make actions taken by the user reversible. For example if data is deleted it should also be possible to 'undelete' it. Many editors provide this capability by automatically storing the most recently deleted text.

Judicious use of the above techniques can encourage the acceptance and increase the effectiveness of the DSS. User-friendly features should be built-in to the prepackaged portion of the DSS (via the Help Manager of Figure 1). DSS builders should also be able to document their models and store help messages for later online access by users. Assistance in doing this is not usually provided by DSS Generators. Model and data dictionary software should definitely be available. Also the modeling language should provide direct access files and language input/output facilities that enable the builder to provide useful help and error messages. Command syntax prompts and recognition require complex programming that might be provided by the UIS component (see later).

Character-Oriented Input and Output

A DSS must allow for ad hoc entry and display of data. The ease and flexibility of these processes can greatly influence user acceptance and effectiveness.

Considering first the input side, bulk data-entry into the DSS data base will usually be performed through the Data Conversion System. However data-entry and editing functions should also be available through the DSS interface. Screen forms displaying field labels and blank fields (or current values) that can be overwritten by cursor positioning and typing are probably the best means for doing this, (Mehlmann [1981]). Depending on the system being used the model definition statements must also be entered - preferably using a full-screen character-oriented editor that is accessed directly from the DSS (rather than from the operating system command level).

The data entry and model definition tasks will usually be performed by a system builder or intermediary; the retrieval and display of data and model results however may be performed by managers and in any case the results must be suitable for their consumption. Regarding data retrieval we note from Table 6 that both screen forms and formal language techniques are possible. Query-By-Example (Zloof, [1977]) is the major example of a screen forms oriented language. Data base relations are displayed as tables on the screen and users specify queries by typing 'examples' of the results they wish to obtain in the appropriate rows and columns of the tables. Experimental results seem to indicate that users can learn QBE more rapidly than a formal query language, are faster in formulating queries and have at least as high a success rate (Greenblatt and Waxman, [1978]).

Turning now to the display of character-oriented information, Table 7 contains a summary of desirable report generator capabilities. Note that the first half of these are concerned with formatting features and space limitations while the latter half are concerned with different ways of arraying and transforming the information. Thus a good report generator will have capabilities that overlap those of a DBMS Query Language together with some modeling capabilities (hence the popularity of FOCUS [1981], RAMIS[1981] and other combination report writer/data management systems in DSS). The report manager accepts the outputs of DBMS retrieval requests and model executions and allows them to be tailored to the demands of the moment. This reduces the complexity of the query and model software and at the same time makes the task of coding retrieval requests and model definition statements easier.

The report generator should be able to produce online and offline hard-copy reports and interactive CRT displays. The latter present a special challenge because of the limited display sizes of current screens - usually 24 lines of 80 characters. There are several approaches to overcoming this limitation and (perhaps) even turning it to an advantage. All require a 'semi-smart' terminal (see characteristics in Table 3).

Moving Window: The information to be displayed is laid-out on a (virtual) two-dimensional surface, the user can rapidly move the viewing area (CRT screen) relative to the surface in a vertical or horizontal direction by pressing function keys (up, down, left, right).

Split Screen: Similar to the above except that the screen can be split into two or more 'windows' that can be independently scrolled. For example as a memory aid, one may wish to hold row labels in the left-half screen while moving columns (and column labels) horizontally

(1) Formatting of numeric data - rounding, commas, dollar signs, etc.

(2) Labeling - substitution of informative descriptions for variables and computed information.

(3) Pagination and wrapping - wrapping avoids problems with reports that are too wide to fit on the page or screen.

(4) Rows/Columns transposition - used to overcome page width limitations or simply to provide another view of the data.

(5) Interactive editing - swap or delete rows and columns, improve headings.

(6) Sorting - numeric, alphabetic, multiple fields.

(7) Aggregation - imbedded data hierarchies often need to be displayed together with totals and subtotals. Some report writers allow this to be done both down-the-page and across it.

(8) Computed rows and columns - the ability to compute new rows and columns as functions of existing rows and columns.

(9) Invisible rows and columns that can be used for computations but not printed.

(10) Output to external files - both to store the report and communicate with other systems.

(11) Device independence - ability to format reports for different CRTs and printers

TABLE 7

DESIRABLE REPORT GENERATOR FEATURES

into view in the right half screen. Similarly one can alter input parameters in one half-screen and view their impact on variables of interest in the other half-screen.

The great popularity of the micro-computer-based accounting spread sheet packages is in large part due to their use of the moving window and split-screen devices. Obviously even unsophisticated computer users find these forms of direct manipulation natural and easy to use. This idea might be extended to allow all data base objects (model definitions, data files and reports) to be viewed through windowing techniques.

As a final example of an innovative interface for character data the following technique seems to qualify as an easy-to-use 'big picture' interface for managers.

Hierarchical Reports: A variant of the 'zooming' idea that will be mentioned later in the section on graphics - however it is useful for character displays and does not require graphics terminals. The idea is illustrated in Figure 2 (adapted from Mehlmann [1981]). To obtain more information about an aggregated data item one simply chooses it by placing the cursor (or a picking device) over it. This process can potentially proceed down to the individual transaction level.

To summarize our discussion concerning character information in a DSS we note: (1) the emerging importance of full-screen-oriented interfaces in data entry, data retrieval, problem-solving and reporting applications and (2) the necessity for providing a powerful report generator that can add variety and flexibility to the user interface and also reduce programming requirements.

Graphics Input and Output

XYZ Consolidated Income Statement, 19XX

Sales	XXX
Cost of Goods	XXX
Other Expenses	XXX
Net Profit	XXX

Income Statement by Division, 19XX

	Division A	Division B	Division C
Sales	XXX	XXX	XXX
Cost of Goods	XXX	XXX	XXX
Other Expenses	XXX	XXX	XXX
Net Profit	XXX	XXX	XXX

Sales by Division and Product, 19XX

	Division A Division A	Division B Division B	Division C Division C
Fidgets	XX	XX	XX
Widgets	XX	XX	XX
Gidgets	XX	XX	XX
Total	XXX	XXX	XXX

FIGURE 2

EXAMPLE OF HIERARCHICAL REPORTS

The advent of cheaper graphics display terminals and plotters together with more sophisticated graphics software has opened the door to new modes of man-machine interaction in problem solving tasks (for an overview see Newman and Sprowll [1979]). In this section we review some successful uses of graphics in DSS and identify desirable features that might be included in a DSS Generator. Careful cost/benefit analysis should precede the selection of any of the more advanced techniques we will be discussing. This is particularly true since empirical evidence does not always favor the use of graphics. Remus [1982] reviews some seemingly contradictory research concerning the relative merits of graphic and tabular presentation methods in problem solving and suggests that the choice of method depends on the task situation.

We distinguish six classes of graphics applications: (1) data analysis, (2) business presentation graphics, (3) information retrieval graphics, (4) design graphics (including CAD), (5) problem solving graphics, (6) interactive interface graphics. Referring to Simon's[1965] three phases of decision-making we would say that applications (1) through (3) above are mainly used in the intelligence phase; applications (4) and (5) are mainly used in the design and choice phases respectively, while application (6) involves the use of graphics techniques to improve the user interface itself and so could be used in any phase.

There are several areas in which the usefulness of computer graphics is beyond question. These include <u>data analysis</u> applications in statistics, marketing and forecasting. Some graphical tools used

Capabilities	Necessary Resources
Data Analysis Graphics Tools	
Histograms	High level graphics software
Line Graphs	
Scattergrams	
Contour Plots	CRT
Exploratory Data Plots	Black & White or Color Plotters
Chernoff's 'Faces' (Wilkinson [1982])	
Business Presentation Graphics	
As for data analysis plus	Screen Projection Slides & Hardcopy (Color-enhanced)
Pie Charts	
Kiviat Star, [op cit]	
Bubble graph, [op cit]	
Information Retrieval Graphics (SDMS)	Interactive graphics terminals with input devices, special software and data base software
Design Graphics	
Interactive Problem Solving Graphics	

TABLE 8

GRAPHIC CAPABILITIES AND NECESSARY RESOURCES

in the data analysis phase are shown in Table 8. The first three of these would be included as a matter of course in any DSS. Early computer systems produced these graphs using character-oriented terminals and printers. Although such plots are useful their low resolution can be inconvenient. It is now common to use either a graphics CRT, a plotter or both. The CRT is useful for exploratory analysis while the plotter is used for more permanent records or communication to groups since plots are time consuming to produce. Although color graphics may seem appealing we are not aware of any solid evidence that would justify its cost in all data analysis applications. Data analysis graphics does however require: (1) a highly interactive system with short (< 10 seconds) response times, (2) powerful defaults for selection of axes, scales, labels, etc., (3) device independence, and (4) a simple and powerful command language for overriding defaults.

The use of business presentation graphics is expanding rapidly. This involves the automated production of color slides and transparencies for oral presentations and high quality illustrations for business reports. Table 8 lists some common formats. Here aesthetics is very important. Sometimes there is a marketing flavor to this - but often it is just a means for providing a good 'interface' to a busy or otherwise reluctant decision- maker or group of decision-makers. Thus presentation graphics may be generated by an intermediary with the real decision-making performed offline. Keen and Scott Morton [1979, p20] and Nash [1977] describe the benefits of simply replacing numerical tabular output by the equivalent information in the form of pie charts, histograms and time series

plots. One major advantage was increased interpersonal communication. A good presentation graphics system will need all the features of the data analysis system (except possibly rapid online response). Usually color graphics will be required.

Direct manipulation of visual images has been shown to be an effective data retrieval interface for both novice and expert users. A prototype 'Spatial Data Management System' (SDMS) is described by Herot [1981, 1982]. Users are given a graphical view of the data base on three color terminals. Entities may be represented on the screens by icons (e.g., images of oil wells or ships). The positions of the icons on the screen may reflect a real world spatial relationship (e.g., maps showing oil leases) or classificatory relationships (e.g., classes of ships by country). To query an SDMS data base users manipulate a joy-stick to position themselves on a 'world-view' of the data on one screen and receive a more detailed view of the object of interest together with textual information from a symbolic data base on another screen. By twisting a knob they can 'zoom-in' to obtain yet more detailed views or return to a higher level overview. SDMS also provides a formal query language for conventional data base queries. Another example of spatial data management is described by McDonald [1982]. In this case both motion and still pictures are used to find objects of interest (for example items on a supermarket shelf).

The SDMS concept offers exciting possibilities since it reduces learning requirements and provides users with an easily controllable interface. Further research on the advantages and disadvantages

relative to conventional data base query systems in different task environments is needed.

Most examples of design graphics occur in specialized areas such as CAD (Computer Aided Design) in manufacturing. Systems analyst 'work benches' are another area of great potential (Ivie [1977]). Workbench systems allow analysts to interactively construct graphical portrayals of information systems (data flow diagrams, HIPO charts etc.) and to link them to a data dictionary. CAD and Workbench applications require facilities for graphics input and software to manipulate and store the resulting representations. These are highly sophisticated and specialized DSS's that will not be discussed further here. However graphical automated analysis tools might be significant in DSS since they can help in the rapid development of reliable prototypes.

Most uses of graphics in DSS are passive in the sense that they display information that could otherwise be presented numerically. This may greatly enhance user acceptance and decision-making capability. However conversational graphics is now being used in interactive problem-solving applications, to replace or augment more traditional management science algorithms. . Here the pattern recognition and intuitive problem-solving capabilities of humans are combined with the computational and retrieval capabilities of computers. Jacob and Sprague, [1979] cite several possibilities and give an example of optimal facility location using human geometric intuition. Another example is provided by the GADS (Geodata Analysis and Display System) which has been used in a number of situations

including the redesign of police beats (Carlson and Sutton, [1974]) and school districting (Holloway and Mantey, [1976]). Different operations research models for solving spatial problems are utilized but the key characteristic of the system is its ability to display alternative problem solutions as overlays on maps. This was used successfully by novice users to resolve difficult multiple criterion allocation problems.

Interactive interface graphics involve the use of graphic symbols and techniques to increase the efficiency, effectiveness and satisfaction of the man-computer interaction. Generally 'direct manipulation' of simulated real-world objects is involved. Some interesting examples are found in the area of automated office systems where, for example, the CRT displays a desk-top complete with 'papers' that can be withdrawn from an 'in-basket', read and then stored in 'file cabinets'or forwarded to others (Smith et al, [1982]). As a long-range trend it seems likely that office automation and DSS applications will merge into a more general 'Management Support System'.

Graphics capabilities are rapidly becoming a definite requirement for an effective DSS. A DSS generator should encompass the usual business presentation graphics as well as allow the DSS builder to develop their own specialized graphic formats. Future DSS's should allow the system full access to interactive graphical input as well, so that users can communicate using light-pens, mouses, and other interactive graphic input devices.

Voice Communication

Voice output from computers is now a common phenomenon in stock exchange and telephone applications where it reduces the cost of human operators. It can also be useful in command and control systems as an alerting mechanism. We are not aware of any applications in problem solving DSS applications. On the other hand, voice input of commands, text and data could be of great use in DSS especially for inexperienced users. However voice recognition systems currently work only with limited accuracy on restricted vocabularies. For the present it may be possible to use voice input for data or commands such as menu choices to eliminate typing. Eventually voice recognition systems and natural language parsers should be perfected giving an entirely new type of interface. For an overview of voice input applications see T. Martin, [1976].

Summary

The extensive array of desirable user interface capabilities that has just been presented provides a formidable challenge to the implementors of a DSS Generator. This is compounded by the fact that it must provide builders with tools to build their own interfaces. For the purpose of this section we will assume that the DSS Generator contains prepackaged data management, modeling, report writing and graphics facilities. Two questions arise (1) What kind of interfaces should be built into the prepackaged software? (2) What tools should be provided for DSS builders to allow them to build their own interfaces? We now address these questions.

Prepackaged Software Interfaces: At a minimum the DSS Generator should provide:

(1) A low selectivity prompt/response or menu choice interface that will allow experts and novices to initially invoke the major components of the DSS. Extending this interface further will allow novice users to access predefined reports and graphs and run existing models. Thus managers will be able to do useful work with a minimum of learning.

(2) Concise formal languages with high expressive power to allow more expert users to perform data retrievals define models, carry-out sensitivity analyses, define reports and produce graphs for data analysis and presentation purposes.

(3) Online help features that explain major concepts and commands and access data and model dictionaries.

Requirements (1) and (2) above should conform to a kind of 'sandwich' principle. The prompt/response or menu interfaces should be grown downwards (say up to three menu levels) until useful prepackaged functions can be performed by novice users. On the other hand the formal command languages should 'cover' the full functionality of the DSS Generator and be built upwards in terms of scope and span of control so that they overlap with the simpler forms of interface. In this way the system can accomodate learning and also satisfy any preferences of users to migrate from verbose to more concise interfaces over time (Gilfoil, [1982]).

While the above requirements seem to us to be minimal we believe that commercial DSS Generators will soon support full screen input and output and some of the more advanced graphics interfaces described above.

Tools to Build Interfaces: The DSS Generator should at least provide the builder with the following two capabilities:

(1) The modeling language should have the input-output and formatting capabilities of a modern high-level language such as FORTRAN or PL/1; this will allow the builder to provide simple forms of the query/response and menu interfaces.

(2) Access to the prepackaged software components should be provided via procedure calls or the syntax of the modeling language. This will allow builders to provide specialized interfaces for standard queries, reports and graphs.

In addition it would be desirable to be able access general programming languages and operating system utilities via procedure calls from the modeling language. This would facilitate extension of the tools available to model builders (i.e., the Model Base) and also allow access to useful software packages such as screen forms generators or graphics packages that are available on the host computer but not provided by the DSS Generator.

5. THE USER INTERFACE SYSTEM

The UIS (see Figure 1) is a separate 'layer' of software that provides many interface functions and isolates users from variations in physical device characteristics and configurations. Thus the UIS provides: (1) device independence, (2) a uniform interface for the users to state their requests, (3) elimination of the need for special application programming to generate displays and reports.

Until quite recently programmers built user interfaces using the I/O and formatting facilities of high-level languages such as COBOL, PL/1 and APL. Each program contained a complete specification of the communication to and from users--complete in the sense that the operating system and device managers needed no further information in order to execute the communication process. The advent of smart terminals, graphics devices and input-output mechanisms that rely on other senses (touch and sound) has changed this picture. The program still produces and consumes information in a logical sense but the input and presentation formats may require other specialized software systems and/or access to a data base. Full screen formats for example are usually difficult to define in a host programming language because the logic and character codes involved are device-dependent. Screen Manager software can relieve the programmer of much of this burden by monitoring transmission between the program and terminal and accessing a library of screen definitions when invoked by the program. Note that the screen definitions are stored in the library by a separate process. Once stored, they can be used by many programs independently of the language in which they are written. The Screen Manager

contains information on individual device characteristics and the ability to map logical device identifiers to physical devices. Thus it acts as a buffer between the interface device and the program providing what might be called 'interface independence'. All of this is reminiscent of a data base management system which provides shared access to data by many different programs together with 'data independence', (Date, [1981]).

The other major components of the UIS as depicted in Figure 1 (Graphics, Help, Report and Menu Managers) also provide the possibility for interface independence. Again libraries of graph formats, help messages, report and menu definitions can be built up and shared by different application programs and subsystems of the DSS Generator. To produce a report for example, an application program can write the relevant information as a series of logical records and invoke the Report Manager passing it the identifier of a stored report definition. The Report Manager can handle formatting details automatically taking into account both the characteristics of the device used to produce the report (CRT, printing terminal, line printer or voice synthesizer) and the identity of the user. The latter information can be used to customize the report or perhaps to mask certain fields for security reasons. A similar sequence of events would take place if the report was requested interactively by the user. In this case the user rather than the application program would identify the report definition and data file.

Continuing the analogy between the UIS and a DBMS we see a similar distinction between host-language and self-contained systems. A host- language system provides specific capabilities that can be invoked by an application program either by enhancements to the source language in which the program is written or through sub-program calling procedures. Self-contained systems are not designed to be called by other programs. They therefore control the user interface and environment more closely at the expense (usually) of generality and the ability to communicate with other applications. A Screen Form Manager designed as described above is an example of a host-language system. On the other hand some Presentation Graphics packages provide examples of self-contained systems. Results of other programs must be transfered to such systems via files or even by data-entry. Generally self-contained systems evolve into more general purpose systems over time by providing more programming language and file communication capabilities. The UIS should be a host-language system from the point of view of the DSS—that is it would be callable from both the command and modeling language levels.

The GADS system described above provides a good example of a UIS (see also Sprague and Carlson, [1982, ch.7] and Yonke and Greenfield [1980]). Before leaving this section we describe one attempt to build a UIS in more detail. The DAISY system (Buneman et al,[1978]) allows multiple logical terminals or 'windows' to be displayed on the user's CRT. Each window provides a separate independent information channel for input and output enabling the user simultaneously to work on several tasks or to view different information groupings. For Command and Control systems one window can be devoted to an 'alerter' function

that monitors a data base and/or incoming messages for a change in system state requiring immediate attention. In an office support system different windows can be devoted to reminders, phone and mail messages, imminent meetings and so on. The DAISY system also supports a large screen for group decision- making, voice output and graphics. The windows can be moved, overlaid and expanded and contracted by the user. Input devices include the mouse and trackball.

6. THE ROLE OF THE LANGUAGE INTERFACE SYSTEM

Languages provide the 'glue' that allows DSS builders to assemble individual components into an integrated model to solve particular problems. They also form an important part of the interface for intermediaries and (possibly) managers. However our placement of the UIS between the user and the LIS in Figure 1 serves to emphasize that even the most powerful and user- friendly of languages may seem obscure and difficult to users if other aspects of the interface such as communication speed, error rates and help features are inadequate (Turner et al, [1982]).

The UIS and LIS jointly perform many complex transformations: (1) Between the issue of a command by the user and its translation into a language statement that will activate (say) the DBMS or MMS, (2) Between the output format of the DBMS or MMS and its displayed format as viewed by the user. On the input side for example the command may be quite English-like. Alternatively it may not initially be a language statement in any traditional sense but rather a touch of a light-pen on the user's screen, a hand gesture or even an eye

movement (Bolt [1982]). All of these inputs must be transformed into operations which are executed at the machine level. On the output side, as we have seen, the raw output from the DSS may be passed through screen forms, report generators or even voice synthesizers on its way to the user. The LIS role in these transformations is to perform the language translations required.

In terms of system architecture the DBMS and MMS may each contain powerful user-friendly language interfaces. In this case the UIS and LIS might serve more in the role of a communications front-end although they should also allow the builder to integrate the languages with special screen formats, help features, and menu displays. Alternatively the DBMS and MMS languages might be mathematically based and more suitable for expert users. In this case the LIS would contain more English-like, user-friendly language translators. These would translate to the DBMS and MMS 'target' languages. This approach is used by a number of data base retrieval systems. For example the 'restricted natural language' system USL (Lehmann [1978]) translates to the formal keyword language SQL (Astrahan et al, [1975]).

As a more advanced feature the LIS might allow DSS builders to construct their own languages specially tailored to fit particular applications or particular user styles. To do this the LIS would contain a parser-generator (Aho and Ullman, [1978]). Note that some natural language interfaces contain this feature to allow application specific vocabulary and grammar rules to be defined (Stohr et.al. [1982]).

7. CONCLUSION

Our discussion of user interfaces for DSS was conducted within the framework of an architecture for a DSS Generator. We surveyed the physical design alternatives and some design principles that have evolved as a result of both practical experience and research. Our discussion of interface features was broadly based and included some techniques that are still under development in various laboratories.

The commercial success of a DSS Generator is strongly related to the quality of its user interface. However as we have seen there are a bewildering number of features that might be provided. These two facts lead us to believe that current piecemeal approaches to the provision of interface capabilities will be replaced by more organized approaches employing generalized software and based on 'logical models' of user interface requirements. The recently developed CORE Graphics standard (Michener and van Dam [1978]) exemplifies a movement toward a theory of user requirements and standard graphics operations for the implementation of user interfaces. We expect this trend to continue and that software resembling the UIS described in this paper will be developed. Providing user interface services in this way will allow the DBMS, MMS and user developed models to share the same interface program logic. Device independence at the physical level and what we have called interface independence at the logical level will be a natural outcome of this separation of software functions. Finally, and most importantly, the UIS will present a uniform and coherent interface to the user.

REFERENCES

1. Aho, A.V. and J.D. Ullman, Principles of Compiler Design, Addison-Wesley Reading, Mass., 1978.

2. Alter, S.L., Decision Support Systems: Current Practices and Continuing Challenges, Addison-Wesley, Reading, Mass., 1980.

3. Astrahan, M.M. and Chamberlin, D.D., 'Implementation of a Structured English Query Language', Communication of the ACM, 18, October, 1975.

4. Bolt, Richard A., 'Eyes at the Interface', Proceedings Human Factors in Computing Systems, Gaitherburg, Maryland, March, 1982.

5. Bonczek, Robert H., Clyde W. Holsapple and Andrew B. Whinston, 'The Evolving Roles of Models in Decision Support Systems', Decision Sciences, pp. 337-356, Vol. 11, 1980.

6. Buneman, O. Peter, Howard L. Morgan and Michael D. Zisman, 'Display Facilities for DSS Support: The Daisy Approach', Database, Vol. 8, No. 1, Winter, 1977.

7. Carlson, E.D. and J.A. Sutton, 'A Case Study of Non-Programmer Interactive Problem Solving', San Jose, Calif., IBM Research Report, RJ1382, 1974..

8. Date, C.J., An Introduction to Data Base Systems, Addison-Wesley, Reading, Mass., 1981.

9. Donovan, John, J., 'Database System Approach to Management Decision Support', Transactions on Database Systems, Vol. 1, No. 4, pp. 344-369, December, 1976.

10. Dunsmore, Herbert, 'Designing an Interactive Facility for Non-Programmers', Proc. ACM National Conference, 1980, pp. 475-483.

11. Elam, Joyce J., John C. Henderson and Louis W. Miller, 'Model Management Systems: An Approach to Decision Support in Complex Organizations', Proc. 1st International Conference on Information Systems, Philadelphia, PA, December, 1980.

12. EMPIRE: Applied Data Research, Inc., Princeton, NJ, 1982.

13. EXPRESS: Management Decision Systems, Waltham, Mass., 1982.

14. FOCUS: Information Builders Inc., New York, NY, 1982.

15. Gaines, Brian R., 'The Technology of Interaction--Dialogue Programming Rules', Int. Journal of Man-Machine Studies, 20, 1981, pp. 133-150.

16. Gilfoil, D., 'Warming Up to Computers: A Study of Cognitive and Affective Interaction over Time', Proceedings Conference on Human Factors in Computer Systems, Gaithersburg, Maryland, March, 1982.

17. Ginzberg, M.J. and E.A. Stohr, 'Decision Support Systems: Issues and Perspectives' in Ginzberg, M.J., W. Reitman and E.A. Stohr (Eds), Decision Support Systems: Proceedings of the NYU Symposium, 1981, North-Holland (forthcoming).

18. Greenblatt, D. and J. Waxman, 'A Study of Three Database Query Languages', in B. Shneiderman (Ed.) Databases: Improving Usability and Responsiveness, Academic Press, New York, 1979.

19. Herot C.F., 'Spatial Management of Data', ACM Transactions on Data Base Management Systems, 5, 1981, pp. 493-513.

20. Hiltz, S.R. and M. Turoff, The Network Nation: Human Communication via Computers, Addison-Wesley, Reading, Mass., 1978.

21. Holloway, C.A. and P.E. Mantey, 'An Interactive Procedure for the School Boundary Problem with Declining Enrollments', Operations Research, Vol. 23, No. 2, pp. 191-206, March-April, 1975.

22. Human Factors: Proceedings of Conference on Human Factors in Computer Systems, Gaithersburg, Maryland, March, 1982.

23. Huysmans, J.H.B.M. 'The Implementaion of Operations Research', New York: Wiley-Interscience, 1970.

24. IFPS:EXECUCOM Systems Corporation, Austin, Texas, 1982.

25. Ivie, E.L., 'The Programmer's Workbench - A Machine for Software Development', Communications of the ACM, 1977, 20, pp. 746-753.

26. Jacob, Jean-Paul and Ralph H. Sprague, Jr., 'Graphical Problem Solving in DSS', Data Base, Vol. 12, Nos. 1 and 2, Fall, 1980.

27. Keen, P.G.W. and M.S. Scott Morton, Decision Support Systems: An Organizational Perspective, Addison-Wesley, Reading, Mass, 1978.

28. Keen, Peter G.W., 'Adaptive Design for Decision Support Systems', Database, 12, Fall, 1980.

29. Lehmann, H., 'Interpretation of Natural Language in an Information System', IBM Journal of Research and Development, 22, September, 1978.

30. Mantey, P. and E. Carlson, 'Integrated Data Bases and Muncipal Decision-Making', Proc. AFIPS 1975 NCC, AFIPS Press, Montvale, NJ, 1975.

31. Martin, J., Design of Man-Computer Dialogues, Englewood Cliffs, NJ, Prentice-Hall, 1973.

32. Martin, T.B., 'Practical Applicatons of Voice Input to Machines', Proceedings of the IEEE, 64, 1976, pp. 487-500.

33. Mason, R.O. and I.I. Mitroff, 'A Program for Research on Management Information Systems', Management Science, Vol. 19, No. 5, 1973, pp. 475-487.

34. McDonald, N.H., 'Multi-media Approach to User Interface', Proc. NYU Symposium on User Interfaces, New York, 1982.

35. Meador, C.L. and D.N. Ness, 'Decision Support Systems: An Application in Corporate Planning', Sloan Management Review, Vol. 15, Winter, 1974.

36. Mehlmann, M., When People Use Computers: An Approach to Developing an Interface, Prentice-Hall, Englewood Cliffs, NJ, 1981.

37. Michener, J.C. and A. van Dam, 'A Functional Overview of the Core System with Glossary', Computing Surveys, 10, December, 1978.

38. Miller, L.H., 'A Study in Man-Machine Interaction', Proc, National Computer Conference, 46, AFIPS Press, Montvale, NJ, 1977.

39. Miller, Robert B., 'Response Time in Man-Computer Conversational Transactions', Proc. Spring Joint Computer Conference 1968, 33, AFIPS Press, NJ, pp. 267-277.

40. Nash, David R., 'Building EIS, A Utility for Decisions', Database, 8, Winter, 1977.

41. Newman, W.S. and Sprowll, R.F. Principles of Interactive Computer Graphics, McGraw-Hill, New York, 1979.

42. PLATO DSS Reference Manual, OR/MS Dialogue, New York, 1982.

43. RAMIS: Mathematica Inc., Princeton, New Jersey, 1982.

44. Remus, William, 'An Empirical Study of Graphical and Tabular Displays and Their Interaction with Demand Variability', Proc. 15th Hawaii International Conference on System Sciences, 1982.

45. Roberts, T.L., 'Evaluation of Computer Text Editors', Ph.D. Dissertation, Department of Computer Science, Stanford University, 1980.

46. Robertson, G., D. McCracken and A. Newell, 'The ZOG Approach to Man-Machine Communication', Int. Journal of Man-Machine Studies, 14, 1981, 461-488.

47. Savage, R.L., J.K. Habinek and T.W. Barnhart, 'The Design Simulation, and Evaluation of a Menu Driven User Interface', Proceedings Conference on Human Factors in Computer Systems, Gaithersburg, Maryland, March, 1982.

48. Schneider, M., 'Ergonomic Considerations in the Design of Control Languages', Proc. NYU Symposium on User Interfaces, New York, May, 1982.

49. Shneiderman, B., Software Psychology: Human Factors in Computer and Information Systems, Winthrop Publishers Inc., 1980.

50. Shneiderman, B. 'The Future of Interactive Systems and the Emergence of Direct Manipulation', Proc. NYU Symposium on User Interfaces, New York, May, 1982.

51. Simon, H.A., The New Science of Management Decisions, Harper and Row, New York, 1960.

52. SIMPLAN: Simplan Systems Inc., Chapel Hill, North Carolina, 1982.

53. Smith, C., C. Irby, R. Kimball, B. Verplank and E. Harsten, 'Designing the Star User Interface', Byte, 7, April, 1982, pp. 242-282.

54. Sprague, R.H., Jr. and E.D. Carlson, Building Effective Decision Support Systems, Prentice-Hal Inc., Englewood Cliffs, NJ, 1982.

55. Sprague, R.H., Jr. and H.J. Watson, 'A Decision Support System for Banks', OMEGA Vol. 4, pp. 657-671, 1976.

56. Stohr, E.A., J.A. Turner, Y. Vassiliou and N.H. White, 'Research in Natural Language Retrieval Systems', Proc. 15th Annual Hawaii International Conference on System Sciences, Hawaii, 1982.

57. Stohr, E.A. and N.H. White, 'Languages for Decision Support: An Overview' Working Paper #64, Graduate School of Business Administration, New York University, September, 1982.

58. Turner, J., M. Jarke, E.A. Stohr, Y. Vassiliou and N.H. White, 'Using Restricted Natural Languages for Data Retrieval: A Laboratory and Field Evaluation', Proc. NYU Symposium on User Interfaces, New York, May, 1982.

59. User Interfaces: Proceedings NYU Symposium on User Interfaces, Graduate School of Business Administration, New York University, May, 1982.

60. Vassiliou, Y. and M. Jarke, 'Query Languages: A Taxonomy' in Proc. NYU Symposium on User Interfaces, New York, 1982.

61. VISICALC: Software Arts Inc., Cambridge, Mass., 1982.

62. XSIM: Interactive Data Corporation, Waltham, Mass, 1982.

63. Wilkinson, Leland, 'An Experimental Evaluation of Multivariate Graphical Point Representations', Proceedings Human Factors in Computing Systems, Gaithersburg, Maryland, March, 1982.

64. Yonke, Martin D. and Norton R. Greenfield, 'An Information Presentation System for Decision Makers', Data Base, Vol. 12, Nos. 1 and 2, Fall, 1980.

65. Zloof, M., 'Query by Example: A Data Base Language', IBM Systems Journal, 4, 1977.

66. Zmud, Robert W., Individual Differences and MIS Success: A Review of the Empirical Literature, Management Science, 25, 1979.