IEEE Computer, Vol. 9 No. 9, Sept. 1976, 6-15

Minicomputers in Security Dealing

B.R. Gaines, P.V. Facey
University of Essex
J. Sams
Laurie, Milbank & Co.

Introduction

There are many business applications which fit far better the terminology and techniques of industrial control than they do that of conventional data processing. A dealer in any commodity whose price is continually fluctuating, be it money, stocks, or physical supplies, requires real-time information about the deals of his colleagues and competitors and the state of the market: he requires on-stream processing of this continually changing information against an on-line file system of historic data acquired about the market. His communication needs are best satisfied by interaction through visual displays even though his data processing requirements are those generally associated with-small batch-processing EDP machines.

In practice the industrial minicomputer has the right kinds of input-output facilities, real-time monitors, and interactive peripherals for these situations, and it is easier to make up its deficiencies in high-precision arithmetic, commercial languages, etc., than it is to make up the converse deficiencies of small EDP machines designed to be effective for batch processing. The very lack of some “commercial” operating system and language facilities on a minicomputer can be advantageous in dedicated systems where low cost and “tailoring” to a particular situation are prime requirements The meager, but adequate, run-time systems of most minicomputers contrast favorably with the massive memory occupancy and configuration requirements of typical EDP systems. Although these differences are historic rather than technical, reflecting different marketing and customer support requirements, they are still significant today in considering those business applications in which low-cost and/or rapid interaction are dominant factors.

This paper describes one successful application of a minicomputer in a business environment where the machine and its terminals have been closely integrated into an existing office and provide both an improved means of distributing information and also on-line analysis of that information in a manner not previously possible. The emphasis is on the role of a computer system in a real financial dealing environment and the technical and operational features of the project which make it possible. In particular the results obtained back up our previous arguments [1-5] that the interactive systems programming language BASYS provides a foundation for rapid development of interactive minicomputer systems tailored in detail to user requirements and that the problems of systems specification and developmental modification of computer-based systems are greatly eased if the interactive nature of the system is fully utilized for on-line development in parallel with actual user experience.

The actual system considered is a display-based inter-active minicomputer system for gilt-edged security broking installed in the London Stock Exchange in January 1972. The system is located in the same office as the 12 dealers it serves and is Continually updated with price information

over a radio link from the exchange floor. For each security, current yields and deviations from the general trend are calculated and displayed on television monitors. Graphic terminals enable dealers to request individual stock-by-stock comparisons, fitted trend lines, and other facilities for investment analysis. The financial aspects of dealing in British government securities and the data processing performed by this system have been described elsewhere. [4] Broad generalizations from our experience in developing and operating a variety of related interactive minicomputer systems in commercial, clinical, and industrial environments, have also been outlined elsewhere, [5] with particular emphasis on the design of man-computer dialogues. This paper focuses on a number of specific factors that have played important roles in the success of this particular application and which help to clarify the role of minicomputer-based interactive systems in commercial applications.

Requirements for a dealing information system

British government securities (gilt-edged stocks or gilts) currently total some £36,000,000,000 nominal value, and form a vital means of funding any residual budget deficit. Gilts are highly negotiable securities whose main characteristics are security of investment, wide range of coupon and redemption date, and very high marketability (each price quoted in the "large" market stands for transactions of at least £500,000. It is possible to deal, on good prices in normal times, in £5 million to £25 million.) The responsibility of the professional dealer is, therefore, financially heavy. He must have a full knowledge of the workings of the market and up-to-date price information, and be able to forecast the actions of other dealers and investors in the light of expected future events.

The physical dealing situation before a computer was installed was that 12 dealers and ancillary staff, each with a multi-line key-and-lamp telephone desk set, were in a small office grouped in clusters specializing in different types of stock. Price information was telephoned in from the exchange floor and written up on a roller-blind display in a maximally visible position. A list of prices and yields of all stocks, calculated from the previous nights closing position was circulated each day. Graphs of some bases for stock comparison, such as redemption-yield as a function of period to redemption, were drawn by hand. Desk calculators were available for the recalculation of yields as prices changed during the day. A timesharing service bureau was used once a day to give stock by stock comparisons based on the last three months' prices.

A partial justification of the utility of continuous access to a computer system is in the volume of price change information to be handled and the complex calculations (discounted cash flow) necessary to reduce this to a uniform basis for comparative assessment. The client relies on the dealer to have complete and accurate information about the current state of the market and the quality of this factual data can be improved by on-line computer processing. However, the system must be considered only as one more aid to dealing, since many of the relevant factors are also qualitative and psychological, personal judgments based on interpretation of information in the light of experience and on the evaluation of what judgments others are likely to be making. A market may move not only on the basis of a rational financial process but also on the expectation of how that process will affect the actions of others dealing in it.

However, in all cases it is advantageous to place such judgments in the context of financial facts, to be able to state that one stock gives a better yield than any other of comparable redemption period, to be able to state that the yield advantage of one stock over another is standing higher

today than it has over the past three months, and so on. Information cannot replace judgment but it is generally assumed that enhanced information leads to: enhanced judgment. This is the basic justification for the installation of a computer-based system: that the dealers have, and are known to have, more complete and up-to-date information presented in such a way that it highlights factors relevant to decisions on dealing.

Technical specification

As illustrated in Figure 1, the configuration consists of a 20K × 16-bit core PDP11/20; 256K × 16-bit fixed-head disk; two 145K × 16-bit magnetic tape units; 6 channels of 32-line × 48-character tv displays; 5 storage-tube graphic displays; two 30-cps teleprinters; and a 600 baud link to the Reuter's "Monitor" financial information network. The computer is installed in the same office as the dealers it serves. Figure 2 shows a typical dealing position with telephone facilities, a tv monitor and "channel-change" switches, and a graphic display and keyboard (on a turntable so that it can serve a group of dealers).

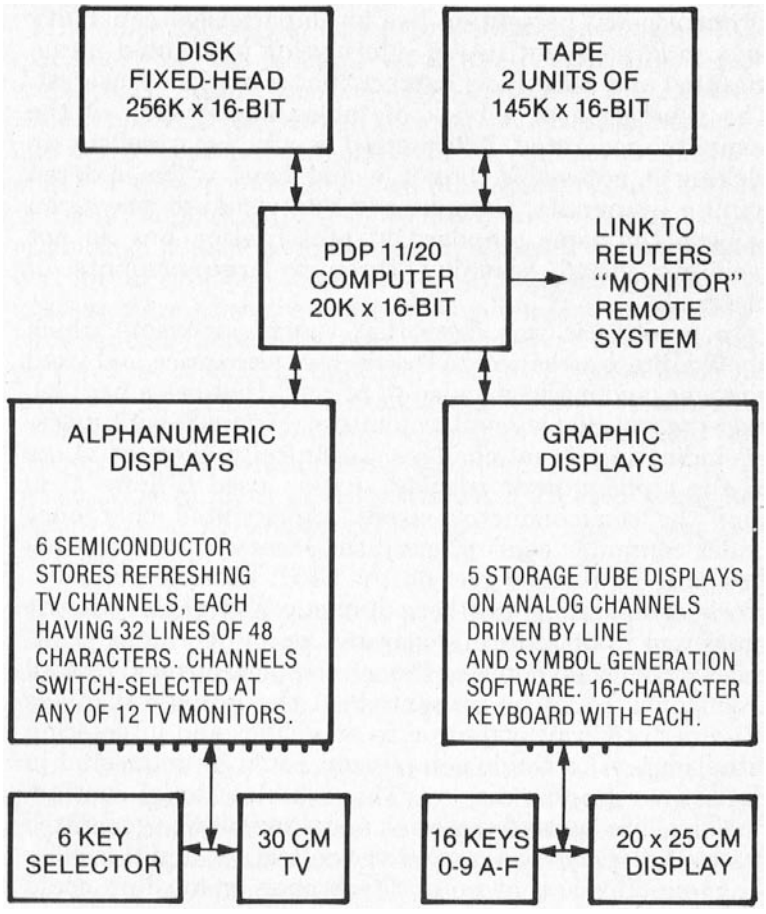


Figure 1. System configuration

Figure 2. A dealing station

The fixed information (interest rate, redemption date, etc.) on the 60 or so gilts at issue at any one time is held on the disk. The prices of each stock are the only independent dynamic variables, and price changes in all stocks are communicated over a radio link from the exchange floor to an operator at a teleprinter. He keys them in under an interactive data acquisition system which generates a variety of dialogues according to the type of stock and change made. A range of yields is calculated along with other functions such as maximum and minimum prices and yields of each stock over the day. The disk also retains "historic" daily records of the last three months' maximum and minimum prices and yields for all stocks. Each month the past month's historic records are stored on magnetic tape for long-term analysis.

The amount of raw information the computer holds is substantial, and the amount of processed information it can generate—for example, by comparing one stock with another—is potentially very great. Taking full advantage of this material is a difficult problem in its own right. Human visual perception has an unparalleled capability to scan large quantities of information (presented meaningfully) and selectively extract that which is of interest. The generation of a book of tables and graphs of the computer-generated information would be excellent in making it accessible, but it would have to be updated minute by minute. Computer-driven visual displays can achieve the same standard of presentation but do not facilitate visual scanning through large amounts of material.

In practice it was clear that the major items which should direct attention to themselves were price and yield changes, and it was decided to present these-as a parallel, ever-present display on tv monitors (Figure 3) with markers indicating recent changes. Technically, the advantage: of the alphanumeric display system used (Figure 1) is that the semiconductor stores are updated only once under computer control, and user access to the information puts no further load on the CPU. Historic stock-by-stock comparisons are less dynamic and could best be presented in book form generated as an overnight printout. Graphical synopses of these comparisons and overall trend lines could be presented on the graphic displays (Figure 4) to draw attention to anomalies and interesting situations. Any single comparison could be requested in detail on a graphic display. This combination of continuous displays, printed material, and dynamically generated selected displays has proved very effective, and it is clear from experience that no one technique or medium could satisfy all the differing requirements.

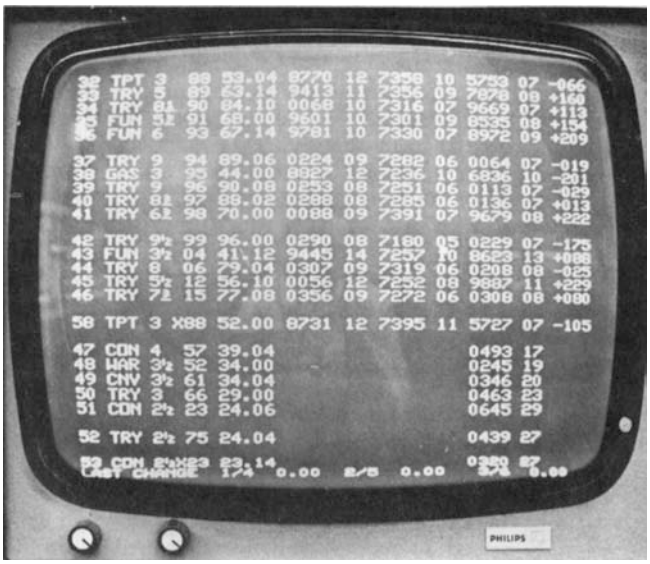


Figure 3. Television monitor yield display

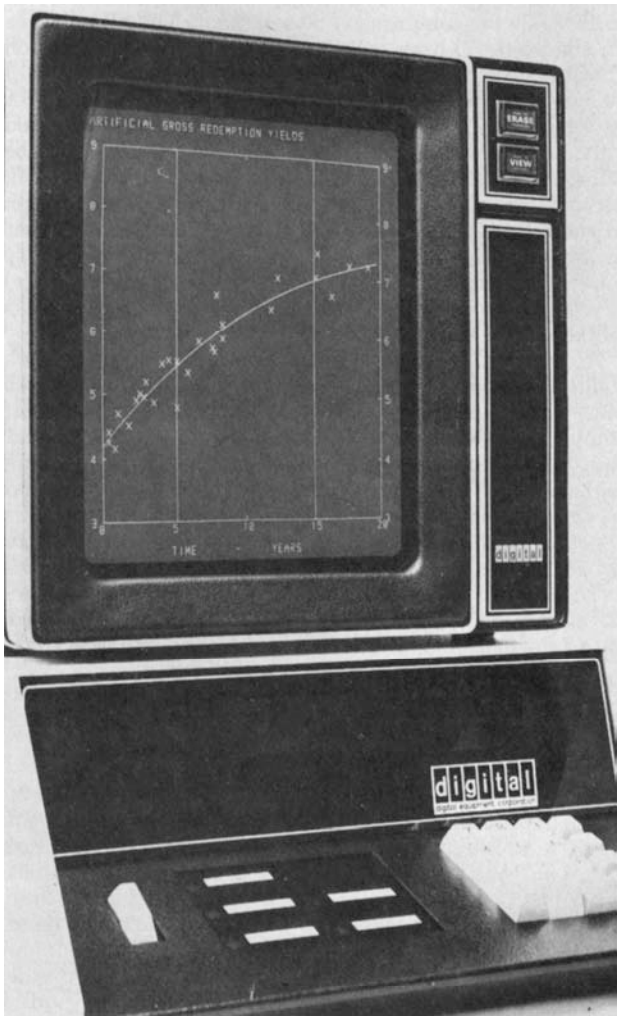


Figure 4. Graphic display of price ratios

In summary the dealer has, literally at his fingertips complete up-to-date figures of recent price changes, the current market prices, the associated yields, and deviations from the fitted curve of yields, for every stock. He is able to quote to clients in whatever detail is required the exact situation for every stock. In addition he can request at the graphics terminal a cross-comparison of two stocks he is discussing with a client and, within a few seconds, be able to quote comparative figures on the relative merits of the two stocks and their behavior over the past three months. On a longer term basis the dealer may browse through a wealth of data about stocks for purposes of investment analysis and to confirm, or refute, his appraisals of stocks based on other sources of information.

Human factors

The close integration of a computer system into an existing office environment clearly has its pitfalls and problems [6]:
(1) The office already exists and operates as an organic system. Communications and data processing methods have evolved over a long period that take no account of a computer. The current mode of operation is viable and will tend to have a robustness that rejects change, particularly disruptive change.
(2) There is no possibility of closing down existing procedures and starting afresh. The office must continue to operate with full effectiveness during the installation and on-site development of the computer system.
(3) No one person is in a position to explain the current operation of the office in detail. Advance systems analysis can only be regarded as an approximation to the actual requirements. It is a way of establishing a reasonable initial integration of the system into the office procedures and avoiding immediate rejection.

These considerations strongly indicate that the new system should be introduced slowly into the existing situation and evolved rapidly, on-site, in response to user criticism. There is one image of major help in visualizing the process—viz., the computer system is itself in the situation of a new person coming into the office and adjusting to its procedures. The new person will first have to settle down and adapt to the existing environment, to become accepted and trusted as a member of the community. Then if he has new functions to perform, they will also have to evolve within the existing framework as have previous procedures.

This animistic view of a computer system is one which we strongly recommend. Many problems can be avoided if they are examined within their social context: “If I were asking you to do this, or giving you information in this form, or responding to your requests in this way, how would you react?” Would I be prepared to do personally what I am programming the computer to do?” These have obvious connotations m programming the interactive dialogue with the computer which we shall consider in a later section. However, regarding the computer as if it were a new person coming into the office also has technical implications, e.g.:

(1) *Reliability*: a level of up-time completely adequate for throughput may be unacceptable in terms of user reaction. A colleague who was away with frequent illness would tend to be bypassed in routine matters even when he happened to be present. His services would not be fully utilized because his colleagues had learned to do without them. Thus even when the computational load in a system is small, the necessity of it being immediately available when required demands high reliability.

(2) *Response time*: reliability may be regarded as the coarsest factor determining response time. However, even when a system is available its responses must be fast enough to avoid user irritation. In particular, its response time pattern should match user conception of reasonable behavior. It is no use being very fast in one activity if another, which the user regards as basic and simple, is slow;

(3) *Rapid-reprogramming*: system reprogramming has to be rapidly achievable because users expect unreasonable behavior to be corrected when reported and alternative procedures to be demonstrated as discussed. Again, the new colleague who takes too long to respond to criticism and adapt his ways to those of the office will come to be regarded as a nuisance rather than a help. The plea so often received nowadays, "Sorry, this will now take six weeks instead of a few hours—we have a computer," is just not acceptable if a computer system is to be fully and rapidly integrated into an existing office.

Minicomputer-based interactive systems

The introduction has emphasized the importance of minicomputers to the development of effective interactive systems. It is clear that computer systems actually form a continuum from the smallest microcomputer to the largest machine. However, there has been a distinction in manufacture and marketing between the EDP machines and the laboratory/industrial minicomputers. This is a multivariate distinction in which one class of machines tends to be used for central systems providing a general service with emphasis on data processing, and the other class tends to be used for local systems dedicated to a specialist service with emphasis on input-output. Historically, the minicomputer is also associated with lower-cost systems and hence also with lack of software, particularly operating systems and high-level languages, since these tend to be expensive both to develop and to run (in terms of configuration requirements).

The unsuitability of batch processing on small EDP machines for the dealing situation has already been emphasized. Many workers have discussed the relative roles of man and computer in such decision-making situations and have hypothesized one sort of symbiotic relationship. [7-12] The computer's accurate memory and rapid numeric and logical process would couple with man's self-organizing, goal-directed strategies, semantic memory, and perceptual abilities to generate, through partnership, a system more effective than either alone. However, they have generally tended to think of the interactive computing provided by "computer utilities'' providing "conversational access'' [14] to "national information systems" [15] through "teleprocessing."[16] However, considerations, such as the "human factors" outlined in the previous section led us to believe that in many situations local, dedicated minicomputers are a more appropriate basis for man-computer systems.

There are a number of basic problems in the design, implementation, and application of interactive systems where the use of a number of minicomputers rather than one central system can give definite advantages:

(1) *Reliability.* A computer system is a complex of electronic and electromechanical modules, each of which must function for the system to be operational; the smaller the system the longer the mean time between failures. This factor is important in all systems, but it is vital to interactive systems where the degree of unreliability that merely causes a decrease in throughput to a batch-processing service results in an unacceptably low level of availability. Many

applications require a continuous 24-hour-per-day service, but even when a delay in access can be tolerated technically, it can be devastating to the remote interactive user community who are already isolated from the system and become totally so when it is not available. In particular they are faced with psychologically disturbing uncertainty in deciding whether to try to access the system again and face further rejection, or wait even though it may possibly have become available. It is a natural human reaction to attempt to avoid such a conflict situation, and it can be avoided by ceasing to use the computer-based system. Since many potential users have no particular enthusiasm for computers and in fact often feel quite the contrary, any such demotivating influence is a major problem.

(2) *Decentralization*. Failures themselves tend to be more catastrophic with computer systems than with conventional information management by clerks and files because, for the sake of efficiency and high workload, as much information as possible is centralized in the computer's files. Distribution of the workload and files of a large system between a number of independent modules increases its robustness.

(3) *Duplication*. The duplication of capability (rather than function) means that a system can operate at a full workload under normal conditions but shed load in a controlled manner when failures occur—e.g., files and work can be moved from one machine to another which has less important functions.

(4) *Low overhead of interaction*. Input/output is generally an expensive overhead on medium-size EDP machines with a single processor and large wordsize. The workload of interrupt-servicing and buffering ties up a powerful processor and expensive high-speed memory. Interactive data entry, which is of major importance in data management systems, is an expensive overhead on machines designed for batch number-crunching. Where the workload is not predominantly high-speed, high-precision arithmetic, but rather data acquisition, then a minicomputer is far better matched to the job.

(5) *Short response time*. This follows from (4) but is separated because of its importance in human terms. A minicomputer with its dense order code (with loss of address scope) in a timesharing system can maintain a number of small interactive jobs in core at reasonable cost and operate on a round-robin time-quantum of 100 msec, giving average response times of a fraction of a second. This makes the system far easier to use both for naive users who are disturbed by long pauses in response and also for skilled users who are frustrated by them.

(6) *Ease of operation and system management*. Most minicomputers and associated peripherals have been designed for industrial use and do not need special environments. The high-speed electromechanical peripherals on larger systems require the most operator care and attention. If the load is distributed to a number of typewriter (not teleprinter) terminals or visual displays, centralized operations can be minimized (basically only file security backup).

(7) *System design*. The most important feature has been kept to last: the prime cause of problems in most computer system development is the attempt to take on too much and to do everything in one major development. There is a crucial limit in system analysis at the point where the system as a whole is no longer completely comprehensible to one person. One task, one person, is a recipe for success (where a "task" is the design of a relatively independent part of a system). Starting with a small project with limited objectives on a small configuration and expanding with more projects, more configurations lead to rapid, robust progress. Systems tend to freeze with

time and successful use, and it is far easier to set up a new system than to increase the capabilities of an existent one.

These are clearly all potential advantages of the use of small, functionally complete systems. Any of them can be squandered by poor system design, and it is possible to point to large, well designed systems that are superior in all respects to far smaller systems of poorer design. There are also substantial advantages in large, centralized systems all stemming from the economies of size possible through sharing resources. However, where effective man-computer interaction is the prime objective, minicomputer-based systems have, potentially, major psychological and economic advantages.

Software requirements

We have emphasized the lack of software support for minicomputers, a weakness that is gradually disappearing as manufacturers develop software support for the commercial minicomputer market. However, it is easy to mimic both the facilities *and the defects* of classical EDP systems on minicomputers. Languages such as COBOL and RPG with their conventional libraries and operating systems are not suited to interactive dialogue processing. However, both the arithmetic facilities and string-handling of interactive languages such as standard BASIC are inadequate for many commercial applications.

The essential requirements in minicomputer support software for commercial applications are:

(1) *Fast response to terminal interaction* for reasons already noted.

(2) *Small size* for reasons of cost. It is ridiculous to commence with a low-cost minicomputer configuration adequate to the task and then have to expand it to support a large operating system and language run-time system. We advocate a minimal operating system for terminal buffering, round-robin job scheduling, and file access, closely integrated into an interpretive language, with less than 4K × 16-bit words each for operating system and language. Job partitions in the dealing system and other BASYS applications are typically under 2K × 16-bitwords.

(3) *High-level language* for ease of system development and modification. It is probably not necessary to justify this nowadays. In the dealing system it was essential because:

(a) The *system specified* was by no means complete, and the application was novel. It was clear that considerable on-site development and modification of the system would be necessary, and this would most conveniently take place while the system was in operation. Substantial reassembly, linking, and loading under these conditions would be difficult and, without hardware protection, would require the entire system to go down.

(b) The *time* for the complete development was very short, some 3 to 6 months, and a high-level language, where a substantial number of the facilities required were already provided, was attractive.

(c) It was desirable for the application software to be *taken over* eventually by the users. A suitable high-level language would make this feasible, whereas assembly language might make it impossible.

The usual argument against a high-level language is the overhead of its run-time system size and, perhaps, also its inflexibility in handling special peripherals and special data structures such as

strings. However, none of these are essential defects. We were fortunate in developing BASYS originally on a TSS8 [17] that allowed only 4K × 12-bit words for a complete user partition including interpreter. The full language interpreter occupied only 2.7K words including full access to monitor filing facilities, variable-precision integer arithmetic, string-handling, special peripheral driver (remote film-strip terminal), and interactive debugging and editing. There were no overlays. The 1.3K × 12-bit partition remaining for user programs and data structures in BASYS proved adequate to support medical record-keeping systems, psychological testing systems, advertisement booking systems, etc., each written as a set of comparatively short (100-line) overlays. This initial size constraint has carried over to later implementations of the language.

(4) *High-precision arithmetic* is uncommon in minicomputer systems and yet vital for commercial applications. Amounts of 100 million dollars or pounds kept to 1 cent or 1 penny require 10 digit precision. If integer arithmetic is used (which is desirable in accounting packages to give full control over rounding) then intermediate results in calculations may require greater precision. We have typically provided 1 through 7 × 16-bit word variable precision integer arithmetic giving up to 33 digit precision. The dealing system precision is set at 4 ×16-bit words in most sections giving some 18 digits.

(5) *Contextual string-processing* in a very general form is essential to the programming of natural dialogue with users—not natural language dialogue which is beyond current capabilities and requirements, but free-form communication with syntax and semantics natural to the user under his current office procedures. For example, in the dealing system it was essential that price information could be entered either in decimal or fractional form: 66.5 or 66½. A simple requirement that is problematic on many systems is that a space is a far easier separator to use than a comma (because of typewriter layout), and yet many languages do not allow the input field separator to be varied, or to satisfy a variable definition like "a comma or any number of spaces greater than zero." BASYS gives complete freedom to the programmer in contextual string analysis with facilities comparable in extent and power to those of SNOBOL [18].

(6) *Interactive programming* is not a mandatory requirement, but we have found it a major help not only in debugging but also in system design. It is possible to sit at one terminal of a system in normal use and demonstrate alternative dialogue sequences, output formats, etc., as part of a discussion with a user, and then implement their requirements—*all without affecting the system's ongoing operation*. This facility also has its dangers, of course, particularly if some users are remote and find the system literacy changing as they communicate with it! However, properly controlled, it is one of the most powerful features of an interactive, interpretive system.

For the dealing system a very comprehensive single-job disk operating system was available with the machine. It had two disadvantages in this application: (a) no extension to multi-job working, and (b) more elaborate facilities than actually required, leading to higher overheads than necessary. It was, however, considered highly desirable that system software development should be minimized and as much manufacturer's software as possible should be used, in order to ease maintenance problems by driving the peripherals with manufacturer's software, and to enable advantage to be taken of software drivers for new peripherals fitted to the system at future dates without excessive software re-engineering. Hence, a decision was taken to write drivers for the special peripherals under the standard DOS and implement a scheduler using the line-frequency clock to run a simple timesharing system apparently running as a single user to DOS.

An interpretive implementation of BASYS with variable-length integer arithmetic (up to 7 × 16-bit signed operands = 33 decimal digits) was chosen as suitable both for the text handling and the financial calculations. This was implemented on the PDP 11 to give a single-language timesharing system with job slots for each graphic terminal and for the typewriter. To maximize speed no automatic swapping was incorporated and all jobs are locked in core. However, the core avocations are completely flexible and dynamically adjusted automatically as application program overlays are caned. Semaphores are used to overcome potential clashes in main memory requirements. Application program overlay caning is extremely simple in BASYS fall simple variables and whatever arrays and strings are specified are passed to the called program), and our previous experience had shown that there were advantages in writing such systems in a modular fashion as a succession of comparatively small BASYS overlays.

BASYS retains the structure of BASIC in that a program is a sequence of numbered lines ordered by their (not necessarily consecutive) line numbers, and statements consist of a meaningful keyword followed by an expression, or sequence of expressions, e.g.:

```
25    LET Y = 9
40    PRINT 'Y IS' Y
150   INPUT U V W
260   PRINT U*V W*100
500   GOTO 150
```

The meaningful keywords seemed to contribute much to BASIC's high readability and the need to insert them itself prevents the vast, opaque syntactic constructions possible in ALGOL and mandatory in LISP.

The role of the line numbers and their use in transfers of control are more open to controversy. However, the situation of a programmer at a terminal is that he cannot see all his text at a time but must access individual lines. Numeric labels corresponding to the order of statements give an immediate picture of the stored program structure. Using the same line numbers that make for ease of editing as the "labels" for transfer of control is then a natural construct necessitating the acquisition of no new concepts. We accepted the use of line numbers in BASIC and have not regretted it. Indeed in BASYS it is extended to enable dynamically varying character strings and string constants to be stored and referenced as part of the same structure of numbered "program" lines.

We wished to minimize the effort of program creation and documentation and maximize the clarity of the result by having (a) no unnecessary syntax on program entry— the programmer should be able to enter the minimum string necessary to specify a statement; (b) full clarity in program listings—the system should recreate missing syntax on output and format it appropriately. To achieve this we had to drop some BASIC conventions, notably the non-significance of spaces which were natural separators, readily inserted with the space bar. This allowed a comma, or one or more spaces, or implicit separation, to be specified as optional separators. Command keywords could then be defined as a string of letters which matched, or partially matched, one of the stored commands. Hence any command could be shortened to its minimal unambiguous initial string. We chose command names so that the first two letters were sufficient to resolve ambiguity; and a single letter alone, if ambiguous, was interpreted as the most commonly used command—e.g.,

25L Y = 9
500G150

are abbreviated forms of the first and last statements in the example above. When the program is listed they will appear in the full form of this example.

Figure 5 gives an example of actual program creation at a terminal. If even a partial match is not possible (first entry of line 120), an immediate error message is generated with an exclamation mark, !, at the offending point. If a statement is typed in terminated by ESCAPE (rather than NEWLINE), it is printed out again immediately in full form. Note how L alone becomes LET but LI is interpreted as LIST. Note also that commas are not necessary as separators in line 100.

```
>10P'SQUARES & CUBES
>60L X=0
>100P X X*X X*X*X
>120LX=X+1
ERROR C
 120 LX!=X+1
>[I'LL TYPE IT AGAIN ENDING
>[WITH ESC TO GET PRINTOUT
>120L X=X+1
 120 LET X=X+1
>150IFX<10:G100
>LI

  10 PRINT 'SQUARES & CUBES
  60 LET X=0
 100 PRINT X X*X X*X*X
 120 LET X=X+1
 150 IF X<10 :GOTO 100

>
```

Figure 5. Interactive entry of a BASYS program in brief form and full generation on listing

Line 150 of Figure 5 appears like the normal conditional of BASIC. However, it is more general in that any sequence of commands may be placed on one line, separated by colons, to form a single statement—e.g.,

100 PRINT X X*X X*X*X: LET X=X+ 1: IF X< 10: LOOP

is completely equivalent to lines 100 through 150 of Figure 5. BASYS is a two-dimensional language in which execution continues along each line, then transfers to the next. Conditional tests are regarded as decisions to continue execution of the current line or to go straight to the next line. Advantage is taken of this feature to generate implied conditionals in many other statements, particularly the input-output statements which may fail for good reason (end-of-file), and the pattern-matching string operations described later—e.g., the sequence

100 OPEN ‘FRED’: GOTO 120
110 PRINT ‘CANNOT FIND FRED’: STOP
120

or better,

100 OPEN ‘FRED’
110 ELSE: PRINT ‘CANNOT FIND FRED’: STOP

each use the implied conditional in the file OPEN command to test its success.

Since BASYS is executed interpretively it is simple to provide for all commands to be directly executed at any time. When programs are halted by use of a control key, or stop with an error message, the data structures are available and may be examined and changed and then execution continued. Figure 6 illustrates a sequence of interaction in which the program is changed after an error and execution continued (the X command is a line editing facility). This example also illustrates the use of a bracket as a (non-executable) "command" to indicate a comment and the availability of UNLESS as an alternative to IF.

```
>1[CREATED 19/7/70 AS EXAMPLE
>50P'DEMO
>90IN X Y
>110P X X*X X*X*X
>120L X=X+1:UN X>Z:G110
>LI

   1 [CREATED 19/7/70 AS EXAMPLE
  50 PRINT 'DEMO
  90 INPUT X Y
 110 PRINT X X*X X*X*X
 120 LET X=X+1 :UNLESS X>Z :GOTO 110

>RUN
DEMO
:4 7
     4     16     64
ERROR U  AT 120
 120 LET X=X+1 :UNLESS X>Z! :GOTO 110
>X120/Z/Y
 120 LET X=X+1 :UNLESS X>Y :GOTO 110
>G110
     5     25    125
     6     36    216
     7     49    343
>P X Y
     7      7
>
```

Figure 6. Interactive debugging and editing of a BASYS program

String processing in BASYS. Our interpretive implementation of BASYS automatically gave us facilities for manipulating the character strings forming program lines, and it was natural to store string data in the same way. Any "program" line beginning with a $ sign is a string variable initially containing whatever follows the $ sign. Such variables are referenced as $<line number>, where "line number" is a number, or numeric expression evaluating to a line number—e.g.,

```
>100 $HI THERE
>PRINT $100
HI THERE
>LET X=99
>P$X+1
HI THERE
>
```

Apart from its implementation advantage, this mechanism for string variables (a) enables string constants to be listed as part of the program, (b) enables string constants to be placed in that part

of the program where they are used, and (c) makes string arrays naturally available and, in particular, efficiently implements sparse string arrays.

String expressions containing string constants, variables and literals, and numeric variables converted to strings have a natural syntax—e.g.,

```
>PRINT $100 'WHAT IS' X 'TIMES' X
HI THERE WHAT IS 99 TIMES 99
```

Contextual string analysis is based on the concepts of a sow string being analyzed, a destination string to which output may be appended, and various pattern-matching commands. System variables contain the line numbers of the source and destination strings and a pointer to a character within the source string. These system variables are automatically set up by the string processing commands but are accessible to the programmer for the more complex string processing.

The command PUT <string expression> sets up a source string by assigning the value of the expression to it and setting the character pointer to zero. The command AS <string variable> sets-up the string variable as a destination string initially null. The command WITH < string expression > is used to append the value of the expression to the destination string, typically to replace a pattern matched in the source string.

TO, FROM, and SEEK are template search commands specifying a template to be looked for in the source string, starting from the character pointer. If the pattern is found, then the pointer is updated to the next character beyond it. If the template does not fit and the match fails, then neither the pointer nor destination string is affected. FROM specifies an anchored search (for an initial template) and SEEK an unanchored search (for an imbedded template). TO specifies an unanchored search in which characters in the source string prior to those matching the template are appended to the destination string. The forms of template include string variables, numeric

strings(automatically converted and assigned to a numeric variable), and a specified number of characters. The commands and templates are powerful enough to cover most requirements, but enable the string analysis to be expressed simply and meaningfully—e.g.,

```
>1 $ABCDEFGH
>PUT $1: AS $2: FROM 'CD' :TO 'G'
>PRINT $2
EF
```

Figure 7 gives some more detailed examples, showing how a date format is matched and the numeric fields extracted, and how a simple sentence can be split up and reconstructed as a reply. Figure 8 is a simple "calculator" program with its own syntax that illustrates the use of the string processing to analyze a formal "language."

```
>[NOW HOW TO RECOGNIZE A DATE
>10 $DATE:
>100 INPUT ?10 :SEEK DA'/'MO'/'YR
>120 ELSE :PRINT 'WHAT ? ' $0 ' - EG 3/7/73' :GOTO 100
>140 PRINT DA MO YR
>RUN
DATE:5 JUNE
WHAT ? 5 JUNE - EG 3/7/73
DATE:5/6/73
     5     6     73
>
>[NOW A SENTENCE SPLITTING
>10 $WHAT DO YOU LIKE BEST ? :
>100 INPUT ?10 :AS $1 :SEEK 'LIKE' :TO %
>120 ELSE :SEEK 'LOVE' :TO %
>140 ELSE :PRINT 'THAT'S NICE' :GOTO 100
>160 PRINT 'FUNNY, I LIKE ' $1 ' AS WELL' :GOTO 100
>RUN
WHAT DO YOU LIKE BEST ? :I LOVE CHICKENS
FUNNY, I LIKE CHICKENS AS WELL
WHAT DO YOU LIKE BEST ? :GO AWAY
THAT'S NICE
WHAT DO YOU LIKE BEST ? :
```

Figure 7 Demonstration of the string-processing commands in BASYS

```
>25 $COMMAND:
>100 INPUT ?25
>120 IF $0 '?' :PRINT 'OPTIONS ARE: ',
>130 AND :PRINT 'ADD, MUL, DIV - EG ADD 5 AND 6' :GOTO 100
>140 SEEK 'ADD' X Y :PRINT X+Y :GOTO 100
>160 SEEK 'MUL' X Y :PRINT X*Y :GOTO 100
>180 SEEK 'DIV' X Y :PRINT X/Y '    REMAINDER' QA:GOTO 100
>200 PRINT 'CANNOT GET ' $0 :GOTO 130
>RUN
COMMAND: WHAT SHALL I DO ?
OPTIONS ARE: ADD, MUL, DIV - EG ADD 5 AND 6
COMMAND:MULTIPLY 9 TIMES 7
    63
COMMAND:ADD 5 & -5
     0
COMMAND:PLEASE CAN YOU DIVIDE 9 BY 4
     2    REMAINDER     1
COMMAND:1+2
CANNOT GET 1+2
ADD, MUL, DIV - EG ADD 5 AND 6
COMMAND:
```

Figure 8 A simple calculator language implemented using BASYS pattern matching

These examples demonstrate the operation of the pattern matching string analysis facilities in BASYS in simple situations. The naturalness of this command family should be apparent for use in the commonly required string analysis needed to support interactive data entry and conversational dialogues. Our objective was not only to supply string-processing facilities but also to retain the readability, and the transparency of function, which is such an important feature of BASIC.

BASYS has other extensions designed to improve the programming environment and increase modularity: names can be multiple characters, parameters can be passed to subroutines either by value or by reference; variables and arrays in subroutines may be declared local, and so on. It is interesting to note that the majority of these "additional" features required remarkably little extra code. Often they actually reduced the code because simplicity and generality for the programmer seem to correspond to similar features in the implementation.

The implementation of BASYS is straightforward using a "bottom-up" approach. First one writes a family of low level routines to create a virtual machine with character-handling and the required form of arithmetic. Then one writes a set of intermediate routines that manipulate the BASYS data structures. Finally one writes the top-level routines that evaluate string and arithmetic expressions. The time taken to do this is about two man-weeks on a machine with a well-integrated, disk-based machine code development system. As we have emphasized previously [2] it is the quality of these developmental resources which has the prime effect on the software development time for well-defined system programs. We have found it worthwhile to write a good assembler and linking loader for a minicomputer before attempting any other software development if its facilities are primitive. We have found it equally vital to base the development on the best interactive text editor available rather than any other resource. An emulator of one machine on a more powerful, better supported, disk-based configuration is useful, but a good editor is vital.

Conclusions

We have attempted to give an appraisal of the role of minicomputers in real time commercial applications based on our experience with a system for security dealers. Hardware, software, and human factors all play an important part in system requirements and success. It is not possible in the- space of this paper to demonstrate how they come together in detail, particularly in the programming of interactive dialogue. A fairly detailed exposition of some of the techniques we have developed to give maximum support to both untrained and skilled users of the system appears elsewhere [5].

A good example of the way in which the string-handling facilities of BASYS can be used to adapt the computer system to the user is the "shorts language" developed for input and display of prices of the short stocks. These prices are communicated in a dealer's jargon which has various short forms for common situations and, on analysis, turns out to be consistent, unambiguous, and less prone to errors in verbal transmission than number strings. Briefly: only the buying and selling fractions are communicated (the number of whole pounds—the "big figure" is Obvious with short stocks)- sixteenths are referred to by the numerator only—5 is 5/16; "under" means less 1/32, "close under" means less 1/64—these were written, e.g., U3 (=5/32t, CU3 (=11/64; similarly for "over" and "close over";:"either side" means a price range from −1/32 to +1/32 about a center price. This was written, e.g., −7−(=13/32 to 15/32); "close to close" meant +1/64, e.g., 1/2 CC (=31/64 to 33/64). The "language" is used for both price input and display, and there has been highly favorable reaction to it by dealers; it is a "natural" language.

A good example of the modularity and extensibility of BASYS is the link to the Reuter's Monitor computer system shown in Figure 1. It was not m the original design and was added some two years after the main system had been in full operation. Price and yield changes of gilt-edged stocks are automatically transmitted to Reuters as they are entered in the dealing system.

The communications module is written as an independent BASYS program that waits on a semaphore for a price change and then automatically transmits it in the required protocol using the string-handling facilities. The same BASYS program also responds asynchronously to operator keyboard requests to send other information to Reuters.

One crucial feature of any information management system is the quality of its data. Here the compactness of the dealing system is a distinct advantage. As prices are typed in they are immediately displayed; any errors are noticed and shouted to the operator. On a system with a very much larger data base there would be serious problems of accuracy that would be very difficult to overcome. Our experience has been that, in dealing, all "rules" are eventually broken. Prices do change in amounts and patterns that had been stated to be "impossible," so that building "intelligence" into data entry routines can be dangerous.

Over the past five years BASYS has been implemented on a wide range of machines (PDP 8, PDP 9, PDP 10, PDP 11, Prime 100, and MINIC I) and used in diverse applications including hospital administration and foreign exchange dealing in a number of European financial centers (where dual PDP 11 configurations have typically replaced very much larger EDP machines). Current developments include both expansion to very much larger configurations (64K words upwards) that are really no longer "minicomputers," and compression to the microprocessor plus floppy disk configurations now becoming available. Technically the configuration of a microprocessor within a VDU with floppy disk backing store and 16K × 8-bit byte RAM for BASYS and user program partitions is very attractive for many interactive EDP tasks. We are currently developing the communication protocols for networking such configurations to give a highly robust, cost-effective, and expandable system.

The main defect of BASYS in current implementations is its relative slowness (compared with Fortran, say) due to the interpretive execution. The 10:1 speed drop has proved surprisingly insignificant in many applications. Even in a university environment, when the language has been used for modeling and simulation, signal processing, pattern recognition, and so on, the benefits of the language facilities seem to outweigh the loss of speed. However, particularly with current MOS microprocessors, speed can be a significant limitation and in the most recent implementation (for an LSI 11 under RT 11) we are experimenting with BASYS/Fortran compatibility. BASYS there has the same data types as PDP 11 Fortran (16-bit integer, 32-bit real, 64-bit double) and can call Fortran subprograms. Thus for any particular application a library of Fortran subprograms can be installed as routines called by new BASYS commands or functions. This gives specialized, high-speed features while retaining all the advantages of an overall interpreter. Only application experience can validate this approach but its modularity, flexibility, and interactive features make it prima facie a very attractive software architecture for small commercial systems.

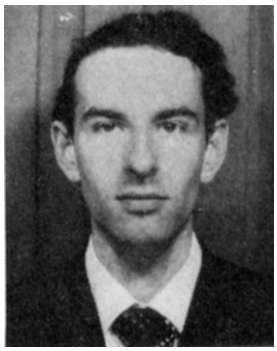
The system described in this paper has been in full-time operation since January 1973 and has come to be accepted as part of the normal facilities of dealing. It demonstrates that a display-based, on-line minicomputer can be a very effective tool in a commercial environment and that the developmental effort required is reasonable in magnitude. Technically the project is a further illustration of the power of a high-level interpretive language on a minicomputer in providing a flexible, rapidly-adjusted system that can be closely matched to the terminology and requirements of its users.

References

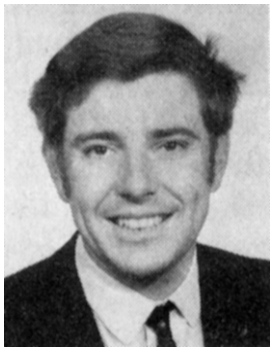
1. BASYS User Manual, Dept. Electrical Engineering Science, University of Essex, Colchester, England, 1974.
2. P. V. Facey and B. R. Gaines, "Real-Time System Design under an Emulator Embedded in a High-Level Language," Proc. Brit. Comput. Soc., DATAFAIR 73, pp.285-291, April 1973.
3. T. C. S. Kennedy and P. V. Facey, "Experience with a Minicomputer-Based Hospital Administration System," Int. Journal Man-Machine Studies, Vol. 5, pp. 237-266, April 1973.
4. B. R. Gaines, P. V. Facey, and J. Sams, "An On-Line Fixed Interest Investment Analysis and Dealing System," in Proc. Eur. Compuhng Congress (EUROCOMP 74), pp.155-169, May 1974.
5. B. R. Gaines and P. V. Facey, "Some Experience in Interactive System Development and Application," Proc. IEEE, Vol. 63, pp. 894-911, June 1975.
6. K. London, The People Side of Systems, McGraw-Hill, London, 1976.
7. J. C. R. Licklider, "Man-Computer Symbiosis," IRE-THFE, Vol. HFE-1 pp. 4-11, March 1960.
8. L. Press, "Towards Balanced Man-Machine Systems," Int Journal Man-Machine Studies, Vol. 3, pp. 61-73, January 1971.
9. A. M. Hormann, "A Man-Machine Synergistic Approach to Planning and Creative Problem-Solving," Int. Journal Man-Machine Studies, Vol. 3, pp. 167-184 (Pt. I), pp. 241-267 (Pt. II), 1971.
10. H. T. Smith and R. G. Crabtree, "Interactive Planning: a Study of Computer Aiding in the Execution of a Simulated Scheduling Task," Int. Journal Man-Machine Studies, Vol. 7, pp. 213-231, March 1975.
11. W. Edwards, L. D. Phillips, W. L. Hays, and B. G. Goodman, "Probabilistic Information Processing Systems: Design and Evaluation," IEEE-TSSC, Vol. SSC-4, pp. 248265, September 1968.
12. D. J. Hall, G. H. Ball, D. E. Wolf, and J. W. Eusebio, PROMENADE An Improved Interactive Graphics Man-Machine System for Pattern Recognition, Final Rep. SRI Project 6737, Stanford Research Institute, 1968.
13. D. F. Parkhill, The Challenge of the Computer Utility, Addison-Wesley, Massachusetts, 1966.
14. W. D. Orr (ed.), Conversational Computers, J. Wiley, New York, 1968.
15. M. Rubinoff (ed.), Towards a National Information System, Spartan, Washington, 1965.
16. H. Lui and D. Holmes, "Teleprocessing Systems Software for a Large Corporation Information System," AFIPS Conf. Proc., SJCC, Vol. 36, pp. 697-709, 1970.
17. A. van de Goor, C. G. Bell, and D. A. Whitcraft, "Design and Implementation of TSS/8: a PDP8 Based TimeSharing System," IEEE-TC, Vol. C-18, pp. 1038-1043, November 1969.
18. R. E. Griswold, The Macro Implementation of SNOBOL 4, Freeman, San Francisco, 1972.

Contributors

Brian R. Gaines is chairman of the Electrical Engineering Science Department at the University of Essex, Essex, England, where he first became a faculty member in 1967. He is also the editor of the *International Journal of Man-Machine Studies* and an associate editor of the IFAC journal, *Automatica.* His prime research interests are the interplay of computer languages, operating systems and architecture, algebraic system theory, and human factors in systems engineering. He has directed research contracts with the Ministry of Technology, the Department of Health and Social Security, and the National Research and Development Council and has been a consultant to several industrial firms, including ITT, Plessey, and Brown Boveri Kent. Gaines has authored approximately 50 papers on topics including stochastic computing, computer design, interactive computing, human operator studies, machine learning, and adaptive control. He studied mathematics and psychology at Trinity College, Cambridge, England, where he obtained the BA, MA, and Ph.D. He is a-member of the IEEE, the ACM, the British Psychological Society, and the Experimental Psychology Society.



Peter V. Facey is the senior research officer in the Electrical Engineering Science Department of the University of Essex, Essex, England, where he has worked since graduating with a degree in physics from Brasenose College, Oxford, England, in 1968. His prime research interests are concerned with the structure of computationally efficient and humanly effective software/hardware systems. He has been responsible for a number of research contracts on computer systems development with the Ministry of Defense, the Department of Health and Social Security, and the National Research and Development Council; and has been a consultant to a number of companies concerned with software and system development. Facey is a member of the ACM.



Jonathan B. S. Sams is an associate of Laurie, Milbank, and Company and a member of the London Stock Exchange. His special field is investment analysis of the British gilt-edged securities market. He worked on the English Electric DEUCE system in 1960-1961 and qualified as Fellow of the Institute of Actuaries in 1973. Sams received the BA degree and the MA degree in mathematics at Cambridge University in 1965 and 1969, respectively.