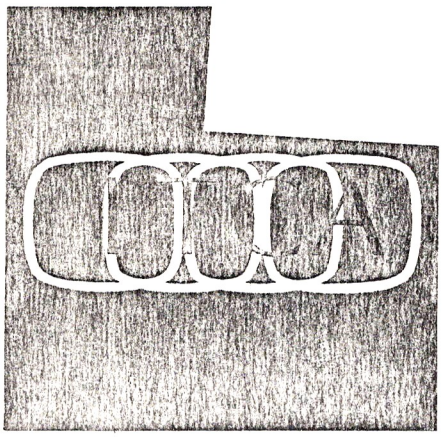
mel



RESEARCH

Software for Telesoftware

Report 122/79

INDEPENDENT TELEVISION COMPANIES ASSOCIATION LIMITED
Knighton House, Mortimer Street, London W1N 8AN, United Kingdom.

CONTENTS

1. INTRODUCTION

The purpose of this study is:—

i) To define the specific requirements for a computer language to be implemented cost-effectively by a Broadcast Telesoftware system, and to investigate consequent hardware constraints on terminal design.

ii) To investigate the suitability of existing computer languages for such a purpose, taking into account the possible range of applications for such a system in a domestic environment.

The study first of all examines likely areas of application of such a system and from this suggests possible terminal configurations. A close comparison is made with the view-data telesoftware system proposed by the British Post Office and fundamental differences are highlighted.

A comprehensive exposition of a suitable Broadcast Telesoftware system is then given, from which the desirability of having not one but two languages, for use in different parts of the system, becomes apparent. The requirements of these languages are then given and suitable languages are chosen from the variety available.

Having thus given a description of a suitable system and chosen languages to suit it, the discussion then centres on the possibility of standardisation of some or all aspects of telesoftware with the British Post Office.

Conclusions drawn from all parts of this study are presented and some recommendations based on these conclusions are offered.

2. APPLICATIONS OF BROADCAST TELESOFTWARE

Of course, it is impossible to predict all of the applications for Telesoftware, but several groups of applications can be identified, and these groups will hopefully highlight the fundamental requirements which future applications will demand of the telesoftware system. It is, of course, vital that the system when initially specified does not seriously inhibit the applications of the system in the future. A careful line has to be drawn between flexibility and economics. An ultra-flexible system would tend to be uneconomic in the early years, perhaps inhibiting its growth, while an inflexible system could be extremely economic in its early days, leading to rapid growth, but would then become restrictive as applications become more advanced, and hardware becomes ever cheaper.

Application Groups

2.1 Self-assessment

Typical examples of applications in this group are mortgage calculations, tax calculations and welfare rights examinations.

The characteristics of this group are that they are of a question-and-answer nature, with numerical and logical calculations to be performed by the program upon data (both numeric, logical and textual) provided by the user. Although uses such as mortgage and tax calculations would generally only require numeric (e.g. monetary values and time scales) and logical (e.g. options) data from the user, other uses, such as welfare rights may require the use of textual data. To elaborate:- it would be possible to show all of the acceptable textual data on the screen as a numbered list, and the user then be requested to input the appropriate number, but a more acceptable solution in some cases may be the provision of an alpha-numeric keyboard by which textual data could be input to the program for analysis. The second approach, although requiring more hardware (the alphanumeric keyboard) and apparently more effort on the part of the user (more keystrokes) has the advantage of not inhibiting the response of the user as much as the first approach, and does, with the provision of a suitably designed program, offer a much more "friendly" appearance to the user.

Another characteristic of this group is the amount of textual information held in the program. Generally, in terms of the amount of storage required for program instructions and data, the amount required for data will predominate, often by a factor of several times. The program instructions will be concerned with i) outputting appropriate text, ii) analysing the responses from the user, and iii) performing the calculations implied by these responses. All of these, with the possible exception of ii) which could become quite involved if an intricate user-interface were required, do not involve that many program instructions, whereas the text required, even in the modest programs of this nature which have been tried out on the experimental telesoftware system, has been most substantial.

The use of graphics by programs in this group is a distinct possibility (e.g. for the display of graphs by financial programs) and this tendency will dramatically increase the amount of "textual" information in the program.

2.2 Education

Typical examples of applications in this group are i) language and literacy teaching, ii) mathematics and numeracy teaching, iii) scientific and technical areas, iv) non-numerical areas like history and general knowledge, and v) "trade" areas such as first-aid and cookery.

The use of telesoftware to improve literacy and to teach languages will be distinguished by the large amounts of text to be incorporated into the programs, as well as the

availability of a full alphanumeric keyboard attached to the terminal.

The teaching of numeracy can be seen as one of the more obvious uses of telesoftware, and will not require more than a minimal telesoftware terminal with a numeric keypad attached, although special keyboards (e.g. large size) may be required in certain circumstances, such as teaching the physically handicapped.

The teaching of mathematics, can, in the more simple cases, be regarded in a similar manner to the teaching of numeracy, but the more advanced aspects of mathematics will require more sophisticated facilities. For example the teaching of algebra would require the addition of an alphanumeric keyboard. (It is apparent that, with the severely limited graphics available on teletext, areas such as geometry and trigonometry would not be suitable subjects for telesoftware). A substantial amount of text will be required for the teaching of any topic other than areas like basic numeracy and in most cases will dominate the amount of storage required in the terminal.

The teaching of scientific and technical topics with telesoftware is similar in requirements to that of mathematics, but, in general will again be severely hampered by the very poor graphics facilities offered by teletext.

The teaching of non-numeric subjects will require possibly the largest quantities of text to be held in the telesoftware terminal, and will also require an alphanumeric keyboard. Although the amount of numeric calculations to be performed by such programs may be small, considerable computing may be involved in text manipulation and analysis.

The teaching of "trade" subjects such as first aid has already been demonstrated as being feasible with telesoftware with the minimum terminal requirements. These topics lend themselves to a wide range of program and terminal complexities.

2.3 Games

Games programs tend to fall within two subgroups:- i) verbal and reasoning games, such as Mastermind, and ii) "dexterity" games such as ping-pong and Star Wars.

The first of these subgroups is ideally suited to telesoftware, since it can in general be accommodated within a minimal terminal, and does not generally require anything other than the standard numeric keypad for inputting data, or large amounts of program instructions or text to be held in the terminal. However, when games such as chess are considered, the requirements change considerably. The minimum graphics resolution necessary to present an acceptable representation of a chess board is 128 x 128, whereas the maximum teletext resolution is 72 x 80, so that considerable difficulty will be experienced in this respect, without even considering the need for textual information to be presented at the same time (e.g. the score, which would have to be given as an alternative presentation to the chess board).

"Dexterity" games present a different set of requirements. Since they would generally be constructed to suit the system and not be constrained to some other historical

format, the limited resolution of teletext would not be a fundamental limitation but only a limitation on the complexity of the visual presentation that can be employed.

The area of "dexterity" games raises some requirements, which, although of possible value in the other application areas, have not yet been touched upon. The most obvious is of course the need for paddles, joysticks or whatever other control mechanism suits the particular game. These are essentially "analogue" devices in that they are used by the game player(s) to indicate to the terminal the required position of some item on the screen. Some associated device is used as a "hit" mechanism, usually just a push button.

This type of game introduces into the program running in the terminal the need for timing. This might be merely for controlling the rate at which the action on the screen is carried out, or for more overt reasons, such as giving the player a limited time to perform a particular action. In the former case "real" time is not required, but in the latter case there will be applications which will demand the measurement of "real" time to an accuracy of at least one quarter of a second (thus ruling out the use of the broadcast teletext time for this purpose).

A distinction between games which are between the terminal and player and those which are between two (or more) players with the terminal "managing" the game should be made. In the first case, a player strategy as well as the means for managing the game must form the program in the terminal, whereas, if the terminal is merely managing the game, no such strategy is required but several control devices (one for each player) will have to be provided. This concept of machine-managed games is not limited to dexterity games of course, and can be applied to several other applications of telesoftware.

2.4 Database Broadcasting

An example of this group is "credit card fraud countermeasures". In this application a telesoftware terminal is located near points of sale in stores and shops. A small telesoftware program is loaded into each terminal at the start of each working day (or it could be permanently fixed into the terminal) and throughout the day a list of stolen or lost credit cards is transmitted (effectively as pages of this program) at intervals and is used by the program to update the list held within the terminal. The shop-keeper, when presented with a "dubious" credit card, keys its number into the terminal. The program then tries to match this number with one from the latest broadcast list. If it succeeds then the shop-keeper is advised by the terminal to take preventative measures, otherwise the transaction proceeds normally. This example illustrates the technique of a broadcast "data base", which is particularly relevant when the amount of data is fairly small but is subject to frequent updating. Depending on the application, all or part of the database can be held in the terminal after it has been broadcast, thus leading to fast searches through the data. Alternatively, when a search is initiated by the user, the terminal could wait for the data to be broadcast before examining it.

This also raises the possibility of "closed user groups" where the data to be transmitted

would be of interest to only a limited number of users but it is assumed that techniques to render the data unusable to other users would be contrary to the franchises of the various Independent Television companies.

2.5 Home Programming

The telesoftware system should be specifically designed for the broadcast of useful and reliable programs. It should *not* be designed to permit the user to program the terminal himself since this provision requires such different features as to render the system much more complex and alters its characteristics considerably. This argument will be elaborated later when a comparison with the system proposed by the British Post Office is made.

3. PROGRAM RELIABILITY

Since the telesoftware system is specifically designed for use by members of the general public with no knowledge of computing whatsoever, it is most important that the programs executed by the telesoftware terminals exhibit the utmost reliability. It is necessary for the program instructions (but not so vital, in some circumstances, for the textual part of programs) to be protected in transmission by error detecting and error recovery techniques such that the possibility of the terminal executing an incorrectly received program is very small. However, investigations into such methods do not form part of this study, and it is assumed that the program to be executed by the terminal is identical to that conceived by the programmer. It is also assumed here that the terminal itself is functioning correctly at the time that it comes to execute the received program. A possible means of ensuring that this is so, would be for the control program in the terminal to enter a terminal-check mode when first switched on, and to continue with the normal mode of operation only if all has been found to be well. If a malfunction is detected an attempt could be made to put a suitable message onto the screen to inhibit further telesoftware action.

The problem of concern in this study, then, is that of ensuring that the program as specified by the programmer is as reliable as possible. This can be crystallised into the following definition of program reliability:- that the terminal will not produce meaningless, obscure or incorrect information to the user regardless of what information (including sequence and timing) is input to the terminal by the user or any peripheral device connected to the terminal.

This means, in practice, that any invalid combinations of input data must be detected by the broadcast program and not be allowed to impinge on the fundamental elements of the terminal, i.e. the control program and the microprocessor hardware. For example, if the user provides data to the program which would ultimately result in the microprocessor attempting to divide a number by zero, then this state of affairs must be forecast by the program as soon as possible after the offending data has been input and

a suitably "friendly" conversation enacted with the user in order to correct the data. It is most important that this conversation be carried out in terms of the application and, in particular, of the data to be input, and not in internal or computer terms i.e. a message of the form "the number of recirculating filter modules you have suggested is too small to be practicable – a value of at least 20 in these circumstances would be reasonable" would be acceptable, while a message of the form "invalid operation – attempt to divide by zero" would *not* be acceptable.

4. TERMINAL CONFIGURATIONS

Based on the application areas outlined above and some miscellaneous and esoteric applications, a list of some of the attributes and peripheral devices that may be found on a telesoftware terminal of the future are listed below:-

The basic telesoftware terminal, consisting of a television receiver, teletext decoder, telesoftware control program in ROM, Broadcast Language interpreter in ROM, the minimum amount of RAM, and the teletext numeric keypad.

A QWERTY-type keyboard for the user to type textual information into the terminal, perhaps in conjunction with an educational program.

Additional RAM, to permit either more sophisticated programs to be received and executed, or to allow programs to run faster.

Multiple joysticks, paddles, etc, together with electronic clocks and timers, to permit, for example, "skill" programs to be executed. Also required in this area might be flashing lights, sound generators, speech synthesisers (also useful with educational programs), rifles and lightpens.

Tactile output (of text and ideally graphics) would provide an excellent means whereby the visually impaired might use telesoftware.

For the "offline" storage of programs and data cassettes, cartridges and disks will prove useful. Disks will also be beneficial in effectively extending the possible size of programs executing in the terminal, and for providing local databases upon which the broadcast programs might operate.

To provide "hard copy" conventional printers and plotters may be required, as well as devices which can provide a copy of the information on the television screen (in monochrome, but ideally in colour).

Badge and document readers may be useful in some semi-commercial applications for the input of data.

Connexions may be required to and from devices in the home (examples might be:

control of lights to simulate occupation as a deterrent to burglars; monitor of a car's electrical system to provide a diagnostic capability).

Some possible enhancement paths from the basic terminal might thus be:-

More RAM

Peripheral devices as required for particular applications

Additional ROM's, e.g. to permit home programming, or to "lock" specific programs into the terminal for commercial use. For example, terminals used for credit-card verification would generally only ever have one broadcast program loaded into them, and so the program could be permanently installed in such terminals, thus avoiding a) the need to load the program each day, and b) the need to actually transmit the verification program at all, merely the data for it.

5. COMPARISON WITH THE BRITISH POST OFFICE (BPO) SITUATION

It is expected, from reports in the press, and conversations with consultant at CAP, that the BPO will offer its own telesoftware system for public use in the near future. Personnel at CAP have given a tentative date for operation of a prototype system as October 1979.

The BPO's view of telesoftware (or more accurately CAP's view, since CAP appear to be supplying the BPO with the technical path to follow) is that it should be analogous to the mass storage system (e.g. an "intelligent" cassette recorder) of a home computer. This means that the user of the terminal (after having initiated a successful connexion to Prestel) would type a request for a program to be loaded into his terminal and an indication to be given to him when this loading had been satisfactorily completed. He would then tell the terminal to run the program. This is very similar to the way that a normal home computer is operated with a firmware resident BASIC system providing the programming language. This is in fact the language which the BPO have chosen to be transmitted to, and to be executed by the Prestel telesoftware terminals. The actual language to be transmitted will be a compacted form of ANSI Minimal BASIC, with extensions to be determined by CAP. The compaction method will involve techniques such as implied line numbering and single byte codes for keywords. However, apart from this compaction, the programs transmitted will be in "standard" BASIC, and the firmware within the terminal will consist mainly of a normal BASIC interpreter system, similar to that in a normal home computer (such as a Commodore PET). Also there will be provision for the user to examine the BASIC program which has been loaded into his terminal and modify it if he wishes, or indeed to input his own BASIC program from his keyboard, or any suitable peripheral connected to his terminal (such as a cassette recorder).

One implication of the above is that the BPO telesoftware terminal will necessarily be equipped with a full QWERTY keyboard in the same way that a home computer is so equipped.

This has far reaching consequences, and highlights one of the fundamental differences between the philosophies of the BPO and ITCA. The BPO regards itself as being (or becoming) the communications carrier between the Information Providers (IP's), who will provide the BASIC programs as standard pages on Prestel (in the same way as IP's provide textual/graphic pages on Prestel at the moment) and the users, who will expect to receive over the telephone lines a program written in BASIC which will run in their terminal. Thus the BPO will leave the integrity and reliability of the programs in the hands of the IP's, in the same way that no editorial control is made by the BPO on normal Prestel pages. A failure or abnormality in the BASIC program will normally make itself known to the user as a message from the BASIC interpreter, as if the user himself had written the program. The program would often halt under these conditions, and control would be passed from the program to the BASIC interpreter. The BPO also proposes that a BASIC program running in a terminal would be able to transmit information back to the Prestel computer, in order perhaps to interrogate a database, in exactly the same way that the user himself could do by using his keypad.

Contrast that policy with the ITCA approach which is to regard the terminal with an ORACLE supplied program running in it as, to the user, effectively a piece of hardware. That is, the user will be unaware that a "computer program" has been loaded into his terminal, merely that a "TV game" (say) has been broadcast and is now running as part of his terminal. He will be able to stop, continue and restart the "game", but he will not be able to examine the program, let alone modify it or insert his own. Any failure apparent to the user will be as a result of either a hardware malfunction in the terminal or a fault in the transmitted program. In either case the user will regard it as a fault in his terminal and not as a computing problem.

Thus the BPO system will use, and can possibly tolerate, lower standards of reliability and integrity in the programs it transmits than can ORACLE, since the Prestel telesoftware user will regard his terminal as a "computer" while the ORACLE telesoftware user will regard his terminal as a sophisticated TV games machine.

The Prestel system also has significant transmission differences to ORACLE, and although these do not impinge on the languages that are used to transmit the programs, they do provide an area where the teletext and viewdata telesoftware systems must diverge. The time required for the Prestel computer to respond to an action on the part of the user is quite short and is generally independent of the page that the user wishes to access (with the exception of pages that he has recently accessed). The transmission of Prestel pages to the terminal is fast (1200 bits/s) and fairly reliable (due perhaps to the Prestel computer always being accessed by a "local call"?) and CAP have found it only necessary to insert a block check character (BCC) at the end of each page, which is validated by the transmission control program in the terminal, in addition to the parity bit which is appended by the Prestel system itself to every character transmitted. If the terminal detects an error in any of the character parity bits or the BCC it requests the Prestel computer to retransmit that page, which is a very fast response since the Prestel computer retains the last page transmitted to each terminal in a "fast access" memory.

(It is understood from consultants at CAP that the BPO are planning higher speed

options than the 1200 bits/s from computer to terminal currently used, and also transmission protocols more suited for the computer-to-computer communications of viewdata telesoftware.)

On the other hand, the time taken for ORACLE to "respond" to an action on the part of the user depends on the number of pages being broadcast by the system and also on the relation of the page requested to the page being broadcast at the time the user's request was made. In general, ORACLE's response is at least an order of magnitude slower than that of Prestel. Hence it is important that once a page has been received, every effort be made by the terminal to remove any errors present in the page before resorting to the time consuming action of waiting for the page to be broadcast again. Thus broadcast telesoftware has to use a checking scheme providing not only error *detection* (as in the Prestel case) but also error *recovery*. Thus broadcast telesoftware will use a combination of character parity, BCC and error correcting codes (such as the Hamming Code).

Whereas the data from the Prestel computer travels mainly along solid copper and a few switches, the data from the ORACLE computer must contend with radio interference and fading, and has been demonstrated as having a high error rate, reinforcing the need for a sophisticated error recovery mechanism for broadcast telesoftware which would be unnecessary and unattractive to the BPO.

The fundamental differences between the BPO and ORACLE approaches to telesoftware can be summarised thus:-

i) the ORACLE telesoftware terminal will be regarded by the user as a sophisticated TV games machine, while the Prestel terminal will be regarded by the user as a home computer, and

ii) the error control techniques of the two systems are radically different due to the great dissimilarity between the two transmission media.

Thus, BPO telesoftware and ORACLE telesoftware are essentially complementary in their marketing approaches: BPO telesoftware is for commercial users, while ORACLE telesoftware is aimed at the domestic market.

However, there is scope for some standardisation between the BPO and ORACLE systems and this will be discussed further in Section 11.

6. THE BROADCAST TELESOFTWARE SYSTEM

Overview (see Figure 1)

A fundamental part of the overall telesoftware system will be the program production and testing system (PPTS).

The function of the PPTS is to produce fully tested, highly reliable programs in a form immediately suitable for transmission by the ORACLE system. It must be possible for the PPTS to produce these programs and to test them, on both simulated and real telesoftware terminals, without reference to the ORACLE system itself, and to only send the final results to ORACLE. A block diagram showing the fundamental elements of the PPTS is shown in Figure 2.

The PPTS would essentially be a standard multi-access computing system with a flexible and suitably sized filing system.

Once the idea for a particular program has been conceived, the requirements of the program would be defined rigorously so that the program can be specified in flow chart form (together with graphic layouts if required) and then written in the Programming Language for Telesoftware. This program would then be input to PPTS and edited if necessary using the file editing programs. Compilers and/or interpreters would then turn the program into a form suitable for execution *by the PPTS*. Meanwhile the graphic layouts would have also been input to the filing system, and various production aids would be used to turn these layouts into data suitable for incorporation into the program under development. This combined program (or parts of it) would then be subjected to various software "test rigs" where it would be exhaustively tested and evaluated using test data constructed for the purpose. These tests will include running the program (in PPTS executable form) on a simulated telesoftware terminal.

When the program is considered to be "solid" it would be passed to a translator program which "rewrites" the program into the Broadcasting Language for Telesoftware, and from there to a formatter which divides the program into pages suitable for ORACLE. The final test (i.e. the "road test") of the program before it can be sent to the ORACLE computer system would be to run it on a real telesoftware terminal connected to the PPTS.

Although the Program Production and Testing System may seem to be complex, all of the parts of it described above are vitally necessary for the production of reliable programs for broadcast telesoftware. In particular, the use of two separate languages, the Programming Language and the Broadcast Language, would appear to be an unnecessary complication. In fact, the use of two independent languages in this way has several outstanding advantages, and no significant disadvantages. The reasoning behind this approach will be expanded below, first by giving the requirements of a language in which the programs are to be written, then giving the requirements of the language in which the programs are to be transmitted to the users' terminals, and then, by showing the disjoint nature of these two sets of requirements and the penalties incurred by having two separate languages, to conclude that the approach chosen has overwhelming advantages.

7. REQUIREMENTS FOR A PROGRAMMING LANGUAGE

What follows are the general requirements for the language in which the telesoftware programs will be written. The bias is towards providing the programmers with the best possible facilities for producing reliable and maintainable programs.

7.1 The language should be well defined in its syntax, its semantics (in particular its arithmetic, logical and input/output facilities).

7.2 A language is needed which will enable programming to be carried out in a structured manner, leading to faster program production and more error free programs, together with easier maintenance.

The "maintainability" of programs is an often overlooked aspect of programming, but one of vital importance, not least in economic terms. Problems can and do arise considerable periods of time after a program was written, perhaps an error shows up unexpectedly or an improvement to the program is required, and it is only then that the "visibility" of the program (a result of the combination of good programming staff and a structured programming language) makes itself felt in the minimising of the time and effort required to modify the program while still maintaining its integrity and reliability.

7.3 A high level language is required, with many facilities to enable the rapid production of the required program constructions without the need to resort to programming "tricks". (These "tricks" are almost always impossible for another programmer to understand, and usually difficult even for the original programmer after a period of time). The avoidance of the use of "tricks" is vital for program maintenance and enhancement.

7.4 The language chosen must support the combining of sub-programs from various sources, and must be in keeping with the concept of a "library" of sub-programs which can be incorporated simply and reliably into the final programs. This "library" technique will be of great economic importance in the continued production of programs for the ORACLE telesoftware system.

7.5 The programming language must be conducive to the technique of "shell" programs, and text and graphics generation packages. A simple example of the "shell" program technique is the First Aid program demonstrated on the experimental ORACLE telesoftware terminal at the International Broadcasting Convention in 1978. The program consisted of a standardised "kernel" program which handled the presentation of text and questions, the interpretation of the answers and the control of the flow through the "tree" structure of the text. Into this "kernel" program were "plugged" the explanatory texts, the questions, the responses to possible answers, and the "tree"

structure of the required presentation. Thus, the "subject expert", in this case a specialist in first-aid, was only concerned with the text, questions, answers, and the sequence of presentation, and *not* with the program itself. Thus the two areas of programming and subject expertise have been kept separate, so that the programmer does not become involved in subject matter, and the subject expert does not have to be a trained programmer.

7.6 The chosen language does not have to be an "easy" language, since, due to the requirement that highly reliable and easily maintained programs are vital, together with the constraint that the programs should be as compact as possible for transmission and terminal storage reasons, only competent professional programmers can be employed in the production of these programs.

8. REQUIREMENTS FOR THE BROADCAST LANGUAGE

What follows are the fundamental and vital requirements for the language in which the telesoftware programs will be transmitted. The bias is towards providing a "clean" environment, taking into account the peculiarities of the broadcast system, the economics of terminal manufacture, and the required appearance of the system to the user as a sophisticated TV-games machine.

8.1 The Broadcast Language must be independent of the equipment provided in the terminal by its manufacturer. In particular, the language must be independent of the microprocessor employed in the terminal.

8.2 It follows from the above that the programs cannot be transmitted in the machine code of any particular microprocessor, and must therefore be broadcast in either a high level language or an intermediate level language. To run programs in such languages, a computer has to be provided with either a compiler or an interpreter (or a combination compiler/interpreter). A compiler takes as its data the program written in the high level language and produces machine code which can then be run directly on the computer of interest. In the context of telesoftware, the compiler approach suffers from two main disadvantages. Firstly, the program in the high level language (the "source" program) needs to be held completely in the terminal while the program in machine code (the "object" program) is being produced by the compiler, so that at some instant both versions will be held simultaneously in the terminal's random access memory. This leads to an excessive amount of memory being required in the terminal. Secondly, the execution of the object program cannot commence until the source program has been received and compiled. This would result in an even longer delay between the user's request for a program and the commencement of the running of the program.

The advantage of the compiler approach is that the object program would use the

microprocessor in the terminal efficiently. This is not seen as a great advantage, however, since even the current microprocessors which may be used in telesoftware terminals are very fast as well as being very low cost.

Using the interpreter approach, the source program only is held in the terminal's random access memory, and each statement of this program is individually "interpreted" into a sequence of calls to machine code sub-routines which form part of the interpreter. It should be noted that at no time does the source program use the terminal hardware directly, only via the interpreter. Thus full protection of the terminal against malfunctions of the source program can be provided entirely by the interpreter. The interpreter, once a page of program has been correctly received, can immediately commence execution of it, thus not incurring any extra delay beyond that incurred by the broadcast mechanism. The interpretive approach results in programs running typically about 30 times slower than the compiled version, but even the interpreted programs run acceptably fast on microprocessors of the current generation, so speed of execution is not regarded as an area of contention, particularly as 16-bit microprocessors, which seem very suitable for this application, are now becoming available, and will offer very substantial speed improvements over the current 8-bit microprocessors.

The conclusion is, then, that the telesoftware programs be run by an interpretive system, which will be resident in Read-only Memory (ROM) within the telesoftware terminal.

8.3 The number of bytes (8 bit portions of digital information) needed to transmit each telesoftware program should be as small as is reasonable. The time taken to load a program into a terminal is composed of the time taken for the required page to reach its turn to be broadcast, and the time taken to transmit the actual page. If the size of a program is not minimised the number of teletext pages needed to contain it will be the dominating factor in the time required to load the program. The size of a program in bytes also influences the chance of a transmission error occurring, with the consequent possibly of a long pause before the offending page is rebroadcast.

The size of the transmitted program is also mirrored by the amount of RAM needed in the terminal to contain it, thus reinforcing the above argument.

8.4 It is highly desirable that the system be able to transmit larger and more complex programs to the terminals and that even fairly modest terminals be able to run them (perhaps with some restrictions, such as a slower speed of execution). It is important, therefore, that the broadcast language be able to support the fragmentation of a program into several units which can be transmitted independently. The user would commence operation with the master unit of the program, which would then, when the user had reached the conclusion of this unit, cause the control program in the terminal to load a subsequent unit of the broadcast program into the terminal. Apart from the master unit, the sequence in which units of program were loaded into the terminal would not be fixed, but would depend upon the interaction between the user

(and possibly the peripherals attached to the terminal) and the units of the program previously loaded. It is most important that data generated as a result of this interaction be made available to subsequent units of the broadcast program when they are loaded into the terminal.

8.5 The broadcast language must be able to support the wide range of peripheral devices which may be connected to terminals. Since the nature of most of these devices will not be known when the broadcast language is specified, they will require to be regarded by the broadcast program as "conceptualised" peripherals. That is, broadcast programs will regard all peripherals as belonging to a very small set of idealised devices, and the control program (or the interpreter program) in the terminal will perform the appropriate "mapping" of these idealised devices into the actual devices.

Also, as indicated in the review of possible applications for the telesoftware system, the broadcast programs will need to be able to initiate clocks and timers, to be able to continue execution while these clocks and timers are running, to be able to examine and modify the state of the clocks and timers, and to be capable of having the flow of execution of the program altered when a clock or timer reaches a certain value.

In a similar manner, the broadcast programs must be able to request several peripherals to provide the program with data at any one time, to be able to continue execution while awaiting data to be presented from any of these devices, to have the flow of execution of the program modified appropriately when data does arrive, and to be able to identify the source of the data.

The broadcast language should also have the full ability to present text and graphics on the screen of the terminal, including full control of the cursor and the ability to modify any part of the screen at will. The use of graphics by the broadcast program will, in general, be different from the use of graphics in standard teletext. In teletext the pictures are precomposed as fixed entities, whereas in telesoftware there will be a requirement for pictures to be constructed as the execution of the program proceeds, the actual picture achieved being a result of interaction between the user and the program (a trivial example would be showing the user's attempts to find a path through a maze). Thus there will be, for example, a requirement for the broadcast program to be able to draw lines of varying lengths and slopes.

8.6 The Broadcast Language will have to be defined completely and unambiguously, in all of its aspects, such as syntax, semantics, arithmetic and logic operations, and input and output facilities, before manufacturers can attempt to produce terminals to receive and execute it.

There is no requirement to permit program construction or modification by the user at the terminal. That is, there is no requirement for editing, or line numbering and renumbering, no requirement for 'operating system' commands, no requirement for syntax and semantic checking, no requirement for run-time diagnostics. The only requirement is to load and execute (by interpretation) the broadcast programs. If a

problem occurs, command is to be passed to the control program as a "fault" occurrance.

This is in contrast to the BPO situation where all of these facilities will be required and will form a fundamental part of the Prestel telesoftware system.

8.7 *Important Note*

The Broadcast Language is the only part of ORACLE telesoftware that requires a *fixed* specification, since it will be published to enable manufacturers to produce terminals which will receive and execute programs in this language. The Programming Language can and indeed will vary to cater for the many applications which will undoubtedly be found for telesoftware. Provided that the programs written in these various languages are translated by suitable programs into the strictly defined Broadcast Language no anomalies will result. Thus, programs can be written in the language which best suits the applications of the programs but the programs are ultimately broadcast to the terminal in a language which is best suited to this operation.

In order to allow manufacturers independent of the terminal suppliers to produce peripheral devices and also to permit vital standardisation with the BPO, a standard hardware interface or interfaces for peripheral devices will need to be fully specified at an early date. It is desirable, but not vital, to have a standard means of mapping "conceptualised" devices onto these real devices.

This area of the connexion of peripheral devices will be examined in a later section.

9. CHOOSING A PROGRAMMING LANGUAGE

As has been indicated above, the language in which a program is written will not be a fixed part of the ORACLE telesoftware system but can vary to suit the application. Nevertheless, any language chosen to be a Programming Language must conform to the requirements laid down above, and must be precisely transformable into the Broadcast Language.

However, it is important to choose a general purpose programming language in which most of the programs will be written. Possible contenders are:-

BASIC	CORAL
FORTRAN	RTL
ALGOL	PASCAL
COBOL	A language specially designed for telesoftware.

The requirement for programming to be carried out in a structured manner rules out

BASIC which is inherently unstructured. BASIC is a language which was designed to facilitate the teaching of elementary programming, and was not designed for the production of "serious" programs. The suitability of BASIC as even an introductory language has been brought into debate internationally recently, since it does not encourage (it may even discourage!) the formality and structuring of programs necessary to produce reliability and maintainability.

FORTRAN and ALGOL are elderly scientific programming languages designed for the solution of essentially mathematic problems. While ALGOL allows reasonably well structured programs to be written, it is weak in its provision of facilities for receiving data from and sending data to peripheral devices. FORTRAN, although possibly the oldest and one of the most widely used programming languages, does not lend itself readily to structured programming, and its input and output facilities are primitive. Programs written in either of these languages could be transformable into the Broadcast Language, indeed ALGOL is often used as a "formal" language in which algorithms (program solutions of various diverse problems) are described. Programmers often, having read these algorithms written in ALGOL, perform a strict translation into the programming language they are currently employing. Both languages have very poor text string manipulation facilities. Provision of such facilities in the Programming Language is essential if programming "tricks" are to be avoided, since text string manipulation is a vital aspect of programs in most of the likely areas of application of telesoftware.

COBOL, by its very name, is a business-oriented language and is not at all suitable for general-purpose programming. It does, however, reign supreme in the business world, and would be a possible language for non-trivial commercial programs in the ORACLE telesoftware system, provided that it can be satisfactorily transformed into the Broadcast Language. There is serious doubt whether this could be achieved, however. CAP is using micro-COBOL as the language for its own telesoftware system (hosted on Prestel) called Viewdata Terminal Programming (VTP). The applications of VTP are exclusively commercial, and the customers expect and see a desk-top computer programmed in COBOL and loaded from Prestel.

CORAL and RTL are both British "real-time" languages designed for process control, and have most of the attributes necessary for a general purpose Programming Language, except that their use is confined to a small number of computers. However, they are designed to produce highly reliable, efficient programs, with excellent facilities for handling peripheral devices, clocks, timers, etc. RTL, in particular, is a very "neat" language producing highly reliable and readable programs. It is often used by first being translated into a lower level language which is then interpreted by the target computer. Neither language has good text string manipulation facilities since they were designed for industrial process control (RTL is an ICI product originally used for chemical plant control. CORAL is a product of the Royal Radar Establishment and was designed for the processing of radar signals.).

PASCAL was specially designed to produce well structured programs. It is well defined and has been implemented and is in use on a wide range of computers, from large mainframes to microcomputers. PASCAL is now taking over from ALGOL the role of

algorithm description language, since the intentions and methodology are shown more clearly in PASCAL programs. PASCAL has excellent text string manipulation facilities, but it is not specifically a real-time language, merely providing the normal question and answer interaction. PASCAL is often translated into a simpler language (P-code) which is then interpreted by the target computer.

None of the above languages is completely ideal for the Programming Language for telesoftware, and it would be possible to define a language which came closer to being the ideal. However, the great amount of effort needed to specify such a language and to bring it into operational use could not be justified. A more acceptable approach might be to produce enhancements for an existing language. It should be remembered that there is no formal requirement to use a "standardised" language as the Programming Language for telesoftware, since the number of programmers involved will be small and under the aegis of ORACLE.

From the above it is concluded that PASCAL is the best choice for a general-purpose Programming Language for telesoftware, but that extensions to it may be needed to handle the "real-time" nature of some applications, such as the need for clocks and timers and interrupt facilities. COBOL may be useful if substantial commercially-oriented programs are required, but considerable difficulty will be experienced in translating programs written in COBOL into the Broadcast Language.

10. CHOOSING THE BROADCAST LANGUAGE

As has been shown above it is a requirement of the Broadcast Language that programs broadcast in it be capable of being executed via an interpreter program resident in a terminal. It is also a requirement that the programs be broadcast using the smallest practicable number of bytes, thus suggesting that the Broadcast Language should be as "high-level" as possible.

The first of these essential requirements rules out most programming languages, and the remaining contenders are:-

RTL intermediate language
PASCAL P-code
BASIC compressed source.

RTL intermediate language and PASCAL P-code are very similar, being the assembler languages of hypothetical idealised computers into which the RTL or PASCAL source programs are translated. The lower level intermediate language programs are often then executed via an interpreter program in the target computer. Since the intermediate languages are similar and PASCAL has been recommended as the Programming Language, RTL intermediate language will not be considered further.

BASIC is a "higher level" language than P-code, but P-code, since it is essentially an

assembler language has more inherent flexibility, especially in the control of peripheral devices. (P-code, being an assembler, could be executed in the terminal by a micro-processor designed to execute P-code *directly* giving a fast and efficient execution. This sort of implementation would be very difficult, although not impossible, with BASIC).

A comparison was made between a form of BASIC, compacted as much as possible (see Appendix III) and P-code similarly treated (Appendix II). A typical program which has been produced in both forms is given in Appendix V. Several programs were also thus treated, giving very similar results. The comparisons show that for a given program, P-code requires 3 times the number of bytes for transmission than does compacted BASIC.

Thus, it is concluded that, although the instructions may form only a small part of a complete program (the predominant factor in the overall size of a program often being the large amount of text and graphics required), the need to "protect" at least the part of the program containing the instructions during transmission by error detection and correction bits tends to counter this, as does the need to receive another absolutely correct text and graphics (miscellaneous errors may be tolerated in text and graphics since the user can often imply the correct version by context), so that compacted BASIC, requiring only one-third of the transmission time of P-code, should be the choice for the Broadcast Language.

Thus, we conclude that the general purpose Programming Language be PASCAL and the Broadcast Language be a version of BASIC. A subset of PASCAL can easily be specified which will ensure that the translator program, to produce a program in this version of BASIC from the subset of PASCAL, is straightforward.

11. STANDARDISATION WITH THE BPO

There are three possible areas where standardisation between ORACLE telesoftware and BPO telesoftware might be brought about: transmission protocols; broadcast language; interfacing of peripheral devices.

11.1 Transmission Protocols

The radically different environments in which the transmissions of BPO telesoftware and ORACLE telesoftware programs will be made, as shown in Section 4, makes standardisation in this area impracticable.

11.2 Broadcast Language

Although the approaches to telesoftware by the BPO and ORACLE are completely different and really complementary in that the BPO system is aimed at business users

and behaves in a similar way to a home computer, while the ORACLE system is aimed at the domestic user and behaves like a sophisticated TV games machine, it has been concluded in Section 9 that the Broadcast Language for ORACLE should be a compacted form of a version of BASIC. That means that the transmission languages for both systems will be versions of the same nominal language. The fundamental difference, as detailed in Section 4, is that the BPO system will transmit programs of potentially lower reliability than those in the ORACLE system to terminals which, in architecture, are very similar to home computers in that the nature of the programming system is visible to the user. In the case of the ORACLE system the programming system *must* be invisible to the user.

However, it is viable to regard at least a major portion of the terminal system required for the ORACLE Broadcast Language as a subset of that required for the BPO language. Figure 3 attempts to illustrate this in diagrammatic form. Each of the telesoftware systems will require its own transmission protocol sub-system, but will share a section of the control and interpreter system. Different additional sections of this control and interpreter system will be brought into play for each type of telesoftware. It is to be expected that the extra section required for the BPO system will be significantly greater than that required for the ORACLE system, in order to permit, for example, user development of programs.

It should be noted that, even in the worst case (where totally different languages were chosen by the BPO and ORACLE) the only penalty incurred would be two separate interpreter/control systems held in ROM in the terminal. Since it is anticipated that the ROM for the ORACLE system (at least) will be produced in substantial quantities, the cost of ROM in the terminal to cater for ORACLE telesoftware will not be a substantial part of the cost of the telesoftware unit in the terminal.

11.3 Interfacing of Peripheral Devices

This is an area where standardisation between ORACLE and the BPO is vital if telesoftware in general is to become a widespread facility. Without standardisation, a peripheral supplier would be faced with producing two interface versions for each device, and a user (requiring the use of both BPO and ORACLE telesoftware systems) would be forced to purchase more expensive peripherals than necessary.

It should be noted that standardisation is vital only for the hardware interfaces for peripherals. Standardisation in the way these devices are handled by the terminals' software and the manner in which they are presented to the broadcast programs, is not vital and will depend upon the outcome of attempts to standardise the Broadcast Language.

Four main areas of hardware interface standardisation have been identified:-

i) electrical isolation between peripheral devices (and their cables) and the terminal (which will contain hazardous EHT voltages).

ii) control and data signalling protocols.

iii) hardward bus architecture.

iv) mechanical interconnexion details (i.e. plugs and sockets).

To summarise the above section, then, we conclude that:-

i) it seems likely that a substantial subset of a version of the BASIC language can be defined which would form a common kernel for both systems.

ii) standardisation of the hardware details for the connexion of peripheral devices is vital.

12. SOME AREAS IN NEED OF FURTHER INVESTIGATION

12.1 Range and Precision of Numbers

Within the specification of the Broadcast Language, attention will need to be paid to the representation of numbers and arithmetic and logical operations. These must be independent of the word length of the processor used in the terminal (8, 16, or even 32 bits) and must not be unduly restrictive.

12.2 Input and Output

The "conceptualised" peripheral devices must be exactly specified, and although the number of different types of devices should be kept small, the choice should not inhibit the use of the large variety of devices which may arise in the future.

12.3 Graphics

Teletext format graphics has two main drawbacks when applied to telesoftware:- i) it was designed for pre-formatted pictures, not for pictures being manufactured "on-line" as a result of interaction between the user and the broadcast program, and ii) the resolution of 72 x 80 is completely inadequate for most applications. Some consideration should be given, therefore, to the possibility of the telesoftware part of an advanced terminal being able to rectify these deficiences by bypassing the teletext graphics facility and supplying its own, higher quality, interactive graphics directly to the screen driving circuits of the television.

12.4 Text Reduction

A likely major contribution to the overall size of a broadcast program is the textual and graphic information contained within the program, and techniques to minimise the

transmission and storage requirements for this information require consideration. As an example, the First-aid demonstration program shown during the International Broadcasting Convention in 1978 was composed as follows:- executable instructions: 7%; textual information: 93% (including 6% for "pointer" type information within the text, without which the text would have been several times larger). It must be emphasised that, while the program proper was very close to the author's desired form for the program, the text used was less than one-third of the text originally suggested by the author. Also graphics, which could have proved useful, were not utilised by this program (for technical reasons associated with the demonstration terminal).

A most promising method for the compaction of textual information within a broadcast program is to identify (by means of a utility program operating in the PPTS) repeated text strings in the program. These strings would then be held in a "string vocabulary block" within the program, and invoked from as many places in the program as required by simple pointers. A pointer at the end of each string would cause a return from the invocation.

The telesoftware demonstration programs so far constructed also show the need for a TAB function to be available in the Broadcast Language.

12.5 Determining the Amount of RAM in the Terminal

It will be desirable for the amount of RAM installed in a particular terminal to be determined automatically by the terminal, and for the terminal to then, in conjunction with the master block of the requested broadcast program, to decide a) whether the terminal has sufficient facilities for the program to be run at all, and b) if the program can be run, the best method of loading the blocks of the program into the possibly restricted amount of RAM available.

Some type of "look ahead" mechanism in either the control program, the interpreter program or the broadcast program will be desirable, so that the next block of the broadcast program could be loaded invisibly to the user when he is nearing the end of execution of the current block. This is a non-trivial problem!

13. CONCLUSIONS

13.1 There will be a wide variety of peripheral devices connected to future telesoftware terminals, to cater for a large number of applications.

13.2 The ORACLE telesoftware system must provide highly reliable programs which will run in terminals which will appear to users as similar to very sophisticated TV games machines, while the BPO telesoftware system will generally provide programs for more commercial applications where the terminals will appear as devices similar to small computers.

13.3 To provide the necessary high reliability for programs broadcast by ORACLE it is necessary for the programs to be written and tested in one language but to be broadcast in another language, due to the disjoint nature of the two tasks. A substantial program production and testing system will also be required.

13.4 PASCAL would prove a useful general purpose programming language for telesoftware.

13.5 The language in which programs will be broadcast should be a rigorously defined version of BASIC with suitable extensions, compacted textually to minimise transmission time.

13.6 It appears possible that a substantial subset of this language could be defined to form a kernel common to both the ORACLE and BPO systems, but complete compatability between the two systems is not possible.

13.7 Standardisation for the connexion of peripheral devices to telesoftware terminals is vital.

14. RECOMMENDATIONS

It is recommended that:-

14.1 A rigorous specification for the broadcast language be produced and published at an early date.

14.2 Discussions with the British Post Office and its agents be carried out in the same time scale as the production of the specification to yield (if possible) a common subset of this language.

14.3 Discussions with the British Post Office, its agents and any other interested parties be carried out at an early date to establish and publish standards for the connexion of peripheral devices to telesoftware terminals, and

14.4 The provision of program production and test facilities be made at the earliest opportunity to avoid possible delays in the introduction of a public ORACLE telesoftware service.

BRIEF DESCRIPTION OF BASIC

A BASIC program consists of a series of sequentially ordered statements. Each statement is preceded by an integer called the statement number. This number serves as both a statement sequence number as well as a line identifier. An example of a BASIC statement is:

```
100 PRINT 'FRED'
```

When a program written in BASIC is run the statements are executed in order of statement number, unless the execution of a "control transfer" statement affects the normal order. The statement numbers are integers that range from 1 to 99999. They do not have to be in cardinal sequence (i.e. 1, 2, 3 . . . n-1,n) but they must be in ordered sequence (e.g. 10, 20, 22, 25, 30, . . .n).

Each statement starts after its statement number with a full or partial English word. This word denotes the type of the statement. Spaces have no significance except in strings.

Programs are usually run under a BASIC language processor system, and commands, such as LOAD, OLD, NEW are provided to permit the user to load and save programs, list, run, stop, continue and abort programs.

Two types of data are available in BASIC: numeric and string, and constants and variables of both types are allowed. A numeric value is a floating point number, its precision and range will depend upon the BASIC system used. A numeric constant is written as a signed decimal number. It may contain a decimal point and it may be followed by an exponent. A string value is a string of ASCII characters. A string constant is written as a set of 0 or more contiguous ASCII characters enclosed in delimiting quotation marks.

A scalar variable is implicitly defined when it is used in a BASIC program. The type of scalar variable (i.e. numeric or string) is determined by the form of the variable name. The name of a numeric scalar variable is a single letter, optionally followed by a single digit. Each variable represents a single numeric value; there are 286 possible numeric scalar variables. The name of a string scalar variable consists of a single letter followed by a dollar sign. A string scalar variable represents a character string of variable length.

An array is an ordered set of values. All elements of an array have the same data type (i.e. either numeric or string). A numeric array name is represented by a single letter followed by a parenthesised list of one or two bounds. An array element is designated by an array subscript that is either one number (bound) in parentheses (one-dimensional array) or two numbers (bounds) in parentheses and separated by a coma (two-dimensional array). A string array name is a single letter followed by a dollar sign followed by a parenthesised list of one or two bounds.

BASIC expressions are constructed from operators and operands. An operand may be a constant, scalar variable, subscripted array element, or a function reference. BASIC has two

types of expressions: numeric and string. The operators that can operate on numeric operands are addition, subtraction, multiplication, division and exponentiation, and are performed in floating point arithmetic. BASIC defines six relational operators (less than, greater than, equal, less than or equal, greater than or equal, not equal) that may be used in either numeric or string expressions.

BASIC provides system functions (e.g. SIN, COS, SQR) and permits user-defined functions. For example SIN (X) computes the sine of X, X expressed in radians.

Statement types are provided by BASIC to permit the program to send data (either numeric or string) to the user, in a formatted manner if required, and to receive numeric or string data from the user. System functions such as LEN and SUB permit the analysis and matching of string data supplied to the program by the user.

BASIC is a language designed to teach elementary programming and its main failings are i) the multitude of different versions of BASIC, ii) the language does not encourage (perhaps it even discourages) the production of programs in a structured and reliable manner, iii) the lack of parameterised subroutine call facilities, and iv) the lack of a local/global variable feature.

BRIEF DESCRIPTION OF PASCAL

A need was felt for a language well defined in its syntax (the grammar which defines valid statements in the language) and in its semantics (the meaning or procedures implied by a statement in the language). Secondly, a language was needed which would enable programming in a structured manner, leading to faster and more error-free solutions to problems, together with easier program maintenance. These requirements led to the definition of PASCAL by Prof. Niklaus Wirth of the University of Zurich in 1968. The language has been implemented on nearly all mainframe types of computer, and on many mini- and microcomputers. There are very few "versions" of PASCAL, unlike BASIC (where every machine has a different BASIC, and a program written for BASIC 'A' will not run under BASIC 'B' without modification), so essentially PASCAL programs should be transportable from one machine to another.

A language rather like PASCAL, ALGOL 60, existed before PASCAL but it had a crucial weakness in the area of data structures. PASCAL, like most other languages, has the fundamental data types of integer and floating-point numbers, strings and logical. In PASCAL, however, these and other (user-defined) data types may be structured in arrays, sets, records, files and lists.

An array is a structure where an individual element is accessed by a simple data type, e.g. integer.

A set is an aggregate of data of one type, e.g. a set of characters. Individual elements of a set cannot be accessed but operations can be performed on the set as a whole. Valid set operations are union, intersection, set difference, equality and membership.

The record type is perhaps the most powerful and flexible data structure in PASCAL. The components which make up a record can be of any type mixed together and additionally parts of a record, known as variants, can be dependent on other parts at execution time.

The file data structure is the method used to communicate data to and from external devices, such as keyboards, screens, disks and printers. All types of data structure may be read from or written to disk files, but only characters are valid for keyboards and printers.

An unusual feature of PASCAL is the pointer data type. This defines a variable not to have a value as all others do but solely to point to another (conventional) variable which can be of any type. PASCAL does not include list processing functions, as provided by special languages like LISP and POP-2 but these are easily produced using the simple pointer-moving facilities of PASCAL.

Most versions of BASIC do not permit parameter passing to subroutines and do not employ the concept of local and global variables. Parameter passing by reference and or value is fully supported in PASCAL, and PASCAL functions may return any simple data type. These procedures (sub-routines) and functions fully support local and global variables and

can be fully recursive.

Two sample programs, one in BASIC and one in PASCAL, to perform a bubble sort on an array A, are shown in Figure 4 as a comparison.

COMPACTED BASIC

For the comparison shown in Appendix V, the following compaction techniques were employed on the (otherwise normal) BASIC program:-

i) Implied line numbering. The program lines are numbered in cardinal sequence from 1 (i.e. 1, 2, 3, ...n-1, n), so that the line numbers can be implied and thus omitted.

ii) Space removal. All space characters, with the exception of those within strings, are removed.

iii) Keyword compaction. All keywords are replaced by single ASCII characters of the non-printing type.

iv) Carriage control compaction. Single ASCII characters are used as end-of-line markers instead of carriage return/line feed characters.

APPENDIX IV

P-CODE

One type of compiler program for the PASCAL language achieves portability between computer types by producing as its output an intermediate code which is the assembly language of a hypothetical stack-oriented machine rather than object code for an actual machine. This language (P-code) is in an easily readable textual form.

There are 47 instructions available in this "assembly language" and these (in compact form) require between 1 and 5 bytes each for storage. By examining large programs available in P-code, the average storage required for each P-code instruction is about 2.9 bytes. Figure 5 lists the P-code instructions.

COMPARISON BETWEEN COMPACTED BASIC AND P-CODE

Below is shown a program written in PASCAL and in BASIC, one of several for which a comparison was made.

Program in PASCAL:-

```
PROGRAM SORT;
CONST N = 1000;
VAR I, J, K, M, N: INTEGER; A: ARRAY [1 . . N] OF INTEGER;
BEGIN
  FOR I:=1 TO N DO A [I] :=1;
  FOR I:=1 TO N-1 DO
  BEGIN
    K:=I; M:=A [I] ;
    FOR J:=I+1 TO N DO
    IF A[J] > M THEN
    BEGIN K:=J; M:=A[J] END;
  A[K] :=A[I] ; A[I] :=M
  END
END.
```

Program in BASIC:-

```
100 REM PROGRAM SORT
110 FOR I=1 TO N
120 A[I] =I
130 NEXT N
140 FOR I=1 TO N-1
150 K=I
160 M=A[I]
170 FOR J=I+1 TO N
180 IF A[J]<+M GOTO 210
190 K=J
200 M=A[J]
210 NEXT J
220 A[K]=A[I]
230 A[I]=M
240 NEXT I
250 END
```

The PASCAL program produced 97 P-code instructions, which could be packed into 277 bytes. The BASIC program when compacted using the rules given in Appendix III required 94 bytes of storage.

Thus, this typical comparison between P-code and compacted BASIC revealed that P-code requires about 3 times the amount of storage than does compacted BASIC. (Note that this is for program instructions and numeric data only and does not include the storage needed for any text or graphical information required by the program.)

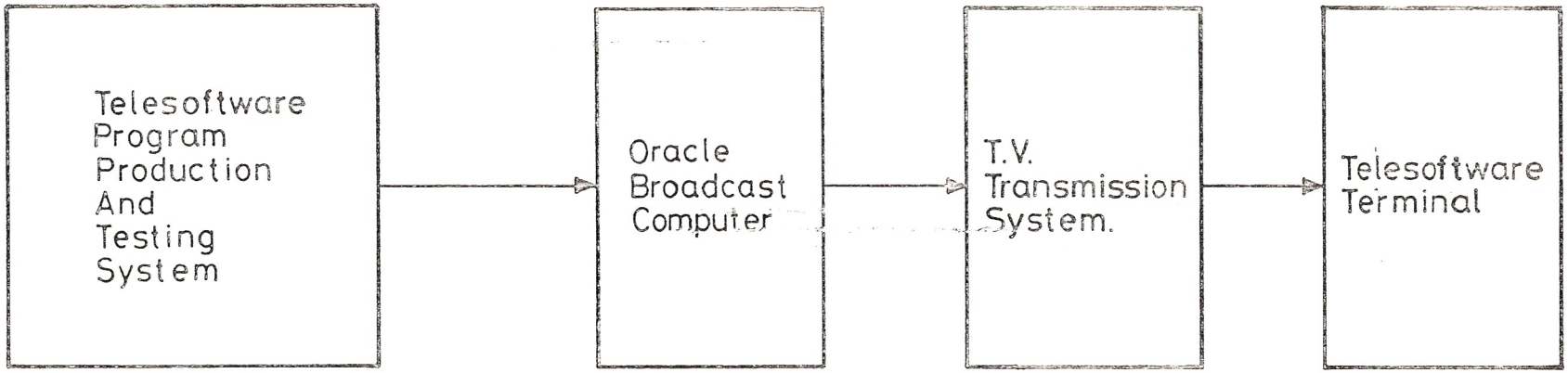


FIGURE 1.

System Overview.

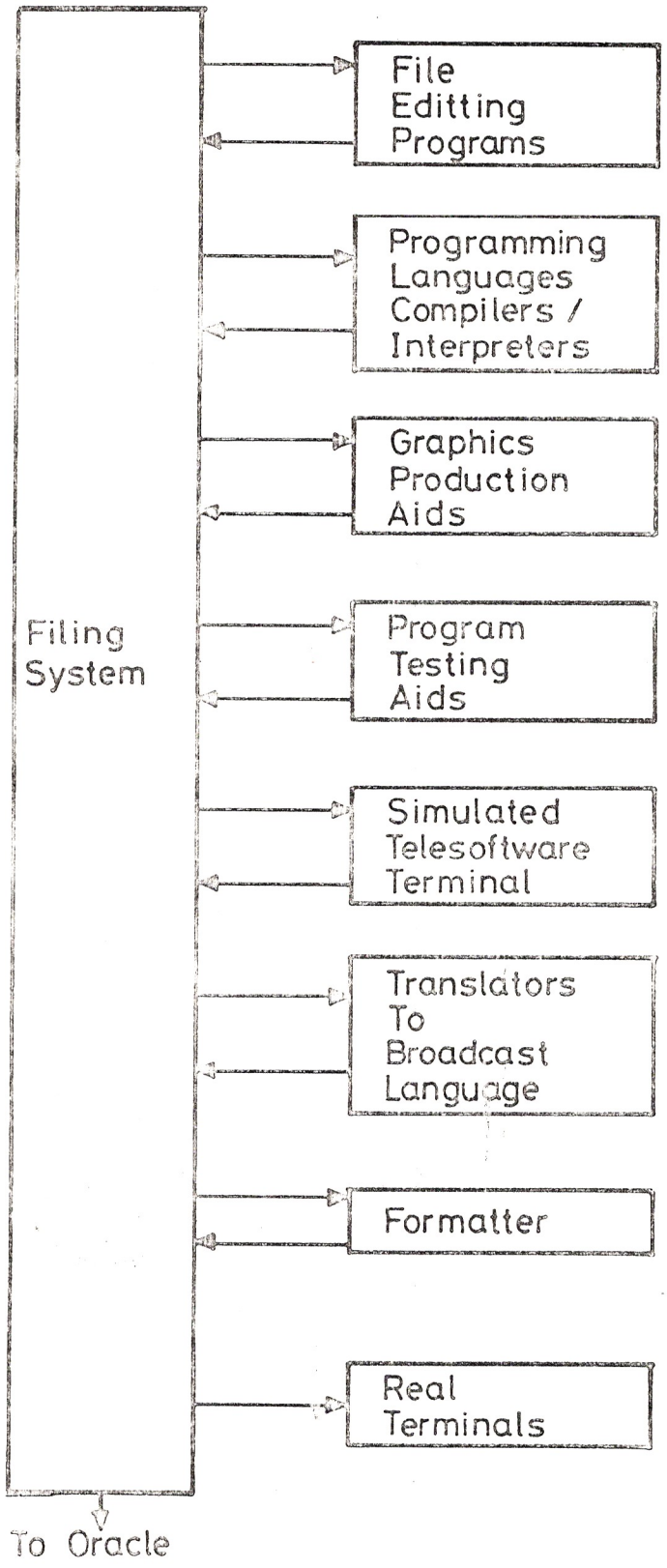


FIGURE 2.

The program production and testing system.

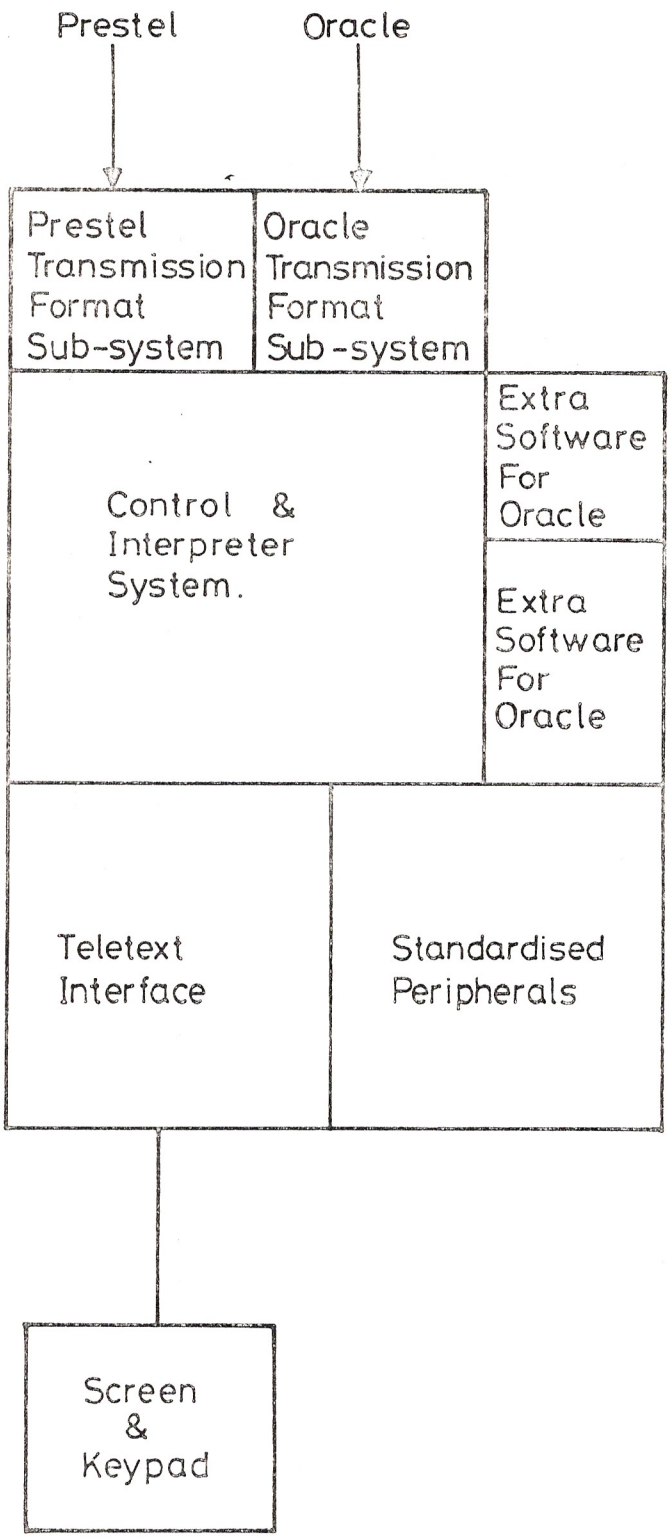


FIGURE 3.

Conceptualised Terminal for ORACLE and BPO Telesoftware Systems.

```
(*L+*)
PROGRAM SORTEX;
CONST X = 50;
TYPE LIST = ARRAY [1 . . X] OF REAL;
VAR A: LIST;
      J: INTEGER;

PROCEDURE BUBSORT (VAR L: LIST; N: INTEGER);
VAR I, M: INTEGER;
      T: REAL;
      SORTED: BOOLEAN;
BEGIN SORTED: = FALSE; M: = N;
 WHILE (SORTED = FALSE) AND (M >=2) DO
   BEGIN SORTED: = TRUE;
    FOR I:=2 TO M DO
     IF L[I-1] > L[I] THEN BEGIN T: = L[I-1];
                               L[I-1]: = L[I];
                               L[I]: = T;
                               SORTED: = FALSE
                     END; (* END FOR LOOP *)
    M: = M-1
   END (* BEGIN BLOCK *)
END; (* BUBSORT *)

BEGIN (* MAIN PROCEDURE *)
 FOR J: = 1 TO X DO A[J]: = 51-J;
 BUBSORT (A,X)
END.
```

Program Written in PASCAL

```
10 ! BUBBLE SORT DEMONSTRATION
20 DIM A(50)
30 FOR J = 1 TO 50
40 A(J) = 51 - J
50 NEXT J
55 M = 50
60 GOSUB 1000
70 STOP
80 REM
1000 ! BUBBLE SORT ROUTINE
1010 S = 0
1020 M = N
1030 IF S = 1 THEN 1999
1040 IF M < 2 THEN 1999
1050 S = 1
1060 FOR I = 2 TO M
1070 IF A(I-1) <= A(I) THEN 1130
1080 T = A(I-1)
1090 A(I-1) = A(I)
1100 A(I) = T
1120 S = 0
1130 NEXT I
1140 M = M - 1
1150 GOTO 1030
1999 RETURN
```

The Same Program Written in BASIC

FIGURE 4 Comparison Between Programs Written in PASCAL and BASIC

Alphabetic list of P-code instruction mnemonics

The T+ column is used to denote those instructions which will in use have a type character added to the mnemonic.

Mnemonic	T+	P	Q	Function
ABI				produce absolute value of integer
ABR				produce absolute value of real
ADI				produce sum of integers
ADR				produce sum of reals
AND				perform Boolean 'and'
CHK	✓	✓	✓	check that top of stack is ⩾ P and ⩽ Q
CHR				convert integer to character
CSP			✓	call standard procedure (Q gives its name)
CUP		✓	✓	call user procedure (Q gives its label, P—see note 2)
DEC	✓		✓	decrement top of stack by amount Q
DIF				evaluate set difference
DVI				integer divide
DVR				real division
ENT		✓	✓	enter block—see note 1
EOF				test for end of file condition
EQU	✓		☆	test for equality
FJP			✓	jump if stack top false to label given in Q
FLO				float the element below the top of stack
FLT				float the top of stack
GEQ	✓		☆	test for greater than or equal to
GRT	✓		☆	test for greater than
INC	✓		✓	increment top of stack by amount Q
IND	✓		✓	indexed fetch, displacement is Q—see note 4
INN				test for membership (*in*)
INT				set intersection
IOR				perform Boolean 'inclusive or'
IXA			✓	compute indexed address—see note 5
LAO			✓	load base-level address (Q is displacement from stack base)
LCA			✓	load address of constant in Q
LLA		✓	✓	load address—see note 3
LDC	✓		✓	load constant given in Q field
LDO	✓		✓	load contents of base-level address (global variable)
LEQ	✓		★	test for less than or equal to
LES	✓		★	test for less than
LOD	✓	✓	✓	load contents of address—see note 3
MOD				modulus
MOV			✓	moves the number of storage units given by Q
MPI				multiply integers
MPR				multiply reals
MST		✓		mark stack—see note 6
NEQ	✓		☆	test for not equal
NGI				negate integer
NGR				negate real
NOT				perform Boolean 'not'
ODD				test for odd
ORD	✓			convert to integer
RET	✓			return from block
SBI				perform integer subtraction
SBR				perform real subtraction
SGS				generate singleton set
SQI				square integer
SQR				square real
SRO	✓		✓	store at base-level address (Q as in LAO)
STO	✓			store indirect
STP				stop
STR	✓	✓	✓	store—see note 3
TRC				truncate
UJC				error in case statement—abort
UJP			✓	unconditional jump to label given by Q
UNI				perform union of sets
XJP			✓	indexed jump; jump to Q + top of stack

FIGURE 5 P-code Instructions