DOCUMENT RESUME

ED 093 377 IR 000 884

AUTHOR Kheriaty, Larry
TITLE Report on a Mini-Computer Based Interactive Computing System.
INSTITUTION Western Washington State Coll., Bellingham.
PUB DATE 74
NOTE 6p.

EDRS PRICE MF-$0.75 HC-$1.50 PLUS POSTAGE
DESCRIPTORS *Computer Assisted Instruction; *Computer Programs; *Computers; Programing Languages
IDENTIFIERS *Minicomputers; PILOT; Western Washington State College

ABSTRACT

In spite of the limitations of minicomputers, Western Washington State College (WWSC) has developed a useful interactive computing system based on a Model 7/32 Interdata minicomputer and the computer program, PILOT. The major disadvantages of minicomputers are the difficulty of securing maintenance and the reliance often on the single language, BASIC. Software systems are crucial and languages such as BASIC or APL are ill-suited for course-authoring for computer-assisted instruction (CAI). Cost, of course, is the principle advantage. The system at WWSC relies on a multilingual interpreter to execute programs in a constant pattern for all languages, once translated into an internal code of single byte operands and operators. PILOT has great structural simplicity but yet great versatility, and perhaps its best feature is its ability to call an already compiled BASIC subroutine. A program was written which can translate COURSEWRITER III to PILOT. The experience of WWSC indicates that a CAI system can be satisfactorily run on a minicomputer if there is access to good software. (WH)

REPORT ON A MINI-COMPUTER BASED INTERACTIVE COMPUTING SYSTEM

L. Kherlopian, Systems Programmer, Computer Center, Western Washington State College

INTRODUCTION

There are two main thrusts to this paper. The primary intent is to introduce you to the mini-computer based terminal system at WWSC, as the title suggests. Before I do that, however, I feel this is a good opportunity to comment on the role of the mini in interactive computing. Since our terminal system has been in the process of conversion from a 360/40 base to an Interdata mini base over the past six months, we have some immediate experience to draw on for this discussion.

THE ROLE OF THE MINICOMPUTER IN INTERACTIVE COMPUTING

First I would like to state categorically that minicomputers are not the answer to all of our computing problems. It is a mistake to think that a five-thousand dollar processor can do the job of a large-scale computer. Some of the problems associated with minicomputers are not unique to the mini. They can pertain to any computer equipment when a switch is made from established vendors, such as IBM, to a relative newcomer in the field of manufacturing computer hardware.

The response time when maintenance is needed has been a problem, partly due to Western's 'remote' location and partly due to the relatively small size of the manufacturer's back-up organization. For instance, we have had to ship parts of an Ann Arbor terminal to Michigan for repairs and replacement. Also the interface between one manufacturer's equipment and another's may not function too well, particularly if your installation is a 'pioneer'. We spent four months working out hardware bugs in the interface between our Interdata and the 360/40. On the other hand, the Ann Arbor and the Texas Instruments terminals were installed with nothing more drastic than a minor plug change.

Far and away the most serious drawback to minis, however, is the lack of available software of any kind, but particularly software for CAI systems. There are few CAI systems available at all, and those that are available off the shelf are primarily one-language systems, usually BASIC. Many educational installations do not want to limit their terminal systems to one language, as that imposes restrictions on the types of applications that can be mixed on the same system, i.e., CAI, interactive programming, remote file maintenance, to name a few. In addition, BASIC has never really established itself as an adequate CAI course authoring language.

Since there are no widely accepted CAI systems for minis, it is very difficult, if not impossible, for one installation to exchange courses with another. This is a serious drawback as you probably know, because of the time and effort required to develop really good courseware.

It is obvious that what is needed is a software system that is compatible with many types of hardware and is capable of handling at least two languages, one of which is a CAI author language, and the other an acceptable problem-solving language. We cannot stress enough our conviction that no one-language system is sufficient to meet the needs of course authors, even when no use other than CAI is made of an interactive system. Existing systems based on a computational language such as APL or BASIC, in addition to being ill-suited to course authoring, are too expensive of core for most users. The languages suited especially to course authoring are too restrictive for the development of certain types of courseware.

In spite of the potential problems of mini-based CAI systems, there is still one overriding reason to try to make them go. That reason is the cost of hardware. Large mainframe systems that are widely accepted for CAI are being abandoned by many users because they are too expensive to run. The choice is either to diminish the size and capability of the interactive computing system or to switch to less expensive equipment. For instance, at WWSC it became clear that CAI had to be done more cheaply or not al all. Our main computer had become overloaded with batch work and the prospects for upgrading it in the near future were not good. We found that a rather modest investment in minicomputer hardware would give us enough power to run about sixteen student terminals. In fact, the cost of computer power for running between sixteen and thrity-two terminals on such a system is about three-thousand dollars per terminal. This price does not include disk storage, which we already had. It can cost anywhere from 10,000 to 20,000 dollars to provide disk storage for a system fo 16 to 32 terminals, depending on the exact amount of disk space required and the brand purchased. Neither does it include the cost of the terminals, which ranges anywhere from one thousand to three thousand dollars each.

The most significant advantage of a minicomputer based terminal system is that relatively low cost can be obtained without going to a huge system. On the other hand, if many more terminals are needed one can readily expand by simply adding on duplicate systems in groups of up to say a maximum of 32 terminals.

There are two inherent advantages to such a method. The first is that the built-in redundancy of systems reduces the possibility of catastrophic results when the occasional, but inevitable, system crash occurs. Secondly, the smaller systems can be dispersed so that the processor and the terminals are grouped close together. This not only reduces costly telecommunications links, but there is a great psychological advantage for the user. It is our experience that the further the computer is from the terminal the more frustration is felt by the user when he has trouble with the system. An on-site computer installation gives the user someone to turn to when difficulties arise. The importance of having human communication between the remote user and the central installation cannot be emphasized too much.

THE MULTILINGUAL INTERPRETER AT WWSC

To turn now to a more specific look at the system we developed at WWSC, we have constructed a system based on a Model 7/32 Interdata minicomputer. This machine is capable of having a million bytes of core storage and has an instruction execution speed approximately equivalent to that of an IBM 360/50. We will be connected through our 360/40 to our main main disk system for storage of programs and data, but a disk could be attached directly to the system if desired.

The system runs under the control of a time sharing monitor. This monitor, or operating system, handles all input/output to the terminals and to the disk storage, whether it is on-line or through the main computer. The language processor itself is based upon a general (or multilingual) interpreter. This interpreter is capable of executing programs that have been translated into an internal code made up of single byte operands and operators. Operator routines are included to perform all functions that are needed to execute any of the languages in the system. Many of the languages share operator routines. For example, the sum of A + B is calculated by the same routine regardless of the language fo the original statement. In sum, the general interpreter is the workhorse of the system.

The programs that prepare work for the interpreter are called front-end translators. One front-end is provided for each high-level language in the system. They convert the respective source statements into a standard interpretive code which can then be executed by the general interpreter. When a program is to be run, it is written to a disk in source form. It is then read from the disk, processed by the appropriate translator, and

converted to interpretive code. At this point, the program can either be executed by the interpreter, or it can be written onto disk for later execution. This latter option means that programs that are to be run many times can be saved in compiled form and used over and over. This cuts dramatically the time required to execute programs, as they are executed immediately after being read from storage.

Our original terminal system, which was based on the 360/40, used a similar general interpreter capable of handling COURSEWRITER III, BASIC and a locally written subset of PL/1 called WPL[1]. We are now moving to a system which combines, for the time being, BASIC, with which you are already familiar, and PILOT, a course authoring language which is relatively new.

Our first introduction to PILOT came from a two-page article by Dr. Sylvan Rubin of the Stanford Research Institute published in the November 1973 issue of Computer Decisions.[2] The article describes the language itself and how it came into being. The beauty of PILOT is that it has a very simple syntax and can be readily implemented on virtually any computer. At WWSC we have written our compiler for PILOT in Interdata Assembler language. According to the article, PILOT compilers have been written in BASIC, FORTRAN, SNOBOL, APL, and PL/1 in addition to various machine-dependent assembler languages. All use the same user notation for the instruction set, and a standard type of notation for local extensions of the language. For the benefit of those among you who may not be familiar with PILOT, I would like to describe the language briefly. The fact that the entire language can be described in a two-page article says a great deal about the structural simplicity of PILOT, but that is not to say that it lacks versatility.

Only four op codes are required to begin coding a course in PILOT: T: for type text; A: for accept answer; M: for match answer; J: for jump or branch. Each of these op codes is followed by a text field. In addition, any of these four op codes can be made conditional in one of two ways. Coding the letter 'Y' after an op code (for example, TY:) causes a statement to be executed only if the last answer to a match was successful. Conversely, appending the letter 'N' to an op code causes a statement to be executed only if the previous match failed. A statement can also be made conditional on the value of an expression coded within parentheses after the op code (for example, A(max=min):). Such expressions should be coded in BASIC notation.

Four more standard op codes for PILOT greatly enhance the language. The first three are R: for remark or comment; U: for use subroutine; E: for end subroutine. The fourth and most powerful is C: for compute. This op code is followed by either an expression or by a CALL to a subroutine. (At WWSC, BASIC notation and subroutines are used.) This feature allows all the computational capabilities of BASIC to be embedded within a CAI course.

Some other features of the language include the availability of statement labels, identified by an initial asterisk (*) and placed immediately before an op code (e.g., *STEP T:). The label can then be used in the text field of a J statement. Variable names preceded by a dollar sign (e.g. $NAME) can be coded in the text field of an accept answer statement. This will cause the student's answer to be stored in that variable. Such answers can be either numeric or alphabetic or mixed. The variable name can then be placed within the text of a T statement. When that variable name is reached by the program, the stored answer will print in its place, adding a personal touch to student-terminal interaction.

There are other features of PILOT that greatly enhance programming ability. Without going into detail, there are statements for course segmentation and linkage; extended pattern matching with tolerance for common spelling errors; statements which allow line graphics within a CAI unit; the ability to control a random access slide projector.

Perhaps the greatest advantage to the link between BASIC and PILOT at WWSC is the ability to CALL already compiled BASIC subroutines from a PILOT course. By eliminating the need for compiling a subprogram each time it is called, many CAI courses can be made available that are too costly in response time on systems which lack the compile feature.

We believe that our system meets the following needs. 1) It is implemented on a relatively inexpensive processor with no built-in necessity for costly telecommunications charges. 2) It is versatile in that the user is not unnecessarily limited in his choice of high-level languages. A new language can be added to the system without modification of existing software and even without significant service interruptions. 3) The general interpreter gives the system the advantages of a single language system. 4) It provides CAI authors with a language which is easy to learn. But authors who wish to do more sophisticated CAI programming are not bound by its simplicity, since they have easy access to both the advanced features of the author language and to a problem-solving language.

It is impossible to ignore the problem of translating existing courseware when a shift to a new CAI language is made. In our situation we solved the problem by writing a program which translates COURSEWRITER III to PILOT. This took approximately 80 - 100 man hours of work. We estimate at the present time that we obtain about 90% complete conversion. That means, that 90% of the statements in a COURSEWRITER III course will be successfully converted without further human intervention. Some courses convert with no change whatsoever. The worst case so far was only 60% successful. We are also aware of the existence of other translators to and from PILOT.

CONCLUSIONS

In summary, we feel that it is possible to run a CAI system on a minicomputer which is both cost effective and meets most of the needs of potential authors. On the other hand we feel that the probability of success is directly related to access to good software. Since many installations lack the human resources to develop their own software we recommend extreme caution before making a move to a mini-based system until it is certain that software can be obtained which is adequate to meet the needs of the system users.

We considered making some suggestions in the direction of establishing standards for CAI software, but are discouraged by previous experience. The climate does not yet seem ripe for such a move. We are not attempting to push our own system as a standard, but are convinced that it is a step in the right direction and are willing to share it with anyone who is interested.

BIBLIOGRAPHY

1) Since there are many similarities between the new and old systems, users who are particularly interested are referred to a paper presented at the ACM Annual Conference in 1973. L. Kheriaty, "A Multilingual Interpreter for Interactive Computing in an Academic Environment", ACM Proceedings, 1973, pp. 290-294.

2) S. Rubin, "A Simple Instructional Language", Computer Decisions, Nov. 1973, pp. 17-18.