DOCUMENT RESUME

ED 112 840 IR 002 540

AUTHOR Brahan, J. W.; Colpitts, B. A.
TITLE NATAL-74; Towards a Common Programming Language for CAL.
INSTITUTION National Research Council of Canada, Ottawa (Ontario).
PUB DATE Aug 75
NOTE 11p.; Paper presented at the Association for the Development of Computer-Based Instructional Systems Summer Meeting (Portland, Maine, August 5-7, 1975)

EDRS PRICE MF-$0.76 HC-$1.58 Plus Postage
DESCRIPTORS Computer Assisted Instruction; *Computer Programs; Curriculum Design; Program Descriptions; *Programing Languages
IDENTIFIERS CAL Programing Languages; Canada; *NATAL 74

ABSTRACT

NATAL-74 is a programing language designed for Canadian computer aided learning (CAL) programs. The language has two fundamental elements: the UNIT provides the interface between the student and the subject matter, and the PROCEDURE element embodies teaching strategy. Desirable features of several programing languages have been adapted to cope with a wide range of display equipment. A variety of computational capabilities, including a calculator mode, provide flexibility in use and response processing. A major goal of NATAL-74 is to provide an effective means to exchange courseware programs. The implementation phase has initially used the DEC-10 computer, but is working toward a high level of machine independence. Cooperation and continuing dialog between CAL users, vendors and researchers is necessary to achieve a meaningful standard for a CAL Language. (CH)

NATAL-74

--

TOWARDS A COMMON PROGRAMMING LANGUAGE FOR CAL

J.W. BRAHAN and B.A. COLPITTS
NATIONAL RESEARCH COUNCIL OF CANADA

Presented at
ADCIS Summer Meeting
5-7 August 1975
Portland, Maine

NATAL-74 -- Towards a Common Programming Language for CAL

J.W. Brahan and B.A. Colpitts
National Research Council of Canada

Introduction

The cost of preparation and evaluation of Computer-Aided Learning (CAL) materials is high. If these costs are to be justified, the materials must be available to large numbers of potential users and capable of being used at a variety of installations.

There are a number of CAL projects underway at centres throughout Canada, yet little exchange of course materials takes place. This is due in part, to the barrier set up by the variety of programming languages currently being used by these centres. Exchange of materials is difficult and time consuming when it necessitates reprogramming.

Current CAL work employs a variety of techniques which include: frame-oriented CAI, adaptive-tutorial, learner-control, simulation and gaming, computer-managed instruction and techniques of artificial intelligence. Student terminals vary from simple typewriter-like devices to multi-media terminals incorporating audio-visual display and graphic input capabilities. CAL centres, to meet their particular requirements, have modified vendor-supplied CAL languages, developed their own, or have made use of high-level programming languages not specifically designed for CAL applications. To date, no one language has established itself as being nationally acceptable. A given language is either not powerful enough to satisfy the majority of user needs, or it places severe restraints on either the computer or the terminals which can be used.

The need seems clear for a language which may be used effectively on a variety of computers, with a variety of terminals, and which will provide the features demanded by the various CAL techniques currently in use.

The Approach

The National Research Council (NRC) makes extensive use of Associate Committees to study, coordinate, and promote research on problems of national significance. The members of these committees are experts in the different aspects and disciplines related to the problem and are drawn from university, industry and government laboratories.

The Associate Committee on Instructional Technology held its inaugural meeting in 1970. The members of this committee come from across Canada and are experts in the various disciplines related to instructional technology. Very early in its deliberations, the Committee noted the problems being created by the lack of a uniform programming language for CAL applications. To investigate a solution,

the Committee formed a Working Panel on CAL Languages whose members came from centres throughout Canada where they were actively engaged in CAL projects.

The Working Panel was assigned the task of defining the characteristics of a programming language for computer-aided learning applications, with particular emphasis being placed on satisfying requirements peculiar to Canada. In their deliberations, the panel members drew on their personal experience with a variety of languages in a number of application areas. In addition, the panel relied heavily on the 1969 EDUCOM report of Zinn, using his "aspects" for comparing programming languages as the basis for the definition of language requirements. The report of the Panel was presented in 1972 in the form of a functional specification.

The next phase of the operation entailed the preparation of a detailed specification to define the fine structure of the language. A Subcommittee on CAL Programming Languages was formed to carry out this task. Early in 1973 the Subcommittee issued a call for tenders for the definition of a programming language to meet the requirements as stated in the functional specification. Prospective bidders were invited to attend a briefing at which presentations were given by Subcommittee members describing the CAL activities at their centres. The intent of the briefing was to provide each bidder with an indication of the extensive range of CAL applications and the milieu in which the CAL language would be used, as an aid in interpreting the functional specification. It was emphasized that the detailed specification should be based in whole, or in part, on a currently existing language if that language satisfied many of the requirements stated in the functional specification.

Late in 1973, a contract was awarded to IBM Canada Ltd. for the development of the detailed specification. This contract was completed in 1974 and an Author Guide and a Specification Manual for NATAL-74 have been published in both English and French.

The Language

The language defines two fundamental structural elements - the UNIT and the PROCEDURE. The UNIT, the immediate interface between the student and the subject matter, includes extensive features for accepting and processing user input and for controlling the presentation of information. It may be thought of as accomplishing one complete instructional transaction. The PROCEDURE embodies teaching strategy; it assembles information required by the UNIT, evaluates information generated by a student response, and controls the sequence of instruction. The selection and invocation of UNITS is under PROCEDURE control. One PROCEDURE may call other PROCEDURES, allowing a powerful hierarchy to be constructed. In addition to the two basic structural elements, a variety of function types are defined. These functions provide a means of standardizing and attaining ready access to operations which support the execution of UNITS and PROCEDURES.

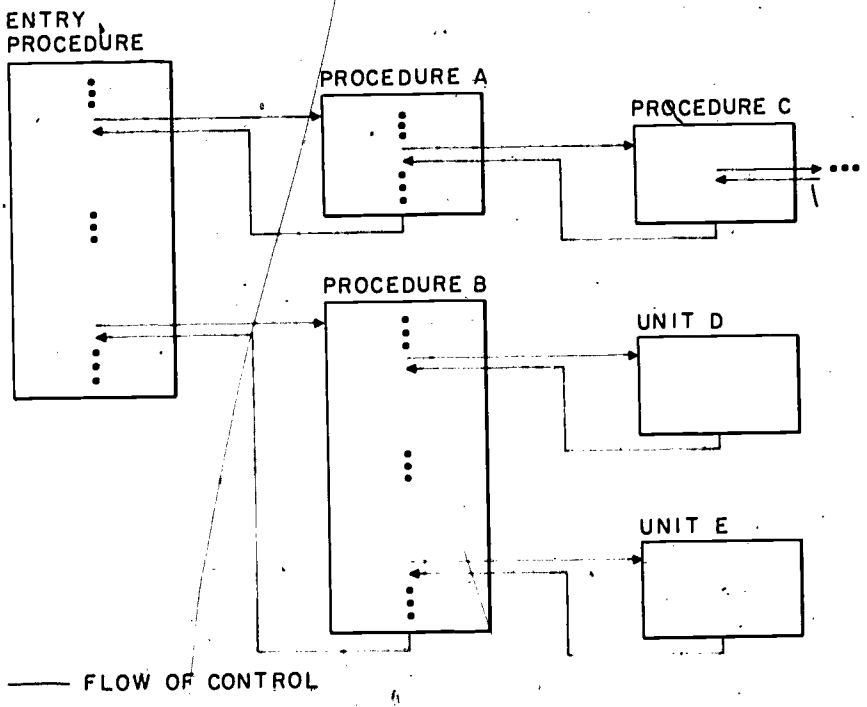

FIG. 1 NATAL-74 CONTROL STRUCTURE

Desireable features have been adapted from a number of programming languages. The dynamic variable types of APL are combined with the power of the control structures of PL/1. Organization of statements around natural language keywords supports clarity and readability. A brief description of some of the features of the language may impart some of its character.

The UNIT incorporates facilities for coping with a wide range of display equipment, satisfying a spectrum of needs from straightforward presentation of text to sophisticated graphic displays. Equally powerful are the facilities for handling user input. Within the UNIT a response is accepted, edited and categorized. Input conditions such as restrictions on elapsed time or the number of characters to be accepted may be set. The language provides a varied list of edit functions which the user may supplement with his own. Categorization functions allow comparisons to be made on several bases. They include algebraic, phonetic, and keyletter comparisons and may be added to by the user.

In addition to the extensive input-output features, good computational capabilities are provided in the language. System-supplied functions include, in addition to standard functions such as square root, absolute value, and trigonometric functions, a range of string manipulation functions to provide flexibility in processing student responses.

System variables automatically provide the author with useful information about a student's progress in a course. For example, latency of last response, minutes this course, and number of unrecognized responses are a few of the data items available should the author choose to access them. The potential for on-line adaptation based on the student's progress exists. Special variables designated "course variables" are accessible by all students within a course.

System PROCEDURES provide some interesting capabilities. One is a facility for placing the system in a calculator mode, permitting the student to perform simple arithmetic computations on variables made available to him by the author. Answers are automatically returned to the lesson when the student signals he has finished with the "desk calculator" facility and control returns to the program. Another facility permits invoking external compilers and interpreters. Upon completion of the desired operations, control again returns to the NATAL-74 program.

Statements in the language are identified by natural language keywords, suggestive of their function, such as DISPLAY (AFFICHER), EDIT (EDITER), and REPEAT (REPETER). Reflecting the Canadian scene, the keywords may be in English or French as desired. Comments can be integrated into the code to enhance its readability.

Two examples of a UNIT have been provided to indicate the flexibility which is possible. Even in its simplest form, a UNIT can interact with a student in a wide variety of ways, dependent on the student's responses. Figure 2 illustrates the chaining feature, whereby reprise statements appearing as a block for any particular category are linked together. The system "remembers" which reprise statements within a chain have been previously executed, and does not repeat the execution of any reprise statement unless it is the last in the chain. In following through the example note that the RETRY statement presents text and branches back to the RESPONSE statement. The REINF statement, on the other hand, presents text and returns to the calling PROCEDURE.

In figure 3, the problem presented to the student is now dependent on the argument passed in calling the UNIT and control is exercised over the format of the display through the use of additional display sub-language commands. The text presented to the student includes variable values, and the student response is compared with appropriate variable expressions rather than with predetermined constants. Also, the original statement of the problem will be maintained on the upper half of the display screen throughout the execution of the UNIT. The student's response begins at the point on the screen designated by the author. Text presented by a reprise statement will appear in the lower half of the screen. Whenever a student is asked to try again, his previous answer will be erased and the cursor repositioned so that his new answer will appear in the same place as his previous answer. As well, the student response may be timed out in this example and reprise statements added for the category OTIME.

Editing, categorization, and reprise statements may be combined in whatever order the author wishes. In addition to the system-supplied edit and comparison functions, the author may invoke his own functions. Thus the instructional transaction performed by the UNIT can be highly complex or very simple as required. When the flexibility of the UNIT is considered in relation to the processing capabilities of the PROCEDURE, the language is seen to be capable of supporting very sophisticated CAL techniques.

The Implementation

The third phase of the project, implementation, commenced in 1974 with the delivery of the detailed specification. As part of the CAL Research Project of the NRC Laboratories, NATAL-74 is being implemented on a DECsystem-10 computer. The language will be available to network members as soon as it is operational so that they may provide the feedback necessary to the testing and revision process.

The major goal of the project has been to develop a language which will be widely available and will provide an effective means for the exchange of courseware programs between centres. Thus the work in the current phase has two clear objectives: a working implementation on the DECsystem-10 at an early date and an implementation which can be transferred to other machines with minimal effort. In an attempt to achieve a high level of machine independence a number of possible implementation languages were examined with the following being considered desirable features:

- Good control structure
- Good data structuring capability
- A variety of data types
- Ability to control features of host operating systems
- Flexibility in control of student terminals
- Generation of efficient code
- No requirement for a large run-time support system
- Available on the DECsystem-10
- Available on a variety of other computers

The languages which were considered were: APL, BCPL, BLISS, FORTRAN, PASCAL, PL/1. From these, while it does not meet all of the requirements perfectly, BCPL was chosen as the one which provided the "best fit" and is being used for most of the implementation.

The implementation has been divided into five main components: a Course Builder, a Preprocessor, an Interpreter, an Executive and a Terminal Handler.

The Course Builder provides the interface between the author and the Preprocessor. It allows him to assemble modules (PROCEDURES, UNITS and FUNCTIONS) into courses, to add modules to courses, and to replace existing modules. In addition, the Course Builder carries out the function of linking the various modules in a course and provides the file management associated with the manipulation of source and processed files.

The Preprocessor checks the syntax of the NATAL-74 source code, allocates storage and generates an intermediate language code which can be more efficiently interpreted at run-time than the source code. The Preprocessor is in essence a compiler which carries out all processing which can possibly be done prior to execution. The Course Builder and Preprocessor play their role during course preparation while the remaining three elements make up the run-time system.

The Interpreter executes intermediate language code, communicating with the student through the Terminal Handler. Access to shareable resources such as files and course variables is through the Executive. Memory management operations required to provide on-line continuity for the user from one transaction to the next are handled by the Interpreter.

The Executive manages the shareable resources available to the user. Space is allocated for course materials and access is provided to course variables and files so as to eliminate possibilities of conflict between users. The Executive also controls student access to the system and courses.

The final, but by no means least important, component of the implementation is the Terminal Handler. This component processes display sub-language command-strings and adapts the output to the parameters of the particular student terminal. It controls timing of display and input operations and transforms input into "normal form" so that processing is, insofar as possible, independent of terminal characteristics.

Machine independence is a characteristic of the implementation design which is given priority. The degree to which it can be achieved, however, is limited by practical considerations such as implementation effort and run-time efficiency. The Preprocessor and the Interpreter are to a large extent machine independent and it should be possible to transport them to another machine with only minor modifications. The Course Builder exploits the file handling system of the host computer and thus would require somewhat more effort to transfer. Similarly, the Terminal Handler relies on the communication facilities of the host and incorporates a number of machine-dependent routines. Techniques used to describe terminal characteristics for example are about fifty percent machine dependent. Finally, the Executive is almost entirely machine-dependent. Because of it's close ties to the host operating system, it has been written in assembly language.

While complete machine-independence has not been attained and is in fact not a practical goal, a level has been achieved which will permit transfer of the initial NATAL-74 implementation without major changes in the overall design and with a large proportion of the code intact.

Summary

Considerable progress has been made since the first meeting of the Working Panel on CAL Languages in 1971. A functional specification for a CAL language and an initial detailed specification have been completed and published. Work on the implementation is well advanced with the first elements of the language expected to be operational by November, and all major elements by March 1976. Much remains to be done, however, before a useful standard is achieved.

The specification of NATAL-74 at the present time must be considered as preliminary and subject to revision. It is anticipated that implementation and application testing by potential users will result in changes. Only when a language is available to users and has been demonstrated to meet their CAL requirements can it be considered as a possible standard.

Emphasis has been placed on co-operation among CAL users throughout the development of NATAL-74. The project could not have reached its present stage without the active participation of the members of the Working Panel and the Subcommittee on CAL Programming Languages. The project has also benefited from the comments and suggestions of CAL experts from a number of centres in the United States and elsewhere.

If the work is to reach a successful conclusion, there must be a continuing dialog between CAL users, equipment vendors and research groups. Without such co-operation, there can be little hope of achieving a meaningful standard in any field.

References

ACIT Working Panel (1972) A Functional Specification for a Programming Language for Computer-Aided Learning Applications. Report No. NRC-13659, Associate Committee on Instructional Technology, National Research Council, Ottawa.

Richards, M. (1969) BCPL: A Tool for Compiler Writing and System Programming. Proc. Spring Joint Computer Conference, Boston.

Westrom, M.L. (1974) National Author Language NATAL-74 Author Guide. Report No. NRC-14243, Associate Committee on Instructional Technology, National Research Council, Ottawa. (Aussi disponible en francais)

Westrom, M.L. (1974) National Author Language NATAL-74 Specification Manual. Report No. NRC-14245, Associate Committee on Instructional Technology, National Research Council, Ottawa. (Aussi disponible en francais)

Zinn, K.L. (1969) Comparative Study of Languages for Programming Interactive Use of Computers in Instruction. EDUCOM Research Memorandum RM-1469.

```
                    *PROB#10:
UNIT;

DISPLAY         &N In how many ways can 5 different books be
                arranged on a shelf so that two particular
                books are next to one another? &;

RESPONSE;

EDIT    NUMBR;  /* Extract numeric fields from answer. /*

RIGHT   CN(48);
REINF   RIGHT   &2L Good! Thinking of the two books as a unit,
                in which either book may be first, is the key.&;

F0      CN(24);
RETRY   F0      &2L Close! Treating the two books as a single
                item is the right idea, but the two books may
                be placed side by side in more than one way.
                Try again. &;
REINF   F0      &2L There are two ways of placing the two books
                side by side. Thus the total number of arrange_
                ments is 48. &;

F1      CN(120);
RETRY   F1      &2L You've stated the number of ways 5 books
                can be arranged on a shelf. You forgot that
                two particular books must be adjacent.
                With that in mind, try again. &;

F2      CN(5);
RETRY   F2      &2L That's just the number of books.  You can
                give me a better answer than that! &;

F3      CC(' ');
RETRY   F3      &2L Please give a number written as a string
                of digits for your answer... &;

RETRY   UNREC   &2L I don't know how you arrived at that
                number.  Break the problem down into subproblems
                and give it another try. &;
RETRY   UNREC   &2L Sorry, I don't follow your reasoning. Here's
                a hint.  Imagine tying the two books which must
                be adjacent together.Think of how many ways this
                could be done. With the two books tied together,
                the number of items to be arranged is reduced to
                4... Now try the question. &;
REINF   UNREC   &2L The two books which must be adjacent can be
                "tied together" in 2 ways. There are 4!=24
                ways of placing 4 different items in a row. Thus
                there are 2*24=48 ways of arranging the 5 books
                with two particular ones next to each other. &;

END     *PROB#10 ;
```

FIG. 2

```
                    *PROB#10:

UNIT (N); /* N is to be an integer between 4 and 8 /*

DISPLAY         &N Suppose &V(N,1,0) different books are to
                be arranged on a shelf so that two particular
                books are next to one another.
                &L The number of possible arrangements is: &;

RESPONSE        TIME=60,POSN=(4,40) ;

EDIT    NUMBR;  /* Extract numeric fields from answer. /*

RIGHT   CN(2*FACT(N-1));
REINF   RIGHT.  &B34&E Good! Thinking of the two books
                as a unit, in which either book may be
                first, is the key. &;

FO      CN(FACT(N-1));
RETRY   FO      &B34&E Close! Treating the two books as
                a single item is the right idea, but the
                two books may be placed side by side in
                more than one way. &W2 Try again.
                &E(4,40,1,50) &;
REINF   FO      &B34&E There are two ways of placing
                two books side by side. Thus the total number of
                arrangements is &F(2*FACT(N-1),6,0) . &;
```

FIG.3