The Modula-2 Experience

Nan C. Schaller
Undergraduate Computer Science Department
Rochester Institute of Technology
Rochester, New York 14620

ABSTRACT

The 1987-88 school year represents the first time that the Undergraduate Computer Science Department at Rochester Institute of Technology (RIT) has offered its five quarter course programming skills sequence with Modula-2 as its primary teaching language. What follows is a description of RIT's first year Modula-2 experience including the trials and tribulations of new languages, new compilers, and untried texts. With only the first half of the sequence having been offered using Modula-2, the benefits derived from the change thus far will be discussed as well as suggestions, conclusions, and a preview of what is yet to come.

INTRODUCTION

During the winter quarter of 1986, the RIT Undergraduate Computer Science Department decided to change from Pascal to Modula-2 as the primary teaching language for its programming skills sequence. The department was motivated to do so because Modula-2 overcomes a lot of the deficiencies of Pascal, particularly in the areas of data abstraction and separate compilation. Modula-2 provides additional benefits with the inclusion of generic data types, open array parameters, and with improved syntax of control structures.

The first freshman class (7 sections) to start the programming skills sequence using Modula-2 began in the 1987 fall quarter. The programming skills sequence is a series of five quarter courses: Algorithmic Structures (Introduction to Programming), Data Structures, Assembly Language, Program Design and Implementation, and Data Organization and Management (better know as "DO(O)M" by the students).

Currently, the first Modula-2 class is finishing up their assembly language course. What follows is a description of RIT's first year Modula-2 experience.

START-UP

The first step in implementing the change to Modula-2 was to locate a suitable, inexpensive compiler. In the fall quarter, the First Year Programming Laboratory was equipped with 39 seats (terminals connected to a DEC Vax 11/780 running Unix) and 6 dialup lines. The compiler chosen was the Powell experimental compiler ("mod" from DEC) . This choice was dictated by ready availability and cost. The cost was particularly important as an upgrade of the First Year Programming Laboratory was imminent.

The Powell compiler presented some difficulties. It relies on C input and output. The faculty teaching the first course felt that this was difficult conceptually for novice programmers, particularly since all available textbooks discussed and utilized the InOut and RealInOut modules for beginning I/O. To overcome this problem, InOut and RealInOut modules were written locally and incorporated into the Modula-2 library. The compiler also has limitations in the way in which it deals with CARDINALS, range checking, and definition modules. In addition, several other idiosyncrasies were discovered. Many times it was difficult to determine whether a particular "discovery" was attributable to the compiler or the chosen text, which had turned out to be riddled with typos and other errors.

The Undergraduate Computer Science Department feels very strongly that a student in the programming skills sequence should receive the same course material regardless of which professor's class the student is attending. To this end, all of the faculty involved with a particular course in the sequence typically meet on a weekly basis to discuss the course topics for the week, common programming assignments, discoveries, surprises and problems. This has proven to be an invaluable mechanism to insure common experiences for the students. It also helped to insure the sanity of the participating faculty, especially with the change to a new language, a new, unproven text, and a new compiler.

FIRST COURSE RESULTS

The experience with the first course (Algorithmic Structures) was mixed. The first course, since it is

presented in a quarter rather than a semester, only covers basic language constructs such as control structures, procedures, sets, user-defined types, and arrays. It also addressed the concept of modules. For the basic language constructs covered, there was not a big difference from those same constructs in Pascal. A lot of students had Pascal experience from high school and, consequently, felt like they had not learned much. They also did not like Modula-2's "standard" I/O modules which are cumbersome and wordy.

On the other hand, the structured programming concepts were as easy to get across using Modula-2 as they had been using Pascal. The concept of modules, although somewhat more difficult for the novice to grasp, established the ground work for the other courses in the sequence.

THE SECOND COURSE

Five sections of the data structures course were offered for the first time in Modula-2 during the 1987-88 winter quarter. This quarter proved to be particularly interesting. The equipment to upgrade the First Year Programming Laboratory arrived during the first week of the quarter. The department had received funding for 30 monochrome Sun 3/50 workstations and the Sun Modula-2 compiler. Because of the periodic extreme system overload experienced with the Vax configuration in the First Year Programming Laboratory, the department decided to proceed with the Sun workstation installation over the two week Christmas break. This decision was made with some trepidation as this was well into the winter quarter, ie., just before the start of the fourth week. This meant that the faculty as well as the students had to be brought up to speed on the workstations and the new compiler as rapidly as possible. Faculty and laboratory assistant training started before the Christmas break using the Sun workstations already installed in the Computer Graphics Laboratory. Student training was handled in the First Year Programming Laboratory during class time. Students were given exercises which introduced them to the workstation capabilities, the new Modula-2 compiler and dbxtool, a window-based, interactive, source-level, symbolic debugger. A summary of the Modula-2 compilers' differences was composed and made available. The overall installation went amazingly well and, as was anticipated, the students' productivity has increased greatly.

Modula-2 started to pay off with the data structures course. Programming assignments were constructed to demonstrate the value of data abstraction to the student. A typical programming assignment for the data structures course has at least two parts. The first part requires the student to write a program which utilizes a particular data structure(s). The faculty member provides the student with the definition module(s) for the data structure(s) and a mechanism by which the student's program can be linked with the corresponding faculty implementation module(s). The second part of the assignment requires the student to write his/her own version of the data structure implementation module(s) using a specific type of implementation. For example, they might be expected to implement a stack using a record and an array. It is necessary for the student's data structure implementation module(s) to execute successfully with both the faculty data structure testing program and with the student's user program from part 1. The next programming assignment then typically requires that the student implement the same data structure(s) in yet another manner, for example, with pointer variables, using the same definition module(s). This implementation module(s) must again execute successfully with both the faculty data structure testing program and with the student's user program from the previous assignment.

SECOND COURSE RESULTS

The use of Modula-2 in the data structures course has been instrumental in teaching students about data abstraction. Some students find it very hard at first to separate the utilization of a data structure from the implementation of the data structure. Others are quite amazed to find that they do not need to alter their user programs when implementing a data structure in a different way. Modula-2 facilitates getting these ideas across to the student.

It should also be mentioned that the Sun Modula-2 compiler, although it has some documented idiosyncrasies of its own, performed much better in general than the Powell compiler had. In addition, the source level, symbolic debugger, dbxtool, which can be used with Sun's Modula-2 compiler, proved useful in increasing the students' productivity.

CONCLUSIONS

So far, the move to Modula-2 has been a positive experience. Modula-2 does overcome some of the worst deficiencies in Pascal. Pascal requires that the data type of an identifier be fully known at all times. This makes it possible for the user to misuse a data structure. For example, if a data structure is constructed with an array, in Pascal the user could legally access a piece of information from the middle of the array whether or not it was currently an active element in the data structure. Modula-2 alleviates this problem through the use of two mechanisms. First, the definition and implementation module structure provides an isolated location for the data structure and its accessing functions. The user only has access to the information in the definition module: documentation, constants, global variables, data types, function and procedure names and parameters. Second, Modula-2 permits the use of an opaque data type. In

this case, the definition module contains only the name (not the implementation) of the data type. The implementation of that data type is specified in the implementation module. The user typically does not have access to the source code of the implementation module. Therefore, if a data structure is implemented using an array, the user will not know how it is implemented and will only be able to access the data structure through its accessing functions.

The isolation of the data structure in its own module provides a mechanism through which the student can realize the ease with which an implementation can be changed without the user program being altered. In addition, it makes the case for separate compilation an easy one to present. Separate compilation was cumbersome at best with Pascal since Pascal was not originally designed with that concept in mind. With Modula-2 it is quite natural to group various procedures into separate modules. One can easily see that once compiled, it is not necessary to recompile these modules unless they are modified in some way. It is only necessary to link these modules with the user modules.

The case for writing reusable procedures is also reinforced with the concept of separately compiled modules. The user has contact with this idea from the very beginning use of Modula-2. Even the input and output is handled with separate, already compiled, modules.

The availability of open arrays and generic data types also contributes to the reusable procedure concept. Open arrays are arrays of a particular data type without regard to length or subscripts. This means that an array A can be specified in a procedure as an ARRAY OF CHAR. Its subscripts will range from 0 to HIGH(A). This procedure can then be used with any singly dimensioned array of type CHAR, no matter how the actual array parameter was actually originally declared. Generic data types takes this a step further. The generic data type WORD, for example, can be used to represent anything contained in a word and therefore an ARRAY OF WORD can generalize the usefulness of a procedure. For example, the procedure below is a generalized swap routine using generic data types:

```
FROM SYSTEM IMPORT WORD;
    ...
PROCEDURE GenericSwap(VAR x, y :
        ARRAY OF WORD;
        VAR ok : BOOLEAN);

(* Swaps the contents of two
   variables of the same size *)

VAR
  count : CARDINAL;
  temp  : WORD;

BEGIN
  IF HIGH(x) <> HIGH(y) THEN
    ok := FALSE;
  ELSE
    FOR count := 0 TO HIGH(x) DO
        temp := x[count];
        x[count] := y[count];
        y[count] := temp;
    END;      (* FOR *)
        ok := TRUE;
  END;   (* IF *)
END GenericSwap;
```

The ease and success of changing to Modula-2 can be greatly enhanced with a good text and compiler. If possible, the compiler choices and texts should be studied in conjunction. Trying the examples in the texts with all the compiler choices would aid in making a good decision. If information is needed to get started, it would be useful to become involved with the Modula-2 information news group (INFO-M2%UCF1VM.BITNET@cunyvm.cuny.edu) on USENET.

It is always a good idea to provide the student with language support tools, such as a good interactive, screen-oriented editor and a source level, interactive debugger. These will help to increase the student's productivity. The debugger, in addition, allows the student to follow his/her code step-by-step, thus enabling the student to visualize the activity provided by each language construct. With this knowledge the student can quickly learn how to program more effectively.

AND NEXT?

The 1988 fall quarter will begin the courses from the programming skills sequence which are expected to put Modula-2 to the "real" test. Both the Program Design and Implementation course and the Data Organization and Management course require the student to design and write large (typically more than 1500 lines) programs on their own. They will also be required to write one large program as part of a team. The value of various design techniques and separate compilation will be stressed. Be sure to come back next spring for an update or contact me sooner if you wish (csnet: ncs@rit, UUCPnet: rochester!ritcv!ncs, or bitnet: ncsics@ritvax).