Modula-2 Legacy Code: Problems and Solutions

Excelsior, LLC

April 28, 2000

Chapter 1

Introduction

Back in 1980s, shortly after Modula-2 was introduced, is was called ”the programming language of the future”. A significant number of companies had chosen Modula-2 as a major instrumental language. There were several DOS compilers available on the market, with JPI’s TopSpeed system dominating.

A few years later, OS/2 became ”the operating system of the future”, and many companies had staked on it. TopSpeed and Logitech provieded Modula-2 compilers for OS/2.

As it turned out, most of those companies have to regret their past decisions now. Windows customer base is much larger than that of OS/2 and is constantly growing, whereas OS/2 customer base is reducing because of the movement to Windows. The situation with Modula-2 is not much better. The ISO 10514 Modula-2 Standard [1], voted in 1996, appeared too late to seriously influence public attitude to the language. Most DOS and OS/2 compiler vendors have dropped their Modula-2 products long ago. TopSpeed sells its new 32-bit compiler for Windows only as part of the Clarion package, which is expensive and is only available for Windows.

So, is there a chance for your applications written in Modula-2 to catch up? There are two ways: (a) migration of the source code to a modern 32-bit ISO Modula-2 compiler; and (b) conversion of the source code to an industry standard language, such as C or C++. At the same time, apllications have to be ported from DOS or OS/2 to Windows which only increases the complexity of the problem.

This paper discusses technical problems that may arise in the process of migration or conversion, along with some possible solutions.

Chapter 2

Migration

So you think Modula-2 is a better language than C or C++, but the old 16-bit compiler ties you up. Apparently, moving to a new 32-bit ISO compiler, such as Excelsior's XDS compiler, is quite a job; this Chapter tries to give you some idea how to solve the problems you are likely to encounter.

2.1 Language

Old Modula-2 compilers, such as TS or Logitech, implement the language described in Wirth's "Programming in Modula-2" (PIM) - the book that had been a de facto standard for years. Modern compilers, including XDS, comply with the ISO 10514 Standard, which added a lot to PIM, but is not a superset of it. Also, probably due to the lack of an official standard, PIM compiler vendors introduced plenty of language extensions. Usually, these problems are relatively easy to solve (context search and replace can be sufficient in some cases), but, for instance, TopSpeed Modula-2 OO extensions have no analogy in other products.

So, if your new compiler has no PIM compatibility mode, you'll have to modify your sources by hand. Of course, in most cases the compiler will display an error message upon encountering a language construct that it does not recognize, but there are subtle differences that it probably does not detect. For instance, absence of the ELSE clause in a CASE statement is equivalent to presense of an empty ELSE clause in PIM. In contrast, according to ISO 10514, if the value of the case selector is not contained in any case label list and the ELSE clause is not present, an exception shall be raised. Therefore, the CASE statement in the following sample:

```
x := 2;
CASE x OF
|0: p0;
|1: p1;
END;
```

does nothing if compiled by a PIM compiler, and raises an exception if compiled by an ISO compiler.

XDS compiler is able to detect situations like the above and produce a warning message.

2.2 Sizes of base types

Another thing to consider is the sizes of the base types. INTEGER, CARDINAL, and BITSET are 16 bits wide in old 16-bit compilers, whereas in modern compilers these types are 32 bits wide. Unfortunately, this problem resembles the famous Year 2000 problem - no one can tell where in a million of source lines your program relies on base types being exactly 16 bits wide.

XDS provides a ”passive” solution — the compiler option M2BASE16 forces base types to 16-bit. Usage of fixed size types from the module SYSTEM, e.g. INT16, CARD32, etc., may also be considered. But in general, it is recommended to use 32-bit types anywhere except when there is a need to represent externally defined data structures, such as network packet layouts or database records.

2.3 Libraries

Wirth proposed just a few simple library modules in PIM, so each actual implementation included its own set of library modules. The International Standard introduced yet another one. Fortunately for TopSpeed users, Excelsior offers an add-on for XDS compilers called TopSpeed Compatibility Pak, that contains library modules compatible with TopSpeed (see table 2.1).

If your Modula-2 compiler is not TopSpeed and your application intensively uses library facilities, you may consider re-implementing some of the library modules (probably on top of the ISO library); otherwise you will have to replace library calls.

Module name	XDS-x86 (Windows)	XDS-x86 (OS/2)	XDS-x86 (Linux)	XDS-C (all platforms)
`BiosIO`	√	√	√	√
`FIO`	√	√	√-	√-
`FIOR`	√	√	√	√
`Graph`	√	√	√	-
`IO`	√	√	√	√
`Lib`	√	√	√	√
`MATHLIB`	√	√	√	√
`MsMouse`	√	√	√	-
`ShtHeap`	√	√	√	√
`Str`	√	√	√	√
`Window`	√	√	√	-

Table 2.1: Module/product availability matrix

2.4 API calls

All Modula-2 programming systems provide capabilities to access the underlying operating system API, but the ways they map C API headers to Modula-2 may significantly differ.

There is an additional problem on OS/2, where many 16-bit calls are now obsoleted by 32-bit ones and are not included in the latest issues of the IBM OS/2 Developer's Toolkit. XDS provides no support for 16-bit system calls, except VIO*, Mou*, and Kbd*. You may need to replace 16-bit calls with their 32-bit analogues or to write C wrappers for them.

2.5 Programming systems

Of course, each compiler has its own set of options. Project systems are incompatible as well. Special care should be taken about compiler options and pragmae which affect program semantics, such as TopSpeed's "`call (o_a_copy -> off)`".

Chapter 3

Conversion

So you do not want to stay with Modula-2 or simply cannot do it because of the customers, but neither can you afford rewriting hundreds of thousands lines in C or C++. In this case, consider using XDS Modula-2 to C converter, available from Excelsior on a custom basis.

Conversion to C/C++ involves some extra efforts in addition to those listed in the previous chapter, because of the differences between these languages and Modula-2. Major differences are discussed in this section.

3.1 Scopes

C and C++ have no local modules and nested procedures. It is still possible to correctly translate nested procedures, but the resulting code lacks readability — a converter has to introduce extra parameters for outer procedure's locals that are used in nested procedures. It is recommended to get rid of nested procedures before conversion.

```
PROCEDURE proc(a: INTEGER);
  VAR b,c,d: INTEGER;

  PROCEDURE loc1(a: INTEGER);
  BEGIN
    b:=a;
  END loc1;

  PROCEDURE loc2;
  BEGIN
    loc1(d+a);
```

```
static void loc1(int * b, int a)
{
  *b=a;
} /* END loc1 */

static void loc2(int * b,
                 int * a,
                 int * d)
{
  loc1(b, *d+*a);
} /* END loc2 */
```

```
  END loc2;

BEGIN
  c:=1;
  loc2;
END proc;
```

```
static void proc(int a)
{
  int c;
  int b;
  int d;
  c=1;
  loc2(&b, &a, &d);
} /* END proc */
```

Note: In this example the variable ”c” needs not to be passed as an additional parameter, as it is only used in ”proc” itself.

3.2 Data types mapping

Some Modula-2 data types have no direct equivalents in C/C++. In particular, there are problems with large set types and arrays indexed from a value other than zero. Sophisticated macros or C++ classes have to be used to preserve semantics of declarations and operations or these types. Macros are more error prone than classes, but usually give better performance. XDS Converter supports both options.

```
// Arrays indexed by an arbitrary range

#define ArrayOf(T,L,H) Array<T,(long)(L),(long)(H)>

template<class T, long L, long H>
class Array
{
   public:
      inline T& operator[]( const long index ) const
      {
         return (T&) ( storage[index-L] );
      };
   private:
      T storage[H-L+1];
};
```

3.3 Statements mapping

All Modula-2 statements seem to have an obvious mapping to C/C++. But a closer look shows that this is not true. In the example below, although the first translation

is correct, the second is not because of the possible side effects of P() and Q().

```
FOR i:= 1 TO 10 DO          for (i = 1; i<=10; i++) {
  S(i);                         S(i);
END;                        } /* end for */

FOR i:= P() TO Q() DO       for (i = P(); i<=Q(); i++) {
  S(i);                         S(i);
END;                        } /* end for */
```

As P() and Q() have to be evaluated only once, the correct translation of the second FOR statement might look like:

```
FOR i:= P() TO Q() DO       tmp = Q();
  S(i);                     i = P();
END;                        if (i<=tmp) for (;; i++) {
                                S(i);
                                if (i==tmp) break;
                            } /* end for */
```

Notice the introduction of a temporary variable.

You may want to rewrite some of your FOR statements before conversion to improve the resulting program.

3.4 Library Mapping

Straightforward translation of Modula-2 library calls typically produces code that a C programmer would never write, e.g.:

```
Storage.ALLOCATE(p,SIZE(A));   Storage_ALLOCATE(&p, sizeof(A));
```

whereas the desirable result is:

```
p = malloc(sizeof(A));
```

XDS Converter could translate Modula-2 library calls to ANSI C or POSIX calls, but due to diversity of libraries in pre-ISO Modula-2 programming systems that functionality is only available on a custom basis.

All XDS compilers and therefore XDS Converter support interfacing to C libraries, so the alternative approach is to replace Modula-2 library calls with C library calls right in the Modula-2 code prior to conversion.

Chapter 4

References

1. ISO/IEC 10514-1:1996 - Modula-2 (Base Language)

   The Modula-2 International Standard.

2. The Modula-2 FAQ
   `ftp://FTP.twu.ca/pub/modula2/m2faq.html`

3. Excelsior, LLC. Home Page
   `http://www.excelsior-usa.com`

   Excelsior offers a full range of Modula-2 legacy code solutions: native code compilers for Windows, OS/2, and Linux, ”via C” cross compilers, custom conversion tools, migration and conversion services, etc.

4. Native XDS-x86 User’s Guide
   `http://www.excelsior-usa.com/doc/xc.html`

   Appendix ”TopSpeed Compatibility Pak” covers TopSpeed to XDS porting issues, but some of the remarks are applicable to the more general PIM to ISO case.