CONVERTING EQUIPMENT CONTROL SOFTWARE FROM PASCAL TO C/C++

L. Hechler, GSI (Gessellshaft fuer Schwerionenforschung), Darmstadt, Germany

Abstract

The equipment control (EC) software of the GSI accelerators has been written entirely in Pascal. Modern software development is based on C++ or Java. To be prepared for the future, we decided to convert the EC software from Pascal to C in a first step. Considering the large amount of software, this is done automatically as far as possible. The paper describes our experiences gained using a Pascal to C translator, Perl scripts, and, of course, some manual intervention.

1 MOTIVATION

The EC software comprises the device representation layer, the real-time layer, and the device drivers [1]. Except for some assembler code, it has been written entirely in Pascal.

For embedded applications there are no integrated cross development systems that currently support Pascal. The system we use runs under VMS and its support expires completely by the end of 2001.

However until now we invested about 40 person-years in developing and maintaining the EC software. A lot of special know-how has gone especially into the real-time layer. The functionality gained in this work must be preserved.

Future control system developments have to be implemented with modern object-oriented methods. Appropriate up-to-date tools are based on C++ or Java nearly without exception.

Existing hardware (400 VME boards) must be used in the future as well since it cannot be replaced completely due to cost reasons. And, last but not least, the conversion must not affect the day by day accelerator operation.

2 CONVERSION

We decided to convert the EC software from Pascal to C in a first step. This allows us to "re-use" the software on one hand and to establish a basis for re-engineering the control system with modern methods and tools [2] on the other hand.

Considering that EC software consists of about 170 000 lines of code (LOC), comments not counted, it is clear that conversion has to be done automatically as far as possible.

The basis for a conversion is EC software for one device class. There are 61 different device classes, each one controlled by dedicated software. To ease the conversion,

we issued a cookbook [3] that describes the process step by step.

To convert the Pascal code into C automatically we use the Pascal to C translator p2c.[1] Perl scripts are then used to adapt the notation of identifiers to our style guide.

In spite of the automation there is a lot of manual intervention required. Besides the preparations for p2c and Perl there are four essential reasons that make manual interaction necessary.

2.1 Compatibility of Data Structures

The p2c manual pages state that "most reasonable Pascal programs are converted into fully functional C which will compile and run with no further modifications". This may be true for stand-alone programs. Given the EC software, it has to be taken into account that in case of communications with other modules, e. g. programs of the operating layer, the structure of interchanged data has to be kept fully compatible because those modules have not been changed.

1. Pascal supports PACKED records and arrays to facilitate having minimal alignment space between elements. C does not support this feature.

2. At GSI we use the Organon Pascal compiler from CAD-UL which supports the dialect of the Oregon Pascal/2 compiler. Their syntax only differs in one key word, but they generate completely different codes. However p2c makes some assumptions about the generated code, e. g. the order of bits in a bitset, which is crucial, for instance, when hardware registers are accessed.

3. In Pascal the allocation size of an enumeration type depends on the number of its elements. It may be one or two bytes. In C the allocation size is always an int.

4. The Pascal string ARRAY [1..len] OF CHAR contains len characters. Its allocation size in memory is len bytes. A C string with equal size is char s[len]. It can hold only len - 1 (printable) characters because of the terminating \0.

2.2 Linking of Pascal and C modules

A CPU of the device representation layer hosts EC software of up to 12 different device classes. On this layer it must be possible to combine modules written both in Pascal and C because EC software for a number of device classes can not be converted at the same time.

---

[1] p2c is part of many Linux distributions. It runs under VMS as well.

Combining Pascal and C modules means that they have to be linked together. In this case identical procedure calling mechanisms have to be ensured.

1. P2c translates routine parameters into a structure that contains a pure C function pointer and a "static link", a pointer to the parent procedure's local variables. This structure is passed to the called function. Both of our compilers, the Pascal as well as the C compiler, need plain C function pointers. The option to force p2c to use this concept is available but does not work.

2. Pascal can handle conformant array routine parameters defined as

```
f(a: ARRAY [lo..hi: INTEGER] OF MyType);
```

by syntactically passing the array as actual parameter only:

```
VAR x: ARRAY [7..13] OF MyType;
f(x);
```

On calling the routine, the array, or its address in case of a `VAR` parameter, as well as the lower and upper limit of the array are pushed onto the stack. Thus the array bounds may be checked by the called routine.

P2c generates C code where the routine is declared and called with three parameters explicitly. The order the parameters are pushed onto the stack differs from that of the Pascal compiler.

2.3 Maintainability

The converted software is not a final product. It has to be maintained for changed or extended future requirements. Therefore readable and comprehensible code is indispensable. To achieve this, sufficient work has to be invested into simplifying and refurbishing the plain C code produced by p2c and the Perl scripts.[2]

1. Pascal supports nesting of routines. The parent routine's local variables lie in the scope of the nested routine. C does not provide this concept. So p2c combines the parent routine's local variables to a single structure and adds an additional link parameter to the subroutine's parameter list that points to this structure, thus allowing the subroutine to access its parents' variables. C code designed like this looks somewhat odd.

2. Pascal provides the `WITH` statement to abbreviate the notation for references to fields of structured variables.

```
WITH struc.field DO subfield := 1;
```

P2c creates a pointer for every `WITH` statement with generated names `WITH`, `WITH1`, `WITH2`, etc. to access the field of a structure.

```
T_field *WITH = &struc.field;
WITH->subfield = 1;
```

---

[2]Due to restricted space only some items are mentioned here.

Often there is no explicit type for the field the `WITH` statement references. In those cases p2c needs to declare an additional pointer type first (`typedef struct T_field ...`) before it can define the pointer itself.

These constructs are hardly found in common C programs.

3. Pascal allows the definition of an array of structures within one statement. A variable definition looks like this:

```
VAR x: ARRAY [1..7] OF
         RECORD i: INTEGER; c: CHAR END;
```

Although C supports a corresponding construct, p2c declares a structured type before it defines the array.

```
typedef struct _REC_x {int i; char c} _REC_x;
_REC_x x[7];
```

To do so p2c must generate a name for the structured type, which is _REC_x where x is the name of the array.

2.4 P2C Errors

We encountered only two substantial p2c errors not mentioned in the p2c manual. Both of them are very difficult to detect since the compiler does not report an error. Overlooking them during the manual intervention means they occur during the runtime of the software where they are moreover hard to debug.

1. In some cases p2c translates a Pascal 32 bit wide unsigned integer type

```
TYPE uns_long = 0..16#FFFFFFFF;
     myType = uns_long;
```

into a single C character type.

```
typedef char myType;
```

The error occurs only infrequently. Unfortunately we were not able to reproduce the circumstances of its occurrence.

2. The Pascal pointer `ptr` should point to a 16 bit wide type, e. g. a hardware register, that has an offset of 4 bytes to a base address `addr`.

```
TYPE uw_p = ^uns_word;
VAR  ptr: uw_p;
     addr: uns_long;
ptr := loophole(uw_p, addr + 4);
```

In rare cases p2c translates the pointer assignment to

```
ptr = (uns_word*)((uns_long*)addr + 4);
```

which results in a miscalculated pointer value. The expression `(uns_long*)addr` type-casts `addr` to a pointer to a 32 bit type and thus adding $4 \times 4 = 16$ bytes to the base address instead of 4.

3   APPLYING THE STYLE GUIDE

Unlike C, Pascal identifiers are case insensitive. P2c takes the first occurrence of an identifier to determine the notation of all subsequent occurrences. Mostly these notations do not conform to our style guide. To force the notation of identifiers according to the style guide, we developed some Perl scripts that do most of the job.

A Perl script recognizes expressions for instance like

```
#define The_Answer 42
typedef struct my_type {...} my_type;
```

and recasts the identifiers accordingly (getting THE_ANSWER and MyType).

To handle more complex constructs, a parser-like script would be required. This is not implemented yet. Thus manual modifications are necessary whereby each identifier has to be adjusted only once.

All changes of identifiers in the software of one device class are then stored as key value pairs in a device class specific local data base (DB). The pairs describe the translation from the old into the new style guide conform notation. The creation and completion of the local DB is done by another Perl script.

A third script is used to apply the translations stored in the local DB to all identifiers in all files of a device class. Additionally a global DB is used which applies the translation of the identifiers of the system interface.

4   STATUS

Currently EC software for 15 different device classes has been converted. Devices have been operated with the converted software for more than 6 months. Some of them are even in therapy operation [4]. Apart from teething problems in the beginning of the conversion process the software has showed good quality and bug fixing is an amazingly rare necessity.

4.1   Time

To estimate the manual interaction effort to convert the software for one device class the process can be split into 4 phases. The outcome is the following distribution:

1. p2c including some preparations   10%
2. manual intervention, part I   20%
3. Perl including building of local DB   20%
4. manual intervention, tests, bug fixing   50%

Phase 2 is necessary since some manual interventions are better done before using the Perl scripts. Although 1 and 3 are the "automatic" phases they also need manual actions, particularly phase 3. With more experience the percentage of phase 4 increases but the overall conversion time decreases.

On an average, EC software for one device class consists of 2200 LOC. Its conversion requires us about 2 person-weeks. To convert the whole EC software consisting of 170 000 LOC we will need approximately 39 person-months or 3.25 person-years.

Without the help of p2c and Perl scripts we roughly estimate twice to four times the effort. There was only one attempt to convert a device class completely manually.

Balzert [5] states that software *development* results in 350 LOC per person-month.

Given this, our method is 2 to 4 times faster than a pure manual conversion and more than 10 times faster than a redevelopment.

5   CONCLUSION

Using p2c and Perl scripts converting EC software from Pascal to C is feasible without major problems. In spite of the automation tools there is a lot of manual intervention required before C software for a device class is ready to be released.

Our method allows us to convert EC software in reasonable time. Entirely re-engineering the EC software would have exceeded our manpower capacity excessively.

With EC software converted to C we are well-prepared to take the next step to C++ (or Java). It should be possible, at least on the device representation layer, to re-use the C functions, which are usually straightforward, as methods of classes in C++. The use of C++ on the real-time layer has to be investigated, particularly with regard to the highly demanding 50 Hz linear accelerator operation.

6   ACKNOWLEDGEMENTS

7   REFERENCES

[1] U. Krause, V. Schaa, R. Steiner, "The GSI Control System", Proceedings of ICALEPCS '91, Tsukuba, Japan, 1991.

[2] U. Krause, "Re-Engineering of the GSI Control System", these proceedings.

[3] L. Hechler, P. Kainberger, G. Schwarz, "P2C - Umstellung der Gerätesoftware von Pascal nach C/C++", Accelerator Controls Documentation U-GSW-08, GSI, Darmstadt, November 2000, http://bel.gsi.de/mk/sty/p2c.html.

[4] U. Krause, R. Steiner, "Adaption of a Synchrotron Control System for Heavy Ion Tumor Therapy", Proceedings of ICALEPCS '95, Chicago IL, USA, 1995.

[5] Helmut Balzert, "Lehrbuch der Software-Technik: Software-Entwicklung", Spektrum Akad. Verlag GmbH, Heidelberg, Berlin, Oxford, 1996.

[6] Dave Gillespie, "p2c - Pascal to C Translator Manual Pages", Caltech.

[7] Udo Krause, "C/C++ Style Guide", Accelerator Controls Documentation O-SIS-10, GSI, Darmstadt, December 2000, http://bel.gsi.de/mk/sty/cstyle.html.