92-96

E10-92-96

S.A.Zaporozhets*, E.V.Chernykh*

THE USE OF PROLOG FOR REAL-TIME SOFTWARE DESIGN

Submitted to "II International Workshop on Software Engineering, Artificial Intelligence and Expert Systems for High Energy and Nuclear Physics", January 13-18, 1992, La Londe, France

*E-mail:shimanskiy@lhe03.jinr.dubna.su

1992

Запорожец С.А., Черных Е.В. E10-92-96

Применение языка ПРОЛОГ

для создания программ реального времени

Обсуждается опыт применения языка ПРОЛОГ для разработки компилятора языка высокого уровня. Компилятор предназначен для программирования модулей на основе трансльютеров в системе сбора данных. Одним из достоинств языка ПРОЛОГ является удобство синтеза и преобразования данных, имеющих структуру дерева. К этой категории относятся описания языков программирования по форме Бэкуса—Наура и тексты программ на языках высокого уровня. Указанное свойство языка ПРОЛОГ использовано авторами для разработки программы-разборщика, являющейся частью компилятора с подмножества языка Паскаль с расширением для задач реального времени. Программа-разборщик генерировалась программой конверсии используемого языка программирования непосредственно по форме Бэкуса—Наура. Генератор кода транспьютера, загрузчик и интерфейс на основе меню разрабатывались традиционным способом. Рассмотрен пример применения компилятора в системе реального времени.

Работа выполнена в Лаборатории высоких энергий ОИЯИ.

Препринт Объединенного института ядерных исследований. Дубна 1992

Zaporozhets S.A., Chernykh E.V. E10-92-96

The Use of Prolog for Real-Time Software Design

Experience of Prolog application in designing of a high-level language compiler is discussed. The compiler is intended to program a transputer module included in a data acquisition system. One of the Prolog language advantages is the convenience of synthesis and conversion of tree-like data structures. Such structures as BNF-definitions of programming languages and texts of high level language programs belong to this category. We used this Prolog feature for the synthesis of a parsing program to design a Pascal-subset language compiler with extension for real-time applicaiton. The parser was generated by a BNF-converter program directly from the BNF-specifications of the language. A transputer code generator, a loader and a menu-interface were designed in the conventional way. An example of the compiler application in a real-time system is described.

The investigation has been performed at the Laboratory of High Energies, JINR.

Preprint of the Joint Institute for Nuclear Research. Dubna 1992

1. Introduction

A program code of real-time system software has to be of high efficiency in spite of the fact that this software is difficult to debug and there is a lack of clearance of program source code because of its dependence on hardware configuration.

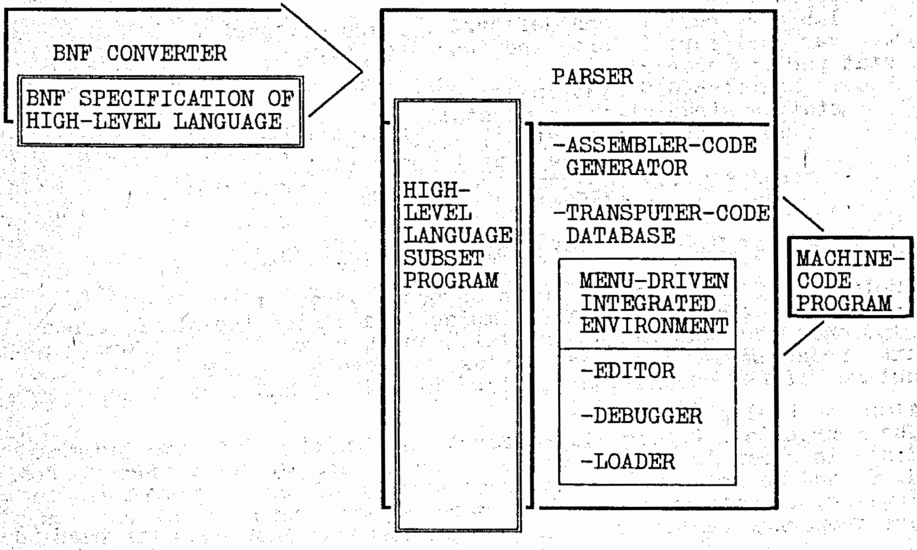
BNF CONVERTER
BNF SPECIFICATION OF HIGH-LEVEL LANGUAGE
PARSER
HIGH-LEVEL LANGUAGE SUBSET PROGRAM
-ASSEMBLER-CODE GENERATOR
-TRANSPUTER-CODE DATABASE
MENU-DRIVEN INTEGRATED ENVIRONMENT
-EDITOR
-DEBUGGER
-LOADER
MACHINE-CODE PROGRAM

Fig.1. The compiler structure.

Conventional programming languages have no tools for effective data acquisition system programming. An assembler is usually used to describe the software interface to reach an efficiency and the assembler routines are mixed with some conventional language (Basic,Fortran,Pascal). But this way does not lead to a readable source code and a program code efficiency .

To solve these problems we have designed the problem-oriented compiler for the Pascal-subset language with extensions taking into account characteristic properties of system hardware (Fig.1).

We discuss the compiler itself and describe an example of compiler application for a real-time data acquisition system. System hardware comprises a transputer module as a base element of the system, a parallel CAMAC branch and a personal computer connected to the transputer via a transputer link.

2. Scheme of compiler designing

To describe the syntax of the programming language, we used the following meta-symbols:

```
        ::=      means 'is',
        XYZ     (uppercase)  means 'reserved word XYZ',
        abc     (lowercase)  means 'syntactical variable abc',
        |                    means 'or',
        {·}                  means 'list of elements'.
```

To reach more readability of specifications we used the uppercase for terminal symbols instead of usual brackets. Fig.2 shows a part of Pascal language syntax definition.

```
program::=program_heading block.
  program_heading::=
    PROGRAM program_identifier file_identifier_list
  block::=decl_part stat_part
    decl_part::={decl_sect}
      decl_sect::=
        lab_decl_part | cons_def_part |  type_def_part |
        var_decl_part |  proc_and_func_decl_part
    stat_part::= comp_stat
      comp_stat::=BEGIN stat{;stat} END
        stat::= struc_stat | simp_stat
          simp_stat::=  goto_stat | assig_stat | proc_stat |
                        inline_stat | empty_stat
          struc_stat::= rep_stat | comp_stat  | cond_stat |
                        with_stat
```

Fig.2 Part of Pascal language syntax definition.

The notation of this type is known as Bacus normal form (BNF)/1/. One needs approximately 80 BNF-formulae to describe the ALGOL syntax and 150 ones for the PASCAL syntax. Another well-known form to define programming language syntax is syntax graphs, but the advantage of BNF-definition consists in that it has a structure very similar to Prolog/2/ clauses.

The appropriate Prolog clause for the definition of the program heading is shown in Fig.3, where P1 and P2 are the parameters to be find, Si and So are input and rest strings of the program text to be parsing, fronttoken is reserved word recognizing predicate, S2 and S3 are temporary variables.

```
is_program_heading(program_heading1(P1,P2),Si,So):-
           fronttoken(Si,"PROGRAM",S2)
          ,is_program_identifier(P1,S2,S3)
          ,is_file_identifier_list(P2,S3,So).
```

Fig.3 Prolog clause for definition of program heading.

The BNF-converter developed transforms a syntax specification file of the BNF-definition to the form similar Fig.3. Each alternative in BNF gives one Prolog clause. (For parser-developing purpose in

some realisations of Prolog there are implemented a so-called Definite Clause Grammar mechanisms/2/.)

In this way we convert 150 BNF-formulae into the parsing program and use it to build a compiler of a Pascal-like language. Other parts of the compiler are assembler code generator, a transputer code database, and a menu-driven integrated environment including a text-editor, debugging tools and a loader(Fig.1).

Fig.4 shows a simple program and a tree-like structure generated from it. The syntax tree of the program is input data for the assembler code generator.

```
PROGRAM xxx(a,b,c);
VAR x,y:integer;
BEGIN
 x:=3+1; y:=9
END.

program1(program_heading1(program_identifier1("x
xx"),file_identifier_list2(file_identifier1("a")
,[file_identifier1("b"),file_identifier1("c")]))
,block1(decl_part1([decl_sect4(var_decl_part1(va
r_decl1(iden_list1("x",["y"]),type1(simple_type3
(type_iden1("integer")))),[]))]),stat_part1(comp
_stat1(stat2(simp_stat2(assig_stat1(var1(entire_
var1(var_iden1("x"))),expr1(simple_expr1(term1(c
ompl_factor1(signed_factor1(factor1(unsig_cons1(
unsig_numb2(unsig_int1(digit_sequence1('3',[])))
)))),[]),[adding_operterm(adding_oper1,term1(com
pl_factor1(signed_factor1(factor1(unsig_cons1(un
sig_numb2(unsig_int1(digit_sequence1('1',[])))))
)),[]))]),[])))),[stat2(simp_stat2(assig_stat1(v
ar1(entire_var1(var_iden1("y"))),expr1(simple_ex
pr1(term1(compl_factor1(signed_factor1(factor1(u
nsig_cons1(unsig_numb2(unsig_int1(digit_sequence
1('9',[])))))))),[]),[]),[])))))])))))
```

Fig.4. Simple program and its internal representation on the compiler after parsing.

3. Hardware dependent language extension

Now it is possible to extend the description of a language with dedicated features of a data acquisition system and a transputer to reach a more efficiency of on-line programs and their readability as well. A transputer is an extremely suitable chip for real-time application because of submicrosecond interrupt latency, real-time kernel and high-speed communication facilities/3/. The system of the transputer machine code /4/ includes special instructions to create, run, suspend or stop tasks and supports time-sharing between the tasks. These features make it possible to distribute system processes in accordance with their latency and to provide a fast response to external events. Fig.5 shows task prioritization and scheduling in the transputer.

Tasks in two lists differ in priorities: high and low ones. There is no limitation on the number of tasks in a list. Processor time is shared between the tasks in the low priority list by the time quantization with a maximum quanta value equal to 1 millisecond. We define the task in a similar way as a procedure with following formula:

```
        program::=  PROGRAM
                     declaration_section
                     TASK task { ;TASK task  }
                     statement section .
        task ::= block
```

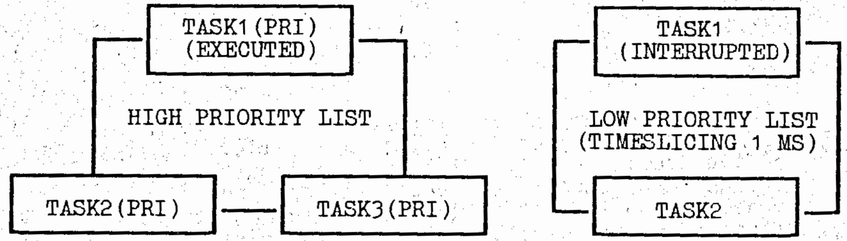

Fig.5. Hardware implemented multitasking in a transputer

Calling a task, we activate the scheduler and add the task vector in the appropriate list of tasks waiting to get control.

The described method gives us an opportunity to include in the description of the language such features of transputer architecture as a special type of memory that represents in this case cells of CAMAC devices in the data acquisition system.

To read out data we define the following variable:

```
            VAR TDC :INTEGER CAMAC C=1 N=1 A=0 F=0;
                ADC :INTEGER CAMAC C=1 N=2 A=0 F=0;
```

and place the code in the procedural part of the program:

```
            EVENT[1]:=TDC;
            EVENT[2]:=ADC;
```

We find this way of programming more readable than conventional:

```
            CAMAC(C,N,A,F,D24,D16,Q)
            CAMAC(C,N,A,F,D24,D16,Q)
```

Communication facilities are presented as a special type of device or a file,so the calling

```
                   READ( LINK0,AR0 );
                   WRITE( LINK1,AR1 );
```

means that the array of data AR0 will be read via transputer link 0 and the array AR1 will be written via transputer link 1.

4. Example of application

We discuss the program development for a simple data acquisition system using the described tools. The system/5/ consists of a transputer module, IBM PC, a CAMAC parallel branch and intended to control a spectrometer during a nuclear physics experiment. The system must accept data on primary and secondary beam parameters from different other data acquisition computers and data from a polarized target control subsystem. During the experiment PC accepts data from transducers through CAMAC and writes them into PC RAM. Then the transputer and PC perform preliminary data processing; PC transmits the information to a TV monitor and writes it on a magnetic tape unit. A number of tasks must be executed in the system, we list only some of them:

- to accept data from an accelerator control system and from a beam transport control system
- to read data from CAMAC modules of the experimental setup
- to build an event-record and send it to the host computer
- to display periodically the data for an operator.

We describe each task as a task in transputer communicating via links with external systems and via a shared memory with others. The task distribution for the system under consideration is shown in Fig.6.

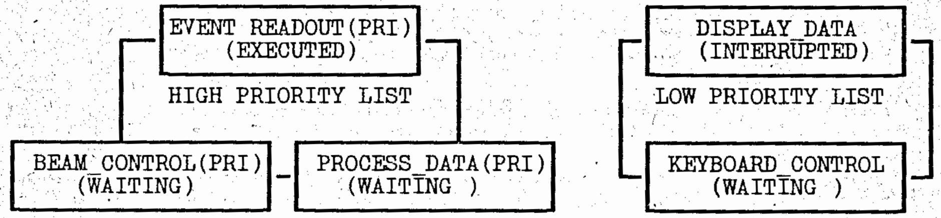

Fig.6. Data acquisition system under multitask control.

The corresponding program structure implementing this task distribution is shown in fig.7.

5. Conclusion

We used Prolog to design system software for a data acquisition system containing a transputer module, PC and a CAMAC parallel branch. We used the hardware operation system kernel in a transputer to provide effective process execution in the real-time data acquisition system of the spectrometer.

- The application of PROLOG gives an opportunity to add purpose-designed operators to standard high-level languages defined in the BNF form.

- The described approach can afford us a possibility in particular to program real-time systems in terms of a high level conventional language with no losses in system efficiency.

```
{--- Simple data acquisition program  -}
        program DAQ;
{--- Readout the CAMAC devices  ---}
            task Event ;
                  var TDC integer CAMAC C=1 N=1 A=0 F=0 ;
                  var EVENT array [1..10] of integer;
                  begin
                    EVENT[1]:=TDC;
                    ...
                  end;
{-- Communication with a remote beam control system --}
            task Beam_Control;
                  var BeamData :integer ABSOLUTE $1000;
                  begin
                    read( LINK1,BeamData)
                  end;
            task Process_Data;
                  begin   ...  end;
            task Display_Data;
                  begin   ...  end;
            task Kearbord_Control;
                  begin   ...  end;
{-- Start -- }
         begin
            Event(pri); Beam_Control(pri); Process_Data(pri)
            Display_Data; Kearbord_Control
         end.
```

Fig.7. Program skeleton for the data acquisition system.

References

1 J.J.Donovan, System Programming, McGraw-Hill,1972.
2 J.Malpas. Prolog:A Relational Language and Its Applications. Prentice-Hall Inc.,1987.
3 T.Woeniger. An Introduction to Transputers. DESY 90-024,March 1990.
4 J.D.Nicoud,A.M.Turell. The Transputer T414 Instruction Set. IEEE MICRO,june 1989.
5 E.V.Chernykh and S.A.Zaporozhets.JINR Preprint, P10-90-216 (in russian).

Received by Publishing Department
on March 5, 1992.