Programming a Logical Control Method by a Parallel Process

P. Jiroušek

This paper deals with the development of the problem oriented language PRIMAS for use in program control. It is based on virtual parallelism of the controlling program to make its hierarchical structure transparent. The author has also worked on the compiler and the control process simulator. This system enables verification of the control algorithm when there is no controlled machine and no control system. The PRIMAS language, compiler and simulator were developed and applied to real tasks, in the course of work on the author's PhD dissertation.

Keywords: Higher program language, logic control, modelling and simulating.

1 Introduction

In developing any kind of industrial control system, it is important what kind of software support is used, in addition to the hardware. In addition to the standard methods, solved by the compiler of linear schemes and various representations of combining equations, there is a wide range of tasks where the support of higher program language is necessary. There are languages like C, Pascal and perhaps Basic. The problem is their in ability to represent combinatorial relations. A sufficient solution is to use a parallelism for programming logical control in a higher language. PRIMAS offers not only the usual language structure but also parallel processes, timing functions, easier object access and simulation instruments.

2 A linear scheme or a programming language?

As an example, let us take a situation where our equipment is performing one operation from its list. The control program is composed of operations from this list. Running these operations depends on a combination of logical conditions. When a condition is fulfilled the running operation should stop and a different one will start. There are usually some other parallel operations running on their own algorithm. In addition the equipment has an alphanumerical display with a keyboard to communicate with the operator. It can also communicate with its major system.

If we want to use the linear scheme compiler or some other form of combinatorial equations compiler, difficulties may occur. We will have trouble to form sequences and cycles and there will be probably a lack of procedures. The dialog to the operator and the internal communications, taking place concurrently with the running processes can be performed only on the basis of commercially-available producers modules. Transparency and program service will also be difficult. There exist higher level of logical control program systems such as STEP 5 and GRAFCET. However there are situations so that program instruments cannot deal with.

In C language the programmer writes functions like Shift Left(), Drill(), etc., and puts them into a sequence. This sequence is closed into the cycle for $(i = 0; i < 5: i{+}{+})$ and creates a readable algorithm for Action(). Problems emerge if we want to bring this action to the end abruptly whenever a condition X appears. Into Shift Left(), Drill() and into other actions we have to write a new ifs (…). Unstructured jumps need to be solved and program transparency suffers. Another problem is parallel dialog to the operator and the communications. Here the interrupt function can be helpful. However the program can become too complicated and totally lacking in transparency.

3 Parallel tasks

The task of running the system control, dialog and communication, which are quite independent actions, at the same time can be solved naturally by the ability to create parallel processes. Parallel processes are also good for creating the actual control process and, even as a tuning tool.

4 Decomposing the task

In usual multitasking, the tasks are not allowed to influence each other and they communicate through exporting and importing files. In the control processes there is typically close cooperation among the actions. If we substitute actions for persons, the whole program becomes more like a company. They have the same interests, work with the same information and know what their colleagues do. This naive comparison proves to be surprisingly adequate when asking: "How many actions, what will they perform, and how do they influence each other?" We can create a "manager", a "communicator", a "service man", a "security technician", and a mass of "workers".

The relations between actions are solved by the main program structure and as are some particular problems. Let us take our example with action conditioned by condition X. Instead of testing this condition in each procedure, we just launch the guarding parallel process. This will continually test that condition. Our action will no longer be needed. When the condition is fulfilled the guard stops the action and launch a different one, or even the same one with changed parameters. This condition can be, for example, a time limit. Our action will set the time limit and the guard watches its runover. This type of the parallel control has the advantage that the actions "see into each others plate".

In order to optimized decomposition, sensitivity and experience are needed.

4 Modeling and simulating

Parallel processes are an excellent tool for simulating and tuning control tasks. A control process is an interaction between the control and the controlled. In simple terms a model consists of controlling processes and a controlled utility. Then we leave them to interact each other. With suitable visualization of internal processes it is possible to observe the behavior of the whole system. The use of cross development software enables the tuning when there is no controlling process and no controlled utility.

In practical terms this means that the more time is used on development, the less time is needed when the program is brought to life. The program just works as soon as it is switched on.

5 Problems when implementing parallelisms

Parallelisms offer many advantages. It remains is to find suitable development software to enable the implementation of parallelism. Let us focus on control systems which do not support UNIX, QNX or some other multitask operating system. These are mostly smaller and cheaper systems, mainly eight bits, with processors such as 8051, Z80, I80186 etc. Most of producers of development software, if they do offer parallelism, will enable it to run only thanks to the real time core of their system. This processes drives the switching and also the mutual communication. This separates the language from the parallelism. There are libraries for approaching a core function. The compiler remains unaffected by parallelisms. Sharing the information in the system in this way is not problem-free. Passing the information through the core by sending messages proves to be a very safe and clear method, but it is not really necessary in this case. If such a message is an integer, sending it becomes to difficult. This variable can be shared between processes inside the memory. This can be done only if collisions between the producer and the receiver of the information are eliminated. The recipient cannot touch this variable at the same time as the producer is changing it. The core control has some procedure to deal with this.

One way to eliminate mutual work with one variable is to apply cooperative multitasking in a suitable manner. The processor is not forced to surrender the process, but it can decide whether or not it is appropriate. The information producer holds on the processor until the data work is finished. Then there is the problem of how to control process switching. If the application program has to take care of this problem, it is not only a burden to bear but also a possible source of any kind of mistake. These processes can cause long delays.

6 Parallelisms in PRIMAS

To deal with the problem discussed above the PRIMAS program system was developed. Its core consists of a higher language orientated to logical control tasks. It is based on the ideas of parallel programming. When looking at the source code, a slight resemblance with Pascal is recognizable. There are procedures, functions, integers, real variables, the classical if-then-else program structure, etc. In addition, the bite objects have been made accessible, the program works with virtual timers, and there are also language structures which facilitate work with these particles.

In PRIMAS, parallelism is supported by the ability to define more than one process. Switching between them is ensured by the cooperative multitasking method. The decision on switching, and its exact placing, is made by the compiler according to a precisely set strategy. The programmer can also influence this placing by himself, but this has been used minimally. Individual processes share the global variables, timers, inputs and outputs, and they can call the same procedures if they are re-enter. Processes can launch and stop each other, while they are accessible to each other through their individual identifiers. The processes can be addressed indirectly, so they become parameters in the procedures and functions.

6 Tuning and simulating in PRIMAS

The PRIMAS program system consists of a compiler for the given control system and the universal tuning program. It is made for MS-DOS. When simulating, one or more external file is attached to define the object placement and the way in which they are shown on the PC screen. The tuning program then interprets the translated processes, and projects all changes onto the screen. Here we can observe how the water level is rising, how the object runs along its trajectory, or how the end switch is working. Then this process can be influenced with the use of the mouse or keyboard. When the mouse is clicked on an object showing a binary value, an immediate change will follow. In other cases, the dialog line shows up and a new value will be inputted from the keyboard.

7 Implementing PRIMAS

While it was being developed PRIMAS was implemented for I8048, I8051, Z80, Z181 and I8086 processors. The processor units of PROMOS system by ELSACO and AREM pro are based on these platforms. Many library functions have been prepared for PROMOS, to support specific hardware. PRIMAS is offered as a device for developing programs for this hardware.

Example

```
%-------------------------------------------------------------------%
%  The following program has no practical means. It only          %
%  demonstrates commonly used elements of PRIMAS.                 %
%-------------------------------------------------------------------%
PROCESSES:     MAIN MainControl
                    ErrorStatus Oiling  <* Simulator *> ;
```

```
CONSTANTS:     HEX[2C]      NAME  PortAdr
                        5   NAME  CycleCount
               REAL 4.2E6   NAME  Coefficient
               ENUM Status0 Status1 Status2;
BYTE INPUTS:   WITH ADDRESS PortAdr NAME  InPort
               WITH ADDRESS      32 NAME  SerialLine
               WITH ADDRESS      33 NAME  DataSL
               WITH FILTER  HEX[2D] NAME  FPort ;
BYTE OUTPUTS:  WITH ADDRESS 1024   NAME  OutPort1 ;
TIMERS:        INTERVAL 100
               Time DelayTimer AlarmClock ;

INCLUDE "\INC\library.prm"

INT:           Counter
               Array[22]     Array[2]  NAME  Element2
               WITH ADDRESS HEX[F000]  NAME  VideoRam
               WITH ADDRESS 100        NAME  MonitorStatus ;
REAL:          ra rp[20] ;
BIT OUTPUTS:   OutPort1.0       NAME  Pump
               OutPort1.7       NAME  ErrorSignal
               MonitorStatus.5  NAME  BufferFull
               OutPort1.1#      NAME  Drain ;
BIT INPUTS:    InPort.2         NAME  Button
               InPort.3         NAME  LoadFull
               InPort.4#        NAME  OilPressureOk
               SerialLine.0     NAME  DataReady
               @(123).5         NAME  Bit5OnAdr123
               Array[3].10      NAME  ImprotantBit ;

INT FUNCTION: CharFromSerialLine(I:Mask)
    INT:   i p[10];
    REAL:  r x[20];
    WAITFOR DataReady
    = AND(DataSL,Mask) ;

PROCEDURE: GetMessage(I:Length,L:Signal)
    INT:   i ;
    i := 0
    REPEAT
       Array[i] := CharFromSerialLine( HEX[7F] )
       i := i+1
    UNTIL [i>Length]
    BufferFull := Signal  ;
LOG FUNCTION:  DumpEnable = Button * LoadFull ;

PROCESS                Oiling
    INT:   i ;
    REAL:  a ;
    BEGIN
       WAITFOR OilPressureOk#
       TIMER := 50   SET( Pump )
       WAITFOR OilPressureOk+[TIMER=0]
```



```
        IF   OilPressureOk#
        THEN SET( ErrorSignal )
        ELSE SET( ErrorSignal# Pump# )
        ENDIF
    AGAIN
;
PROCESS                ErrorStatus
    BEGIN
        IF      ErrorSignal
                THEN  VideoRam := "Oiling Error "  RESET
        ELSEIF  Bit5OnAdr123
                THEN  VideoRam := "PError XY    "  RESET
        ENDIF
    AGAIN
;
PROCESS                MainControl
    OutPort1 := 0
    WAITFOR Button   WAITFOR Button#
    RUN  Oiling        RUN  ErrorStatus
    BEGIN
        VideoRam := "Working Cycle "
        GetMessage(4,0)
        CASE Element2
           OF   'A'  THEN  SET( OutPort1.1 )
           OF   'B'  THEN  SET( OutPort1.2 )
           ELSE            SET( OutPort1.3 )
        ENDCASE
        DO Array[1] TIMES
            OutPort1.4 := OutPort1.4#
            DELAY  TIMER := 2
        LOOP
        WHILE DumpEnable#
            SET( OutPort1.5  OutPort1.6# )
            DELAY  TIMER := 5
            SET( OutPort1.5# OutPort1.6  )
        ENDWHILE
    AGAIN
;
<* INCLUDE "\SIM\simul.prm" *>
```

9 Conclusion

On the basis of analyzing the process of creating a control program, the PRIMAS program system offers two means of development. First, the language for formalizing the control tasks of a given class. Second, the simulation tool.

The PRIMAS program language as a language for describing the algorithm for logical automatic machines offers an innovative and untraditional means. Logical controlling has until now been commanded by methods deriving from finite automatic description. An example is proved by be the GRAFCET system, created in 1997 by the French Association for Economic and Technical Cybernetics. GRAFCET became an international standard for a sequence description. There are still research teams working on the GRAFCET system and implementing this method on to control systems. Although this system is a significant step forward, compared to state diagrams, it still suffers from all the shortcomings discussed above.

PRIMAS was introduced to specialists on GRAFCET and similar systems for modeling sequence systems from the Polytechnique National de Grenoble during the Higher Education in Control Engineering seminar organized by the Faculty of Electrical Engineering at CTU. They declared that no other system similar to PRIMAS was known to them. They thought the reason might lie in "cultural" differences in

logical control and computer science. They also emphasize the importance of solving the problems caused by expressing utility diversity.

PRIMAS offers some new elements and language constructions, that make the algorithms much easier to express. The basic characteristic is the parallel description of the algorithm by mutually integrating processes. In PRIMAS, parallelism is not just a way of exploiting the processor better, but especially it leads to easier control system decomposition. Parallelism is also used for describing a standard situation. Writing it in a single sequence program would be theoretically possible but, practically very difficult and not transparent. Other language properties supporting the development of the control program, are the timing functions, the simple approach to the system interface, and independent object access.

The PRIMAS program system as a means of development can verify whether the system works within the assigned time. Unlike other program systems, for instance systems supporting the GRAFCET, verification is performed in graphical form. This ability is given by the means of simulation in the tuning program and is also contained in the language. This enables verification of the ideas and conceptions without a physical need for the existence of the controlled system.

PRIMAS has been put into practice in several industrial companies and the users have been highly satisfied. They appreciate that it changes and improves the control system programmer and the mechanical part constructor relationship. Simulation has become their meeting point, and it makes their communication easier. In industrial companies where the system has been in use for a period of time, we can see that the wide ability of PRIMAS leads to higher demands on the system author. Some universities working on simulations of discreet processes have also expressed an interest in the PRIMAS system.

References

[1] Jiroušek P.: Vyšší programovací jazyk pro logické řízení paralelními procesy. Kandidátská disertační práce (Ph.D. thesis, Czech Technical University in Prague), Praha 1991.

[2] Jiroušek P.: "Programování úloh logického řízení paralelními procesy." *Automa* 3–4, 1996, (in Czech).

[3] Alla H., David R.: *Du GRASFCET aux Réseaux de Petri*. Hermes, 1989.

Ing. Pavel Jiroušek, CSc.
phone: +420 221 912 387
e-mail: jirousek@troja.fjfi.cz

Department of Solid State Engineering

Czech Technical University in Prague
Faculty of Nuclear and Physical Engineering
Trojanova 13
120 00 Praha 2, Czech Republic