A real-time programming language and its processor for digital control of industrial processes

by LIANG LIANG

INTRODUCTION

The acceptance of digital control for large scale industrial processes, such as chemical, refining, power, and material processing is a step towards total automation and a further improvement in control system performance. Process control oriented programming languages provide a control engineer with means to learn and hence set up, modify, and operate a control system with ease. They eliminate the complication of involving a programmer, who generally has little knowledge about process control engineering. The result is a reduction in time and dollars for digital control system generation.

Such a language and processor has been designed and implemented. It has many desirable features: process control engineering orientation, independence of machine, modular structure, convenience of capability expansion, and conversational mode for on-line design and modification. The basic concept to implement a digital control system is borrowed from analog control. The system is created by connecting elementary control modules together. The rationale is that most experienced process control engineers and field-proven control schemes are still heavily analog control oriented. On the other hand, the language can readily be used to implement highly interacting, nonlinear, feedforward, self-tuning types of control. Thus the language serves an immediate need in the process control field while awaiting the breakthrough of large scale multi-control-variable manipulation at the regulatory control level which seems still remote for actual field application.

Specification

Format

A control system is to be implemented modularly. The basic elements are statements and functional blocks. Each block may consist of a number of statements.

<table>
<thead>
<tr>
<th>P</th>
<th>OP</th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prefix</td>
<td>Operator</td>
<td>Operands</td>
<td>Comment</td>
<td></td>
</tr>
</tbody>
</table>

Figure 1—Statement format

The basic format of a statement is a binary operation with an operator and three operands. For example: ADD, A, B, C means A + B = C.

The prefix could be the label of the last statement of a loop or a transfer entry. Comments could be added following the operand field. Such a statement format simplifies the syntax of the language and hence the source compilation. Although the format is simple, it is wholly adequate for this application. A variety of operations can be implemented. These include: arithmetic, control and logic functions, I/O handling, matrix, and decision table manipulation. The resulting control system resembles an analog control system with statements replacing analog control modules and blocks of statements replacing cabinets of control modules. Program modularity has further advantages for programming and man-to-man communication, adaptability to change and expansion. Beyond these convenience aspects, this organization makes it possible to implement some important features, such as block execution according to priority (handling emergencies), sequence (process startup), sampling period (process normal control), and time-sharing (to utilize computer free time), on-line design and modification, and block execution time estimation.

Control system decomposition

It is left to the judgment of the control engineer as to how a control system should be decomposed. Consider as an example, a control system is to be designed...
for a paper making process. The control engineer may draw up the control schematic as he used to and may divide the control system into blocks according to functions (stock consistency control, pressurized headbox control, multi-effect evaporator control, etc.)

**Source coding**

A simple example is used to illustrate the coding of a source program. A single control loop is assumed. The control engineer assigns names to variables and parameters and starts to code the source program directly from the schematic.

- a. EBK, PRESS
- b. DBK, TEMP, 5
- c. DNS, DFS
- d. SUB, RT8, YT8, ET8
- e. INT, ET8, K15, D3
- f. . . .
- g. EBK, TEMP
- h. . . .

with the interpretation:

- a. Declare the end of previous block PRESS;
- b. Declare block TEMP to be executed every 5 seconds;
- c. Declare number system to be used for the block: decimal, floating point, single precision;
- d. Subtract measurement YT8 from setpoint RT8 and obtain error signal ET8;
- e. Integrate error signal ET8 with integral gain K15 and generate driving signal D3;
- f. More statements may be added;
- g. Declare the end of block TEMP; and
- h. More blocks may be added.

**Rules and procedures**

A fixed statement format is used. Functional blocks must be declared. Both transfers and looping must be confined to the same block. Statements may be entered one by one through a teletype or in a lump sum through reading a stack of cards. During source compilation, the block execution time is estimated as a check on the proper assignment of sampling period to the block.

**On-line design and modification**

During design, check-out, installation or operation phase, changes made on a control system can be anticipated. Thus it is desirable to implement changes on-line, without recompiling the total source program and without affecting the normal running of the process and the control system.

Here, the control engineer refers only to the source program and communicates with the computer in a conversational mode. A version of the source program...
is kept in the memory and is updated automatically whenever a modification is made. The following on-line modification can be made:

a. Insert a block;
b. Delete a block;
c. Restore a deleted block;
d. Modify block sampling periods;
e. Redefine decimal place of a block (fixed point number);
f. Insert one or more statements;
g. Delete one or more statements;
h. Modify parameter or variable values; and
i. Modify the state of a Boolean variable.

Some typical illustrations are:

a. ISB, CONTL, 10;
b. ISS, CONTL, 5, 1
   ADD, SPE, STE, TQ;
c. DLS, CONTL, 3, 2;
d. MSP, CONTL, 5;
e. MPV, K15, 12.5;

Their interpretations are:

a. After the teletype is in Modify Mode, the control engineer enters the operation code of inserting a block, ISB; block name CONTL; and block sampling period of 10 seconds. A comma is used to separate items and a semicolon is used to terminate a message. The latter initiates on-line compilation. The new block will immediately be integrated to the operating program and will be executed once every ten seconds;
b. Insert one statement SPE + STE = TQ into block CONTL following the fifth original statement;
c. Delete two statements from block CONTL starting from the third statement;
d. Modify sampling period of block CONTL to five seconds; and
   e. Modify the value of parameter K15 to 12.5.

The evaluation of the result of a modification requires human judgment. The control engineer is expected to know what he is doing. A more advanced approach is to let the computer accept or reject a modification.

On-line design of a control system is an application extension of on-line modification. That is, one can practically start from scratch and build up the control system by inserting blocks and statements while the control system is running.

Implementation

The language processor is functionally divided into three major parts: Supervisor, Compiler and Modifier. The Supervisor coordinates and schedules the proper execution of all the programs (language processor, man-machine communication, utility programs and control system). The Compiler translates all source entries into machine language object code. The Modifier realizes on-line modification. The implementation of the Compiler will not be discussed. However, the highlights of the Modifier will be given.

After the Supervisor identifies an input as an on-line modification operation, the Modifier takes over and the proper subroutines are called to process the request. Assume an input ISB, CONTL, 5; is entered. The corresponding source will be:

DBK, CONTL, 5 (Declare block)
EBK, CONTL (END of block)

and the object listing becomes (expressed in assembler language for clarity):

Q1 TRUQ2 (Transfer)
Q2 TRU SUPERVISOR (Transfer)

where Q1 is the entry point for transfer from the Supervisor. The first object instruction is a transfer to the second instruction, and the second, a transfer back to the Supervisor.

The corresponding data file will be updated (block name, type, sampling period, execution condition, number of statements in the block, starting and ending location of the block and of each statement in the block). The new block will now be executed every five seconds. Assume the statement ADD, A, B, C is to be inserted into block CONTL following the first statement. The source becomes:

DBK, CONTL, 5 (Declare block)
ADD, A, B, C (A + B = C)
EBK, CONTL (End of block)

The new object listing is:

Q1 TRUQ3 (Transfer)
Q2 TRU SUPERVISOR (Transfer)
Q3 LDA A (Load)
ADD B (Add)
STA C (Store)
TRUQ2 (Transfer)
where patching method is used. Suppose the statement
ADD, A,B,C in block CONTL is to be deleted. The
source resumes its original form:

```
DBK, CONTL, 5 (Declare block)
EBK, CONTL (End of block)
```

The object listing becomes:

```
Q1 TRUQ3 (Transfer)
Q2 TRU SUPERVISOR (Transfer)
Q3 TRU Q4 (Transfer)
ADD B (Add)
STA C (Store)
Q4 TRU Q2 (Transfer)
```

Thus the deleted statement will be excluded. The
implementation of the rest of the modification opera­
tions entails little difficulty and will not be discussed.
But there are several design considerations worth
mentioning. Error messages should be typed out when­
ever a wrong entry is made. Whether the object codes
corresponding to a source statement is a direct insertion
or branching to a subroutine, the choice should be an
appropriate balance between memory space and execu­
tion time. Safety features should be incorporated in the
object program, such as saturation on arithmetic opera­
tion overflows. The capability of the language should be
tailored according to needs. Much of the sophistication
can be excluded for many simple applications. Even
though only teletype has been mentioned, the use of a
CRT or a console with push buttons as man-machine
interface is feasible. Only minor modification of the
language processor is required.

Experiment

The purpose of the experiment is to demonstrate the
use of the language to design a control system on-line.
A simplified heat exchange process with two inputs and
two outputs was simulated by an analog computer.

The simulated system hardware set-up is shown in
Figure 5.

```
Figure 5—Simulated system hardware set-up
```

Two single control loops were designed with power
generation controlling combustion rate and final tem­
terature controlling fluid flow. The control schematic
was prepared as in Figure 6.

The algorithm of the other control loop was the same
except with different naming of variables and para­
eters. While the process was running, the first control
loop was declared and then statements were inserted
one after another until the control loop was constructed.
The control gains and set-point were set respectively.
The source listing of the first control loop was:

```
DBK, CTL1, 2 (Declare block)
SBT, IØ, PG (IØ → PG)
SBT, KØ, RPG (KØ → RPG)
SUB, RPG, PG, EPG (RPG - PG = EPG)
SBT, EPG, O2 (EPG → O2)
MUP, K2, EPG, CR1 (K2*EPG = CR1)
INT, EPG, K3, CR2 (∫ K3*EPGdt = CR2)
ADD, CR1, CR2, CR (CR1 + CR2 = CR)
SBT, CR, OØ (CR → OØ)
EBK, CTL1 (End of block)
```

```
Figure 6—Combustion rate-power generation control loop
```

From the collection of the Computer History Museum (www.computerhistory.org)
In the same way, the second control loop was constructed. The results shown in the following diagrams were highlights chosen from a recorder which was running while the control system was being designed online. It must be noted that they look the same as those obtained by a digital control system designed off-line. However, there is a difference that this control system was gradually built up block by block and statement by statement while the process was running.

CONCLUSION

The language has been recognized by individuals with relevant training, to be quite easy to learn and use. The organization makes both the language processor and the control system flexible for change and expansion. The statement format is simple but adequate. Source compilation is straightforward. On-line design and modification which resembles wire-patching on an analog computer has been demonstrated to be feasible. Although the language is intended for industrial process control, it can be used for process modeling and scientific computation. In addition, it can be integrated into other programming languages for broader application. The language is also effective for laboratory use as in experimenting with digital control schemes.
ACKNOWLEDGMENTS

This paper is extracted from the author's thesis fulfilling the requirement for a MSEE degree (1967) from Case Institute of Technology with support from Bailey Meter Company.

The author is grateful to his thesis advisor, Professor M. J. McCann. He is also grateful to Professors M. D. Mesarovic, I. Lefkowitz and J. D. Schoeffler of Case Institute of Technology, and to Mr. H. T. Hoffman of Bailey Meter Company for their inspiration.