The development of process control software

by JAMES D. SCHŒFFLER

Case Western Reserve University
Cleveland, Ohio

INTRODUCTION

The use of on-line, real-time computers for control of industrial processes has been increasing rapidly during the past ten years. That the cost of the software necessary to implement such systems exceeds even the hardware costs became clear in the initial installations. As a consequence, much effort was devoted to the development of efficient, economical software and software approaches for use in industrial process control applications. During the past five years, there has emerged from these efforts the realization that process control software is different from software for large scale batch processing, time-sharing, message switching, or any other computer applications. It contains its own requirements and problems and leads to a distinct set of solutions. Moreover, this difference is due to more than the size of the computers involved in industrial process control. The objective of this paper is not to catalog the significant features of various software systems in existence today for this has been done very well in a number of recent survey papers. Rather, the objective is to describe the basic structure of current industrial process control software, emphasizing the unique structure of that software, how it evolved, and its current points of controversy and problems.

THE PROCESS CONTROL APPLICATION

The source of the uniqueness of process control software is due to the nature of the application, environment, vendors, and users. There is a large variety of processes in the paper, rubber, chemical, petroleum primary metals industries. Control of such processes generally involves a set of plant instrumentation including sensors, transmitters, and special instruments which provide measurements of physical and quality variables important in the process application. Such measurements may be continuous in time (temperatures, pressures, flows, thickness, speed) or discrete in time (concentrations of a stream as measured by an on-line chromatograph from periodic samples from the stream, average cross machine basis weight as measured by a scanning beta gauge, quality variables which are measured off-line in a laboratory). Process control involves the use of these measurements in order to help the human operator more economically run the process. This objective may take one of several specific forms. For example, regulation or direct control has as its objectives the maintenance of critical process variables at prescribed values called operating or setpoints. The need for regulation of pressures, temperatures, speeds and the like, are obvious. Quality variables too may be regulated as for example maximum moisture in a sheet of paper or thickness of sheet steel. Regulation involves the basic feedback process: the comparison of measured and desired values (set points) to produce an error and the use of that error signal via a control algorithm to change some manipulated variable by a physical actuator. For example, the regulation of tank liquid level might be carried out by manipulating the position of a valve in an input stream. Such control is relatively fast, involving sampling of hundreds of variables several times per second, and outputting to actuators at the same rate. The periodic nature of this task is one of the dominant features of industrial process control in that the time sharing of the computer among many such tasks results in critical response requirements on the hardware and software. Control is a tool rather than the end in itself and necessarily is modified from time to time as the process needs or physical structure change. The application, however, does not permit the taking of the computer off-line for program preparation but rather demands that the software be capable of modification while the process is running on-line with attendant guarantees and assurances of safe reliable bug-free operation. Hence, an essential ingredient in process control applications is operator integration which takes place
through special purpose consoles oriented toward the particular application. From these consoles, an operator may examine any variable in the system, produce summary logs of the operation of the process, demand changes in parameters being used for control of the process (setpoints for example), and may even call for changes in the structure of the control system: adding variables to be scanned, adding control calculations for regulatory purposes, or deleting control loops. This implies that all of the parameters and data associated with the process control regulation program must be in a form which can be mnemonically referenced and modified by an operator. Thus, the database associated with regulatory control cannot be imbedded in the programs. This leads to the special structure of this software.

Supervisory control differs from regulatory control in that the objectives are not the maintenance of process variables at particular values but rather the determination of those operating or set points. For example, operating guides are operating conditions which have been deduced during previous successful operations of the process. When similar conditions hold, the computer directs the operator to use these operating points or else justify any deviations. This takes away the option of operating the process in a safe but uneconomical manner.

Optimization of operating conditions may be the goal of the supervisory control system. Optimizing control is the determination of set points from an analytical performance criterion plus a mathematical model of the process. Successful optimal control has been carried out in the petroleum, steel, and electrical power industries among others. Supervisory control is characterized by much more complex programs than regulatory control but carries with it the same demands on reliability, centralized database, and operator communication. That is, mathematical models of the process imply knowledge of the process state which in turn implies knowledge of all the variables, scale factors, offsets, calibration factors and other parameters which make up the process database. The operation or status of every instrument, actuator, and control in the process is needed to determine the current state of the process. All of this information must be made available to the operator on demand and so it too must be organized in a central, accessible database.

Many special purpose programs are necessary to support the basic application including calibration, hardware checkout, special logging and data reduction programs, startup and restart routines, and general programs to maintain and update the database, including tables of mnemonics. In addition, the functions which are not automatically scheduled according to time or event information must be available on demand of the operator, implying an on-line, interactive command interpretive program. All of these many programs must co-exist, interact through the data base and executive control programs and hence lead to a far from trivial implementation problem. The result is the current process control software organization described in the following sections.

**PROCESS CONTROL SOFTWARE**

For a very brief interval, it was thought that process control software would, after initial debugging, run without change for a long period of time. If this were true, many of the problems of the initial systems would not have existed and the current structure of process control software might be radically different. The process changes often, necessitating similar changes to the software. The chief characteristic of modern process control software is its ease of change, permitting on-line modification of control algorithms and even addition and checkout of programs while the system is running. The typical computer used for industrial process control during the last five years is what today would be considered a medium-sized machine, significantly larger than today's minicomputer in the sense of attached peripherals. As a consequence, it was necessary to provide an executive control program to oversee the multiple interacting programs operating in a real-time, concurrent manner. This need, plus a desire for ease of modification and the demand for a flexible, general process data base leads to the structure shown in Figure 1.

That figure shows a conventional real-time executive control program (a monitor) along with a process data base and application programs. The distinctive part which makes it uniquely process control software is the set of application packages which are designed to carry out specific tasks in a very efficient manner while at the same time permitting easy communication among themselves, the executive, and the application programs.

**Process control executive**

Successful process control involves the implementation of many concurrent tasks. Process control programs doing data acquisition, control calculations, analog and digital output, operator communication and the like call for some multiprogramming capability. Executive control programs recognized this and incorporated the multiprogramming in one form or another. Early executives organized core into two partitions, one containing
permanent resident programs (those requiring fast response or maintaining control system integrity in case of failure of the bulk storage system) and the second being used to swap program modules called coreloads in and out of core. A coreload consists of one or more mainline application programs plus subroutines used by these programs and not contained in the permanent resident software. Coreloads may have different priorities in which case the executive often interrupts execution, swaps the coreload out (saving it on bulk storage), swaps in and executes the higher priority coreload and later restores execution of the interrupted coreload.

Multiprogramming based on coreload swaps proved too slow for many applications. This led, of course, to a need for large and larger core storage so that critical programs could be permanently core resident. This motivated a change toward a more dynamic allocation of memory. The most common system is a multipartition system in which all programs assigned to a partition run with the same priority. This implies that a program, once loaded into a given partition and execution begun remains in that partition until its task is completed. Then the next scheduled task for that partition is swapped in to replace the completed program.

The partition system recognized a perhaps nonobvious fact of life of process control systems: the critical resource is not main memory but rather the bulk storage channel (usually the disk). This is especially true of systems based on moving head disks of course. Running tasks to completion minimizes the number of bulk storage swaps while maintaining the priority levels of tasks.

The multipartition system requires the allocation of a program or task to a partition and the assurance that all such programs fit in a given partition and have the same priority. This requires a careful analysis of the application but results in a smooth, controllable real-time implementation.

Most machines today are of the 16 or 24 bit word length variety. The 16 bit machines in particular take advantage of the partition system because there is no need for dynamic relocation of programs. The 24 bit machines can relatively address ±8K of core, a value sufficient to permit dynamic relocation of most process...
control programs (but not of the data base). In some executives, core is divided into partitions with some of the partitions being used as indicated above but with one dynamically allocated by the executive. The latter facilitates background Fortran-level processing.

Process control executives contain extensive error recovery capability because of the difficulty of restarting the system in case of a crash with critical data in core. Error recovery includes backup of I/O devices where feasible, specification of alternative devices, multiple tries at attempting reading of bulk storage, automatic power fail program backup, and automatic maintenance of copies of critical files like the process data base.

Such executives permit multitasking in one form or another but not usually in the completely general case. That is, tasks in a partition are permitted to schedule (turn on, turn off, delay, or queue) other tasks in that and other partitions. However, the allocation of the task to core is predetermined and its priority is not dynamic, but rather pre-assigned. This has proved necessary not only because of the difficulty of dynamic relocation but also because of the need to guarantee a maximum response time for critical real-time programs.

I/O in these executives is handled in the straightforward manner, using I/O drivers and I/O request subroutines. Drivers are responsible for the maintenance of the interrupt driven I/O with the actual device, outputting or inputting one character or word at a time usually with one interrupt per data item transferred. Little I/O is done through direct memory channels except in the case of bulk storage devices. The I/O request subroutine is used to control the competition among many tasks at different priority levels from interfering with one another. Such routines queue requests, notify routines (reschedule them) when a requested I/O operation is complete, test device status, etc.

As long as the volume of I/O is low or the response time of tasks not critical, such an organization is quite sufficient and essentially like any other application. That is, none of this software is unique to process control.

Two situations arise, however, which call for unique process control software. They are: the need for high volume, very flexibly organized process I/O; and the need for mnemonically oriented operator communication. These require special packages for their implementation and are discussed below.

Special packages for data acquisition and control

The need for generality and flexibility in the typical industrial data acquisition system is paramount. Not only is it desirable to be able to change the parameters of the data acquisition loop or the status of a sensor from active to inactive, but also to be able to add and delete from the list of points in a flexible fashion. Thus, if a sensor is added to the system, it is desirable that the operator be able to add the point to the list of converted values, including all the appropriate parameters necessary for the conversion and all the linkages to control programs associated with the variable out of limits, bad data, and so on. This, of course, can be done by reprogramming the routine, but is undesirable and not necessary. The basic operations involved in converting a point are similar from point to point. The scan routine uses the address of the point, values for the amplifier gain, limits, etc., and in effect, applies similar sequences of algorithms to each of the points to be converted. A data acquisition package takes advantage of this as shown in Figure 2 which shows an interpretive scan program and a set of data structures called loop records, with one loop record associated with each point to be converted. All the data and parameters associated with a particular point is stored in the record, including the

**Table of Loop Records (part of process data base)**

- kth loop record containing all data and parameters for data acquisition and direct digital control of kth control loop

**Data Acquisition and Direct Digital Control Package**

- Interpreter Program
- Scan each loop record periodically
- Carry out appropriate actions by calling common routines using data and parameters from the loop record
- Analog Input Module
- Scan Alarm Module
- Engineering Units Conv.
- Control Algorithms
- Filtering Algorithms and others

**Figure 2—Example of data acquisition and direct digital control package**

From the collection of the Computer History Museum (www.computerhistory.org)
period or time between conversions, status of the point, external names, etc. The interpretive program scans the records in some sequence until it finds one which is scheduled for conversion. It then examines the loop record to determine the appropriate data and algorithms to be carried out on that data point. Pointers to the individual algorithms to be carried out on the point, followed by the parameters to be used by that algorithm, are stored in the loop record. If any error is detected in the conversion, an error recovery routine is entered and executed. This might include an attempt to reconverst the point or to estimate the current value in terms of the previous value (which is stored in the loop record), or preparation or notification of either a program or the operator. The interpretive program then continues through the loop records, is offset and scaled. After conversion to engineering units, the data is limit checked for validity, alarm checked, and perhaps digitally filtered by a moving average filter involving the current value and the previous several values. The set of loop records is a very well organized process data base. The loop record concept is a very flexible one in that the actual algorithms carried out on any particular data point may vary considerably by simply changing the parameters and pointers in the loop record.

The only portion of the program particular to a specific set of data points to be read is the information stored in the loop records. It follows that the interpretive scan program can be written and debugged without actual knowledge of a particular application except that all algorithms which might be needed in the application must, of course, be provided. Individual points can be added or deleted by creating or deleting one of the loop records. Because of the fixed format of the system, it is possible to organize the various loops so that not all points with the same conversion period need be carried out at exactly the same instant of time. That is, points with the same conversion frequency can be subdivided into smaller groups which are converted at the correct frequency but offset or phased with respect to one another in time.

The efficiency of such a system must be considered. The interpreter scans through items in a loop record and in effect calls small modules or sub-routines to carry out the desired operations. There is a certain amount of overhead associated with scanning through the loop record, determining which routine is to be carried out, transferring to this module or routine, as well as extracting from the loop record the parameters and data needed by that algorithm. If the module or algorithm is too small, it follows that this overhead involved in scanning, transferring and extracting parameters, may be comparable to or even large compared to the time to carry out the algorithm. The result would be excessive overhead, for inline code to carry out the same task would be more efficient. The objective is, of course, to use modules whose execution time is large compared to the overhead or scan time so that no real overhead penalty is incurred. The overhead associated with interpreting the loop records is offset by the flexibility of the system. Most important is the ease of operator communication with the data base.

If this can be done, of course, the generality and flexibility of the interpretive loop record organization can be achieved without discernible overhead penalties. Whether or not this can be done is dependent upon the organization of the loop record, the instruction set of the computer, and the addressing modes available in the computer.

Control loops lend themselves to a similar loop record organization. Additional functions (filtering, control algorithms, error checking, outputting, etc.) are listed in loop records often along with the data acquisition information and interpreted by the same or a similar routine. The result is a separate data base and data acquisition and control package which may be interrupt driven independent of the executive or possibly assigned a high priority level and controlled by the executive.

The special purpose package relieves the application programs from considering the data acquisition and direct control task in great detail including error checking, organization of operator communication, and startup. Other applications are similarly best handled by special purpose packages. These include sequencing control—monitoring the state of the process and causing events to occur (turn on burners, valves, etc.), in the correct sequence and when prescribed conditions are met. Examples include batch processing of chemical processes, control of annealing lines, and the like.

Operator interaction becomes a straightforward task with a data base arranged as described above. Interface routines to the data base provide application programs the ability to read or write any particular item in any record (including a full complement of error checking). Copies for backup and restart are available because of the centralized location of data as opposed to its distribution throughout application software.

Operator interaction through a console limits the amount of error checking necessary and makes a very acceptable system for use in an on-line situation with operators who are not trained computer operators. Function keys are the most common communication scheme with labeling in industry-sensible terms. Functions limit the complexity of the operator console support routines but nonetheless are nontrivial in size.
In larger systems, it is convenient to separate the data base into two parts, that which is essential to in-core reliable operation in case of disk failure and the remainder which is then swapped in and out of core (perhaps on a single record basis) as needed.

In addition to special packages which are unique to process control, other application programs are provided which may interface to the executive and the special purpose packages. Optimization is an example, permitting optimal control of a process unit provided a mathematical model and sufficient measurements are available. These differ from other application programs only in that they are "canned" rather than written by the user in a higher level language. Typically, they interact considerably with the data base and its support routines.

SOFTWARE PREPARATION FOR PROCESS CONTROL

Current medium-sized and larger systems universally provide the ability to write application programs in higher level languages, most notably, extended Fortran. This is both to limit the level of expertise of programmers but also to permit control engineers to produce application software. With intelligent use of data acquisition and control data base packages provided by the vendor and the general process control real-time executive control program (including all its entries for scheduling, error checking, I/O control, etc.), higher level language programming becomes quite feasible, especially in applications which involve significant computation. Other applications are developed more efficiently via packages like the data acquisition or direct digital control packages described above. These are usually programmed in assembly languages and linked tightly with the executive. In some cases, they are assembled as part of the executive.

The net result in Figure 1 is an executive which does not control all I/O but only that which is shared among application programs. Special purpose packages maintain a centralized data base and perform specialized control functions. Application programs perform tasks on a somewhat slower basis, but interface extensively to these packages through normal system subroutines.

Because of the need for flexibility and change, on-line program preparation is needed in most process control systems. This takes the form of a background command processor, compilers and assemblers and library maintenance routines as well as link edit routines. Program preparation including debugging can be done in a controlled manner in one of the partitions, without upsetting the remainder of the software system.

In recent systems, macro assemblers have become available, even including libraries of system macros which permit a user to custom tailor the operating system software to his particular application (at least in the minicomputer process control systems). At present, few cross assemblers are available and even fewer cross compilers for software preparation on time-sharing or batch data processing systems.

Much more significant is the preparation of data inputs for the dedicated application packages. The data base contains all the information needed for the carrying out of these dedicated tasks. The operator console, with its function keys, provides on-line data entry, program change, control application modification, and the carrying out of operator commands. Off-line program preparation is through translators of one form or another. The natural language for the application might be a block diagram indicating the interconnection of the measurements, setpoints, filtering, control and feed-forward algorithms. Specified this way, they must be translated to data to be stored in the internal data base for interpretation.

Many such communication languages have been developed in order to provide a means for process engineers who are not trained programmers to set up and maintain an application involving hundreds and even thousands of variables. Such systems specify the application in an English language fashion which is then translated to internal data storage. Others are of the "fill in the blanks" form which is a questionnaire which is completely self documenting (and 100% of the documentation) and which, when punched line for line on cards, provides input to a translator which generates interval entries in the tables of the various packages.

With these programs, an application can be put on-stream very rapidly since the essential programming is done and debugged independently of the particular application. Moreover, the change from one control structure to another (one startup procedure to another, etc.) is very straightforward, amounting only to change of data but not essential reprogramming. There is little penalty incurred in changing the control software, even less than in the case of Fortran programming of application programs. As mentioned in the next section, this package concept is less economical today because of the introduction of minicomputers.

The other trend today is toward the use of "real-time languages" for process control, even involving an attempt at a worldwide standard. There is a difference of opinion as to the economics of standard process control languages. As noted in Figure 1, the bulk of process control software lies in the executive and the application packages. Application programs tend to be written to use these programs and hence are less expensive (the
complexity of the real-time application is imbedded in the application packages and the executive). One school of thought advocates the need for a higher level system programming language for the economic and efficient production of the executive and the application package or at least the tailoring of these standard packages to a given configuration and application. The other school of thought argues that these should not be specialized or customized but be standard so that they can be supplied by any vendor (and at reasonable cost since they can be written off over a large number of installations). This school of thought then argues that the remaining cost is in the application programs and that a language ought to be adopted for this purpose (extended Fortran, or some modification of PL/1 or Algol are among the suggestions).

It appears that no economic study has actually been carried out to determine the savings which might result from either of these approaches. The latter case already provides extended Fortran so that only incremental gains can be expected. The former case is difficult to assess because it depends upon the need for changing of standardized software packages or adaptation of these to various configurations of computers. If it were not for the minicomputer, data could eventually be gathered. However, as indicated in the next section, the minicomputer is changing the economics of process control software significantly.

CURRENT PROBLEMS IN PROCESS CONTROL SOFTWARE

The development of process control software over the past five years or so was directed toward the organization of Figure 1. Recently, packages for data acquisition and direct digital control, sequencing control, batch processing, process optimization, model identification, graphic operator communication, and others were developed. If the use of medium-sized machines continued, this software would have resulted in significant decreases in costs of process control software implementation. Several problems, however, indicate that further development is necessary.

First, the use of a medium-sized machine is difficult to justify economically in many applications. The capability of such machines is such that many tasks can be carried out by a single machine. This, in turn, places severe demands on the executive and application software in order that the installation can be carried out smoothly. Moreover, the process itself must be large enough to justify such a machine and such extensive software costs.

The advent of the minicomputer with its low cost has changed the picture considerably. The lower cost of small configurations implies that several tasks need not be implemented in a single machine in order to justify the use of a computer for control. This, in turn, implies that perhaps the extensive software developments of Figure 1 are not necessary for any smaller applications. The result has been the use of minicomputers for many small applications which in previous years had used larger machines. This, in turn, implies that the number of systems to write off software package development has decreased considerably. Hence, larger systems with very sophisticated software packages have recently been more difficult to economically justify.

The minicomputer poses a set of problems which are considerably different. First, the varied configurations and applications do not always require complex executive or monitor systems. Consequently, there is less justification for the development of extensive executives on the part of vendors. The result is sometimes more software development for the minicomputer system than for a larger installation, robbing the system of its apparent economic advantage over the larger system.

A second source of difficulty is the multicomputer configuration. The use of minicomputers for dedicated applications still requires the coordination of the systems to achieve overall process control including scheduling and higher levels of control. Hooking together a number of small machines each implementing a portion of the application is far from trivial for there is still a need for a coordinated data base and operator communication even if the application is distributed over a number of machines.

A related problem affecting the development of software for smaller machines is the rather commonplace prediction that the bulk of sales for systems will lie in discrete parts manufacturing rather than process control. The software necessary for this application has not been widely developed and agreed upon but will certainly affect the organization and structure of future minicomputer process control software.

The current question in process control software is how to economically produce the special purpose real-time programs necessary for a low cost but efficient implementation in a stand-alone minicomputer. In addition, how can the advantages of packages for specific applications be gained since it is difficult to develop them such that they are compatible with the varied configurations and computer systems arising today? Perhaps the answer is a higher level systems programming language with program preparation on a host machine. Perhaps the answer lies in generalized software preparation schemes such as are used to generate execution programs for large data processing machines. Perhaps the answer lies in the use of sophisticated...
template macro assembly languages. Whatever the answer, the well developed process control software concept is today facing a period of rapid change similar to that of five years ago.

REFERENCES

1 H E PIKE
Process control software
Proc IEEE Vol 58 No 1 Jan 1970 pp 87-97

2 J C MECKLENBURGH P A MAY
An engineering comparison of process control languages
Dept of Electrical Engineering Univ of Nottingham

3 Special issue on computer languages for process control

4 P R WETHERALL
Corall 60—A language for the control of real time processes
ibid 2 p T173 1969

5 D G BATES
PROSPRO 1800
IEEE Trans Indust Electronics and Control Instrumentation IECI-15 No 2 p 70 1968

6 S J BAILEY
Pushbutton programming for on line control
Control Engineering 15 No 8 p 76 1968

7 G W MARKHAM
Fill-in-the form programming
Control Engineering 15 No 5 p 87 1968

8 T G GASPAR Y V DOBROHOVOFF
D R BURGESS
New process language uses English terms
Control Engineering 15 No 10 p 118 1968

9 R E HOHEMEYER
CDC 1700 fortran for process control
IEEE Trans Indust Electronics & Cont Instmn
IECI-15 No 2 p 67 1968

10 T L WILLMOTT
A survey of software for direct digital control
ISA Paper 68-834 October 1968