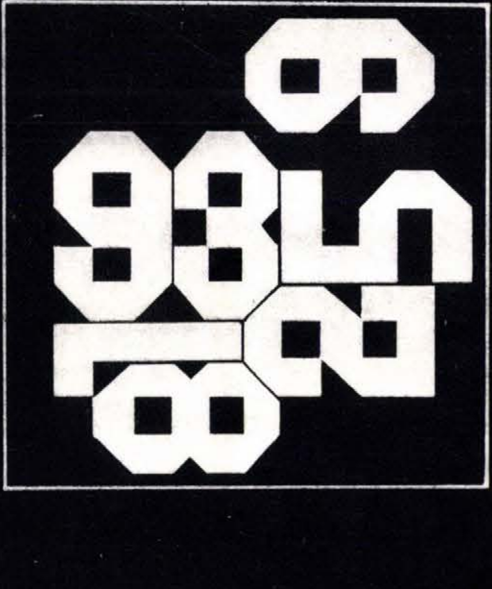
tanulmányok

19/1974

MTA Számitástechnikai és Automatizálási Kutató Intézet Budapest



SOFTWARE FOR PROCESS CONTROL
SURVEY

PREPARED FOR THE 4TH INTERNATIONAL IFAC/IFIP CONFERENCE ON DIGITAL COMPUTER APPLICATIONS TO PROCESS CONTROL

by

Dr. Janos Gertler
Computer and Automation Institute, Hungarian Academy of Sciences, Budapest, Hungary

Dr. Jan Sedlak
Institute for Industrial Management Automation (INORGA), Prague, Czechoslovakia

Published by the Computer and Automation Institute, Hungarian Academy of Sciences, Budapest - Hungary.

Készült az OMKDK sokszorosító üzemében
F. v.: Janoch Gyula

This paper is an attempt at presenting a survey on Software for Process Control. Realizing the extreme extent of the field, the authors intended to concentrate on the most relevant subjects and adopted the following structure:

1. General properties of process control software
   - 1.1 Specialities of process control computer applications
   - 1.2 Structure of process control software
   - 1.3 Preparation of process control software projects
2. The present status of process control software
   - 2.1 Real-time executives
   - 2.2 High-level general-purpose process control languages
   - 2.3 Application packages and problem-oriented languages
   - 2.4 Man-machine communication software
3. Servicing of process control software
4. Standardization.

Work was shared according to the authors' experience and interest. Thus Sections 1. (1.1 through 1.3), 2.4 and 3. were written by J.S. while 2.1., 2.2., 2.3. and 4. by J.G.

It should be noted here that process control software is a field where commercial interests are rather significant. The special position of the authors made it possible to take a most unbiased approach.

1. GENERAL PROPERTIES OF PROCESS CONTROL SOFTWARE

In this introductory section, some general properties of process control software will be dealt with, like

1. Specialities of process control computer applications
2. Structure of process control software
3. Preparation of process control software project.

1.1 Specialities of Process Control Computer Applications

Generally speaking, industrial control systems can be built on the following five levels:

1. Long term planning and strategic management
2. Management and data-banks of industrial systems
3. Order handling and production planning
4. Operative production control
5. Process control

The use of computers in this third application area, especially on the last two levels, brings in a store of new problems, in comparison with the two classical application areas. These areas are the wellknown scientific and engineering computations (SEC) and automatic data processing (ADP). Table 1 presents a short comparison of some traits pertaining to algorithms and computing techniques in the three observed areas. It is concluded that the progress rate in a computer application area is determined primarily by the algorithmic knowledge and by the level of algorithm formulation [E 1] Secondarily, other factors follow, among them especially the capabilities of the computing techniques (i.e. hardware and software) to fulfill all the requirements laid by the respective application area.

The development of industrial control systems shows the following main strides:

- securing the computer-plant-computer feedback information flow
- formulating timing correctly, in order to control the plant in real-time (to ensure the correct time-pace of control)
- ensuring the required ordering of control programs by means of multiprogramming, based on dynamic priorities and time requirements
- processing possible interrupts in the control system caused outside the computer
- ensuring the continuous (24 hour) performance of computer control

TABLE 1. CHARACTERISTICS OF THE THREE COMPUTER APPLICATION AREAS

No. OF ITEM	CHARACTERISTIC PROPERTY	SCIENTIFIC AND ENGINEERING COMPUTATIONS	AUTOMATIC DATA PROCESSING	INDUSTRIAL COMPUTER CONTROL
1.	ORIGIN	1950	1955	1962
2.	LEVEL OF ALGORITHMIC KNOWLEDGE	VERY HIGH	LOW, WITH SUBJECTIVE ELEMENTS	VERY LOW
3.	FORMULATION OF ALGORITHM	CLEAR	HETEROGENEOUS	NOT CLEAR
4.	SURROUNDINGS OF ALGORITHM	SMALL	LARGE	LARGE WITH FEEDBACK
5.	INFORMATION FLOW	SLIGHT	SIZABLE	SIZABLE OF TWO TYPES: MAN - MACHINE MACHINE - PLANT
6.	WORK WITH DATA BANK	NO	YES	YES
7.	ALGORITHM STRUCTURE	SEQUENCE OF PROCEDURES PROCESSED IN SERIES	PARALLEL PROCESSING OF PROCEDURES	PARALLEL PROCESSING OF PROCEDURES IN REAL-TIME
8.	RESULT OF ALGORITHM	SIMPLE (OFTEN A SINGLE VALUE)	COMPLICATED (TABULATION)	CONTINUOUS WORK OF ALGORITHM WITH RESULTS AT DISCRETE INSTANTS OF TIME
9.	TYPES OF PROGRAM	SINGLE PROGRAM	SYSTEM OF USER PROGRAMS CONTROLLED BY OPERATING SYSTEM	- MAN - MACHINE - MACHINE-PLANT - DATA PROCESSING, COMPARISON WITH A MATHEMATICAL MODEL - OPTIMIZATION THE FOUR TYPES OF PROGRAMS ARE CONTROLLED BY REAL-TIME EXECUTIVE SYSTEM (CONTROL PROGRAM)

10.	TESTING OF PROGRAM	STEP BY STEP, USING TEST EXAMPLES WITH INTERMEDIATE RESULTS	SUCCESSIVE: - SUBALGORITHMS - GROUPS OF ALGORITHMS - WHOLE SYSTEM	- STATIC: SUBALGORITHMS, FOUR TYPES OF PROGRAMS (see 9) - DYNAMIC: SUCCESSIVE REAL-TIME TESTING OF CONTROL PROGRAMS WITH FOUR TYPES OF PROGRAMS, USING THE MODEL OF THE PLANT
11.	TUNING OF ALGORITHM	FINDING THE OPTIMUM OF CALCULATION SEQUENCE	SUCCESSIVE ADAPTATION OF THE TESTED PROGRAM FOR REAL EXPLOITATION	AFTER CONNECTION OF THE INDUSTRIAL COMPUTER SYSTEM TO THE PLANT, ANOTHER CHECK AND EVENTUAL CORRECTIONS OF THE SELECTED MATHEMATICAL MODELS, PLANT CONSTANTS, ETC.
12.	COMPUTER	RAPID PROCESSOR, FLOATING POINT ARITHMETIC	MODULAR STRUCTURE OF BASIC COMPUTER + PERIPHERAL EQUIPMENT WORKING WITH ONE TYPE OF INTERFACE, MULTIPROGRAMMING	MODULAR STRUCTURE OF (OR MORE THAN ONE) PROCESSOR + PERIPHERAL EQUIPMENT WORKING WITH ONE TYPE OF INTERFACE, MULTIACCESS UNIT, DIGITAL AND ANALOG I/O UNITS, DMA, TIMER, MULTIPROGRAMMING, MULTIPROCESSING
13.	PROGRAMMING TECHNIQUES	SATURATED	SATURATED	IN DEVELOPMENT
14.	PROGRAMMING LANGUAGES	FORTRAN IV ALGOL 60 PL/1	JOVIAL COBOL PL/1	MACROINSTRUCTION LANGUAGES, ADAPTATIONS OF SCIENTIFIC AND ENGINEERING LANGUAGES
15.	OPERATING SYSTEMS	NOT RELEVANT	YES, WORKING ON THE BASIS OF PRIORITIES; BACKING STORE ORIENTED	YES, WORKING ON THE BASIS OF PRIORITIES AND TIME CONDITIONS, MODULAR STRUCTURE (WORKING ALSO WITHOUT BACKING STORE)
16.	FUTURE OF PROGRAMMING LANGUAGE	ALGOL 68, PL/1	PL/1 (NEW ADDITIONS)	LTPL

end of Table 1.

- enabling control optimization
- carrying out diagnostics and/or evaulation of another algorithm when the control algorithm does not use the computer.

1.2 Structure of Process Control Software

The algorithms of industrial process control in most cases are performed by four types of programs (see Table 1, item 9). These programs are controlled by a control program (real-time executive). An idealized structure of such a system is illustrated in Fig. 1. The lower modules (1,2) handle all the feedback information flows in real-time, i.e., they control data-exchange between man and machine and between plant and machine. For writing these program modules, symbolic-address languages and macroinstruction languages are mostly used. These modules usually work at the top priority levels in the multiprogramming system. Module 4 contains the mathematically formulated algorithms to process the measured plant data, and decision algorithms to produce qualitative and quantitative information to be flowed from the computer into the process. Management and construction of data base for communication with the higher levels of the system is also included in the latter module.

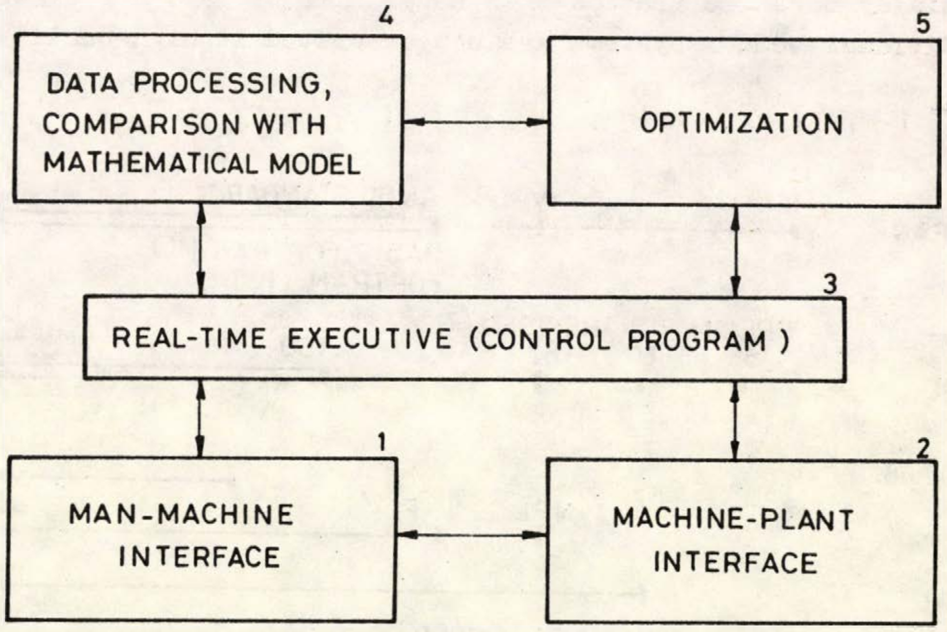

4
DATA PROCESSING, COMPARISON WITH MATHEMATICAL MODEL
5
OPTIMIZATION
3
REAL-TIME EXECUTIVE (CONTROL PROGRAM)
1
MAN-MACHINE INTERFACE
2
MACHINE-PLANT INTERFACE

Fig. 1. INDUSTRIAL PROCESS CONTROL PROGRAM SYSTEM

Algorithmic programming languages are used to write mathematical models. For writing the decision algorithms, however, it is necessary to choose a programming language which can easily describe logical conditions and ordering. These programs are mostly evaluated on lower priority levels. Module 5 contains the programs which modify, on the basis of current results, the control algorithm. These algorithms are often adaptive in nature; they are written in algorithmic languages. Module 5 can be evaluated either off-line or on-line (and mostly on a low priority level) or sometimes its evaluation is called by Module 3. Module 3 controls modules 1,2,4,5. In many systems, Module 3 is the real-time executive as provided by the computer vendor. However, for some particular applications, the vendor's executive is too small (in terms of services) or too large (in terms of overhead). In such cases, special user executives (control programs) are written incorporating (if possible) whole or parts of the vendor's system.

Let us discuss now the structure of process control software. There are well-known vendor programming systems which have reached certain stability. These programming systems are generally noted for their considerable extent and complexity. Exploiting them requires substantially more preparation than usual with SEC and ADP systems. The individual vendor systems are now described in high-quality manuals.

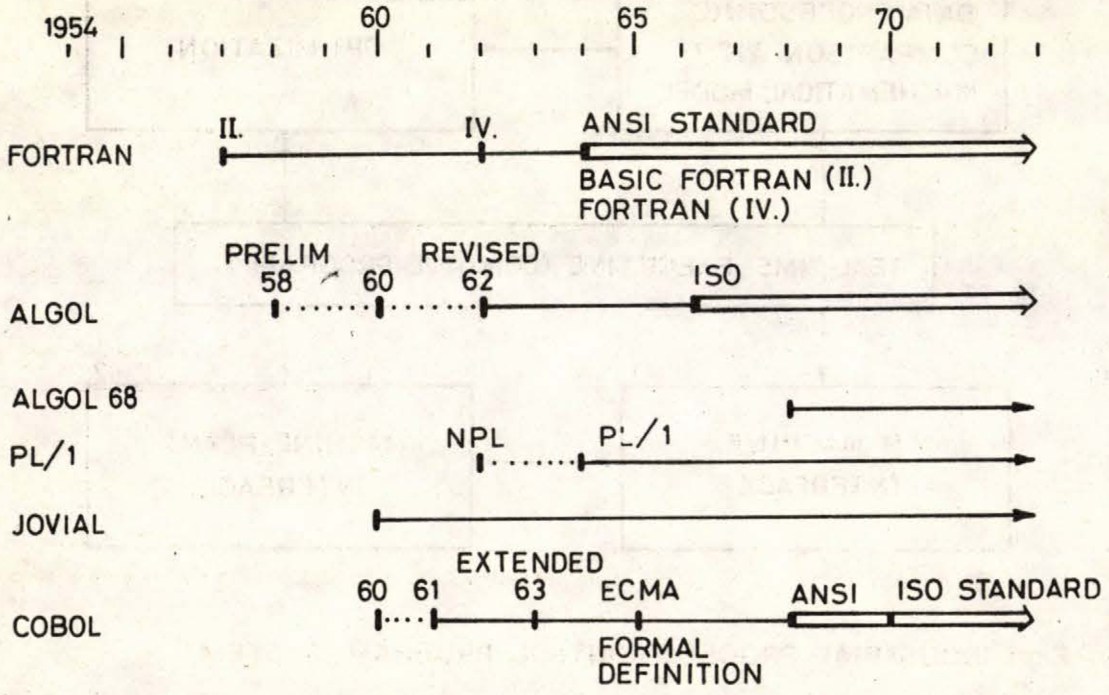


Fig. 2. COMPARISON OF THE DEVELOPMENT OF SEC AND ADP PROGRAMMING LANGUAGES

While process control software of the different vendors differs considerably, a stride for standardization is also experienced (see Section 4.). Unfortunately, process control software as a whole is still in the state of development, in comparison with the SEC or ADP areas. For instance, ALGOL 60 and FORTRAN in the SEC area were standardized as early as in 1964 while COBOL for the ADP area naturally later in 1968 (see Fig. 2.).

Within process control programming, the following components can be discerned:

- executive systems
- programming languages
- application packages
- man-machine communication software.

These components are necessarily accompanied by support software for testing and tuning the complicated program systems.

Executive systems for these purposes have a rather unified function today. They coordinate the execution of the various programs and control the resources of the system, on the basis of time-requirements and external events. The highly modular structure of executive systems enables to do rather efficient process control even without peripheral memory. A more detailed description of the real-time executive systems is included in Section 2.1.

Process control programming languages have their very hard way to standardization. The macroinstruction languages [E 2], so early abandoned in the SEC and ADP areas, have rendered very useful services. It is these languages in which now, in addition to most process control programs, also the basic software for process computers (e.g. executive systems, compilers, etc.) is mostly written. Developments in industrial computer languages are much varied, and can be classified into the following three categories [E 3]:

- adaptations of the SEC languages
- new industrial computer languages
- languages for the non-programmers (e.g., form or dialog based).

The first two trends should in no sense astonish us. They could be observed as early as the origin of the ADP area. The third trend, generally based on a properly chosen macroinstruction system and on a syntactically described library of basic control algorithms, is constructed in such a way that the user-technologist can employ it with success without having any special programmer education.

There are many different users of industrial control computers today. Each of them wants to use his system at different levels.

Therefore the suppliers must obviously have the right hardware and complete and effective software. This is the reason for the gradual spreading/enhanching of application software. Among various application packages, we can find Linear Programming System Packages for on-line process optimization, Telemetry Application Packages, Gas Chromatography Packages, Logic Sequencer for engineering logic sequence diagram manipulation, etc. On the other hand, control computers are equipped with various peripheral control packages and drivers. A choice of them depends on the configuration of the control computer installed. Interactive packages have also appeared for such purposes. Let us mention for instance the plotter package, display package, file manager packages, etc. Special applications packages have been created for terminal communication systems which, beside the special hardware, require special software as well.

1.3 Preparation of Process Control Software Projects

Specialities of the third computer application area appear also in designing industrial control systems. The conventional and characteristic stages, namely

- pilot study
- planning hardware implementation
- development of application
- installing hardware
- control of the real object and improving this control

are well known from the development and realization of control systems. Managing real-time projects forms a special area of computer sciences today. Beside correctly defining the objectives of control and determining the control algorithm, it is necessary to pay a special attention to working team structures. Here the fundamental task is to ensure a mutually understandable inter-team language. This necessity appears as early as during preparatory activities. Co-operation between the technologist (who defines the task) and the system analyst/programmer is a basic requirement. This co-operation starts at the early stage of formulating the control algorithm, i.e., well before programming. Here we can mention two working phases, namely

- system specification
- system analysis for programming.

The technologist should carefully prepare for the co-operation in system analysis by describing his requirements in the form of system specification. From the methodological point of view, it is convenient to progress from a complete list of variables and plant parameters, through specifying their functional relations, to a specification of all the control components present in the system. Thus the following items are prepared:

- a short description of the major control objectives and overall system (including its surroundings)
- a detailed description of the controlled object (plant)
- a specification list of variables and plant parameters and, if necessary, a set of rules for data file ordering and management
- a set of rules for information flows between man - control computer, control computer - controlled plant and also for direct man-plant communication
- a list of algorithms for data processing, for comparison with the mathematical model and for decision-making based on this comparison. Along with this, it is necessary to determine mutual links between algorithms and requirements for their processing in real-time
- a testing example for the whole system and for each of its subsystems. The testing examples should be representative for both static and dynamic tests.

The details of the individual items of system specification are adjusted to the particular control project.

System analysis for programming begins with an algorithm written down in clear form. An internationally standardized algorithmic language, available for writing algorithms, is not yet in existence. Up to now, mathematical languages, technologist languages, flowchart diagrams, decision tables, table layouts with various headings and, for describing time-relations, bar diagrams have been used.

Industrial control systems have mostly been specified without a deep knowledge of industrial computer technique. Therefore, it is very important to perform an analysis of the control algorithm from the point of view of the implementation before starting to write a program.

This phase is finished with a mutually understandable formulation of the control algorithm in a form suitable for programming. No matter whether the process is discrete or continuous, system analysis consists of the following activities:

- checking of the algorithm for consistency and completeness
- removal of unnecessary redundancies from the algorithm (i.e., leaving only redundancies needed for testing)
- implementation of some additions to the algorithm, found necessary in course of the analysis
- clarification of variables, including their scope, range and acceptable processing error
- clarification of the computer approximation of functions designed in the algorithm
- clarification of the control loops from functional and timing points of view (defining the time-pace of control)
- incorporation of the interrelations between individual control loops revealed by system analysis, to be taken into account in timing
- clarification of the activity of the control algorithms during various phases of their operation (starting, running state, alarm state), in relation to another (conventional) control contingent working simultaneously with computer control
- defining the computer configuration for a realization of the algorithm
- checking by means of a testing example, supplied by the technologist, in accordance with a predetermined plan (input data and intermediate results varying with time)
- checking the demands towards the man/process interface (possibility of affecting the process, messages on control states, etc.)
- assignment of program priorities (also taking time into account) in accordance with the designed decomposition of the control algorithm.

A necessary conslucion of this system analysis is the approval by the technologist of a new version of the control algorithm. It is also necessary to determine regulations governing possible further changes by the technologist. It is desirable to limit the number of these changes because the volume of work needed to realize the algorithm cn the computer, after the system analysis, is relatively great.

At the same time, a written formulation of the new version of control algorithm is considered as a documentation for further team work during realization of the algorithm on the computer, and also for later acceptance of the tested program for normal exploitation in the controlled plant.

Experience has shown that system analysis improves the quality of the technologist's original algorithm contained in system specification. Such a course of activities detaches algorithm development from programming. Mixing of these two activities was one of the earlier erroneous courses of work on industrial control projects.

The timely and detailed preparation of the control algorithm thus speeds up system realization. Also, the necessary technical supplements to the computing system can thus be prepared with sufficient time-lead. This proven course of control algorithm development will prevent us from getting into the unpleasant state of "never finished" industrial control systems.

2. THE PRESENT STATUS OF PROCESS CONTROL SOFTWARE

When surveying on the present status of process control software, we will concentrate on four major areas:

1. Real-time executive systems
2. High-level general-purpose process control languages
3. Application packages and problem-oriented languages
4. Man-machine communication software.

Assembly level and macro programming will not be discussed because of its strictly machine-oriented nature.

This part of our survey is mostly based upon a considerable amount of up-to-date written information, placed at the authors disposal by the courtesy of several leading vendors of process control systems. The list of the manuals directly utilized to this work is found in the references. Also a good use was made of an excellent survey by Herbert E. Pike [X 1].

2.1 Real-time executives

The operation of process computers is controlled by time and events. Some programs are due to execute at specific instants of time or after a certain delay or repeatedly at certain intervals. Other programs are initiated by external events originating from the process or operator. Programs just executing may also request some other programs. And internal events, like completion of an I/O operation, require some action of the computer as well.

The house-keeping of process computers is organized by the real-time executive software system (in some systems it is named differently like operating system, director, etc.). The fundamental functions of the real-time executive are

- allocating system recourses (like operative memory, central processor, etc.);
- handling peripheral operations;
- handling events;
- time-scheduling.

Program, priority, data

The system deals with two sorts of code: program and data. The program code describing the job to be done by the computer is broken into pieces. There are 3 major types of program units:

a. Tasks (programs, core-loads). These are relatively large executable and generally relocatable program units. They are activated solely through the executive system and are scheduled mostly on a priority basis.
b. Routines. These are relatively small program units for fast servicing of different events. They are activated by the executive system unconditionally.
c. Subroutines. These pieces of program are activated directly by the tasks and may be [A 5]
    - dedicated, that is available for one task only;
    - common, that is, available for several tasks but not interruptable;
    - re-entrant, that is, available for several tasks and interruptable.

A mark of importance is attached to each task, its priority. Priority may be

- either a permanent attribute of a task;
- or assigned to it when the task is activated.

In some systems, a limited number of priority levels exist (e.g. 7 levels [A 1], 4 levels [A 10]) and each task is attached (statically or dynamically) to a certain level. In other systems, any natural number within relatively wide limits (e.g. 0 to 99 [A 5], 0 to 255 [A 3]) may be assigned as priority of a task.

The data units are either dedicated (associated with a specific task) or common data units. The latter serve for data communication between the different tasks. A common data unit is accessible for any task, if declared so in the task. The access of a particular task may be of read-write or read-only type [A 7].

Core allocation

In very small systems with no background memory the whole code resides in core. In other systems which include bulk memory (disc or drum) the core is divided into two major parts. One part accomodates the executive system (whole or part of it) and may also contain core-resident routines, tasks and common data areas. The other part serves as running area for the bulk-resident tasks. There are several approaches to handling this running core:

a. Only one bulk-resident task may stay in core at a time [A 5, A 9].
b. The running core for bulk-resident tasks is segmented at system generation time and each group of bulk-resident tasks is associated with a particular segment (only one task at a time may occupy a segment [A 3, A 7, A 8]).
c. The whole running core for bulk-resident tasks is dynamically allocated in run-time [A 1, A 2].

Bulk-resident tasks due to run for any reason need to be first transferred to core. They are placed on a core waiting queue (thread) and given access to core in the order of their priority. With respect to the bulk-resident task just staying in core, several solutions exist:

a. The newly activated task, even if of higher priority, has to wait until the task just staying in the core running area (or its assigned segment) is completed [A 3].
b. Some previously specified tasks are interrupted and swapped by higher priority tasks [A 5, A 7, A 8].
c. Higher priority tasks always interrupt and replace lower priority ones [A 2].

CPU allocation

Tasks which have been activated and stay in core (either as core-resident tasks or following a bulk-to-core transfer) compete for the use of the central processor unit. They are placed by the executive on the CPU waiting queue and are serviced in the order of their priority.

Whenever the executive starts operating, it takes over the central processor by interrupting the task just running. The outcome of the executive action as to the use of the central processor may be:

a. Control is unconditionally returned to the interrupted task following a short executive computation (perhaps execution of some routines [A 7,A 8]).
b. A priority-based selection is made from the CPU waiting queue to choose the task that will be allowed to run (this may and may not be the one just interrupted).
c. An unconditional transfer is done to another task.

If more tasks have the same priority, they are serviced

- either first-in first-out [A 3]
- or in pure time-sharing [A 1].

A special feature: under "crisis-time activation", the priority of the task is automatically increased if it is not executed within a specific time [A 1].

The executive investigates the CPU queue to make a selection following

- an external event (process or operator interrupt)
- an internal event (I/O or bulk-memory interrupt)
- a timer interrupt
- an executive call
- completion, termination or suspension of the running task
- lapse of a given time [A 2].

(In each particular system, only different parts of this list are incorporated.)

Following a lasting interruption of a task (type b. or c. above), return to the interrupted task may happen

- directily from the interrupting task (since the interrupting task can also be interrupted, this is a chained recursive organization of tasks) [A 9, A 10] ;

- through the CPU waiting queue according to priorities (this is an independent organization of tasks) [A 2, A 3, A 4, A 5, A 7] .

In order to ensure return, register contents are saved for the interrupted task.

Peripheral handling

Input-output operations are handled by the executive system through specific calls from the requesting tasks. Core-to-bulk and bulk-to-core transfers in course of the execution of tasks are treated in a similar way.

Requests for peripheral operations are placed on the waiting queue of the respective peripheral device. The method of sequencing and servicing the requests is different in different systems:

a. The requests are sequenced first-in first-out [A 1] .
b. There are two groups of requests, normal and priority, the priority requests preceding the normal ones (within a group: first-in first-out) [A 2, A 4] .
c. The requests carry the priority of the requesting task [A 3, A 5].
d. The requests are assigned priority by the requesting task [A 7] .

Some peripheral operations (some outputs), once requested, do not require return to the initiating task. Those which do are, in most systems, handled in two different ways:

a. After issuing a request, the task continues its execution. The completion of the peripheral operation is signalled as an internal event and is serviced (buffered) by a routine without affecting the scheduling of the tasks.
b. After issuing a request, the task suspends its execution. The completion of the peripheral operation is signalled and, in addition to being serviced by a routine, causes release of the initiating task. It then returns to execution through the CPU waiting queue.

Task states

Summarizing the foregoing, we draw up a simplified scheme of task states (Fig. 3.) . A task is always in one of the following states:

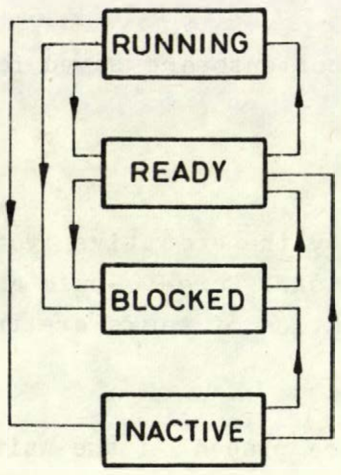


Fig. 3. Task states.

- Running
- Ready
- Blocked
- Inactive

The states Running and Inactive are self-explanatory. In state Ready are the tasks waiting on the CPU queue. In state Blocked are the tasks waiting on the core or some peripheral queue or being suspended (by themselves or the operator) until an external event, a specific time or synchronization (actually, this state comprises several sub-states).

The possible state-transitions are:

Inactive →	Blocked:	an inactive bulk-resident task is activated;
Inactive →	Ready:	an inactive task residing or staying in core is activated;
Blocked →	Ready:	the blocking condition is lifted (core found, peripheral operation completed, event happened, time elapsed, synchronization done);
Ready →	Running:	the task is of the highest priority among the ready tasks;
Running →	Ready:	the task is interrupted by a higher priority one;
Running →	Blocked:	the task is suspended waiting for the completion of a peripheral operation, the occurrence of an external event, specific time or syrchronization with another task;
Ready →	Blocked:	a ready bulk-resident task looses its running core;
Running →	Inactive:	the task is completed or terminated.

Note that tasking is discussed here as implemented in most existing systems. Some new ideas will be introduced in connection with high-level languages (see Section 2.2).

Event handling

Handling of external events (interrupts) is similar to that of internal events. If an event occurs, the executive takes over and locates the event. Then the response to an external event is either or both of the following actions:

a. An interrupt service routine is executed (without affecting the schedule of tasks).
b. A task is activated, generally by being placed on the core or CPU queue in accordance with its priority, or exceptionally by direct transfer of the control of the CPU [A 9] (this activation may also be organized as an interrupt service routine).

Interrupt service routines are associated with events through tables and may be microprogrammed [A 3]. In most systems, interrupt service routines possess the highest priority with no priority sequencing among themselves. In some cases [A 9], they are arranged into different levels of priority and serviced accordingly.

External events are signalled to the system through special hardware facilities like "interrupt lines" [A 1], "event flags" [A 7] or "interrupt status words" [A 9].

Time-scheduling

Tasks that need to be executed at a specific instant or after a certain delay or at certain intervals of time, are placed on a time queue accordingly. The time queue is updated automatically at frequent times. Whenever a time-scheduled task becomes due, an internal event (interrupt) is signalled and the task is placed on the CPU (or core) queue. Its priority is pre-specified by the user.

Executive calls

Executive calls (commands, etc.) serve for the communication of tasks with the executive. They are used to

- activate, time-schedule, synchronize, suspend or terminate a task
- assign or change priority
- request peripheral operations (including bulk-transfer and file-operations)
- obtain information of task states
- obtain information of time.

Executive calls are serviced by special routines.

2.2 High-level general-purpose process-control languages

In this section, the high-level general-purpose process control languages will be discussed. These languages are especially meant for process control (or, generally, real-time) applications but are general-purpose in the sense that they are not oriented at any particular machine or application area within process control.

General-purpose process-control languages are principially procedural languages. This means that most of their statements are executable (describe operations), and the sequence of execution of the operations is primarily determined by the order of these statements. In addition, these languages include some non-sequential specification-type statements as well.

Process control applications possess two basic characteristics that programming languages should comply with:

a. Executive operations like tasking and I/O must be directly programmable.

b. Run-time efficiency of the programs (in terms of CPU time and core-space) is crucial.

High-level general purpose process-control languages are developed

- either by taking a general algorithmic language (like ALGOL, FORTRAN or PL/1), adding some features for executive operations and (perhaps) omitting some others and imposing certain restrictions in order to improve run-time efficiency,
- or by defining a new language.

With the proliferation of languages and unification of language-principles the difference between the two approaches is diminishing.

A considerable number of high-level general-purpose process-control (or real-time) languages have been published in the recent years. Some of them are real-time extensions of FORTRAN [B 3,B 4,B 5,B 6, B 7] while the others are based on other general algorithmic languages or are more or less new ones [B 8,B 9,B 10,B 11,B 12,B 13, B 14,B 15,B 16,B 18,B 19,B 20,B 22]. Most real-time Fortrans are implemented on a particular machine. The proliferation and acceptance of the other languages ranges from valuable academic exercises to relatively widely used national standards.

We are trying to avoid any classification or evaluation of the referenced languages and will restrict ourselves only to showing their basic characteristics. "Purdue Fortran" will be first discussed as a synthesis of several Fortran extensions. The real-time properties of some non-Fortran-type languages will also be treated through the example of a few selected languages.

Purdue Fortran [B 2]

"Purdue Fortran" has been developed by the Fortran Committee of the Purdue Workshop on Standardization of Industrial Programming Languages, to unify the different process control extensions to Fortran. Part of the proposed language extension has already been adopted as an ISA standard while the rest is being considered.

The language extension takes ANSI Standard Fortran (X3.9-1966) as a basis and consists of a set of standard procedures. These realize different actions which are generally needed in a computer process control system but are not included in the Standard Fortran. The procedures are grupped as follows:

- tasking
- process I/O
- file-hangling
- day and time information
- bit-string manipulations
- bit manipulations.

The full list of real-time procedures (tasking, process I/O and day-and-time) is given in Table 2.

Table 2.

Tasking procedures

1. START (i,j,k,m)	- start a task after a specified time delay
2. TRNON (i,j,m)	- start a task at a specified time
3. WAIT (j,k,m)	- delay continuation of a task for a given time
4. HOLD (i,m)	- suspend continuation of a task
5. RELSE (i,m)	- release a task from suspended state
6. EXIT	- terminate a task (self)
7. ABORT (i,m)	- terminate a task (other)
8. LINK (i,m)	- segment a task
9. EST (i,m)	- establish core-residence
10. UNEST (i,m)	- cancel core-residence
11. CHNGE (i,j,m)	- change task priority
12. STTSK (i,j,m)	- interrogate task status
13. CON (i,j,m)	- connect a task to an event
14. UNCON (i,j,m)	- eliminate an event connection
15. FREZE (i,j,m)	- disable a connected event
16. THAW (i,j,m)	- enable a connected cvent

The formal parameters are:

i - name of an integer array that specifies the task (for 8, 9 and 10: program unit) concerned;

j - for 1, 2 and 3: the length of time (direct of referenced); for 11: the assigned priority; for 12: name of an array into which the information will be placed; for 13 through 16: name of (or reference to) the event.

k - reference to the units of time;

m - indicator of the disposition of the request.

Process I/O procedures

1. AISQ (i,j,k,m)	- sequential analog inputs
2. AIRD (i,j,k,m)	- random analog inputs
3. AO (i,j,k,m)	- analog output
4. DI (i,j,k,m)	- digital input
5. DOM (i,j,k,n,m)	- duration-controlled digital output
6. DOL (i,j,k1,k2,m)	- latching digital output

The formal parameters are:

i - number of analog points or digital words, resp.;

j - reference to the acquisition (or transmittion, resp.) and conversion information;

k - name of the array where the information will be placed or taken from, resp.;

n - duration

m - indicator of the disposition of the request.

All process I/O procedures are available in two variants, one causing suspension of the calling task and the other not (e.g. AISQW and AISQ, resp.).

Day and time information procedures

1. TIME (j,m)	- time of day
2. DATE (j,m)	- calendar date

The formal parameters are:

i - name of the array where the information will be placed;

m - indicator of the disposition of the request.

end of Table 2.

Other languages

Now the characteristics of some non-Fortran-type high-level general-purpose process-control languages will be described. As examples, four languages will be taken which are currently being investigated by the European Group of the Long-Term Procedural Language Committee of the Purdue Workshop. The languages are CORAL 66 [B 15], RTL/2 [B 20,B 21], PEARL [B 16,B 17] and PROCOL [B 22,B 23].

For a class of languages, like CORAL 66 and RTL/2, run-time efficiency has been the primary objective. These languages exhibit a straightforward structure and contain no explicit real-time features. Real-time operations like tasking and I/O are implemented by machine dependent procedures and macros. Assembly (or machine) code sequences may be inserted into high-level program texts and also macro-instructions defined and used throughout the program. Note that both CORAL 66 and RTL/2 have been implemented on several machines and are used relatively widely.

In more sophisticated languages like PEARL and PROCOL, in addition to the more or less complete arithmetic features of the modern general languages, special language-level facilities are available for real-time operations. These real-time facilities fall in the following groups:

- system description
- tasking
- synchronization
- process I/O.

System description in these languages in necessitated by the fact that they deal with events (interrupts) and external variables (process or consol points) through symbolic names. Symbolic names are linked with the corresponding physical points at system generation time. For events, a physical point may be a single hardware interrupt, a group of such interrupts or a "software interrupt" (executive operation); their specification is dependent on the hardware system. For external variables, the type and number of the peripheral equipment and the connection point is to be specified (in PEARL, also the complete data-path). Further, system description in PEARL also comprises specification of the computer, including type, features of the CPU, core size and channels.

Tasking will be discussed, slightly simplified, along the line of PEARL (tasking facilities in PROCOL may be considered a subset of those in PEARL). The particular areas to be treated are

- task generation
- task activation
- other task-operations
- scheduling
- event-handling.

Tasks are generated statically or dynamically. Static task generation means that task-names are introduced and associated with the code of the task once for ever. Under dynamic generation, first only the name of the task is declared and the code is associated with the task upon activation.

Activation of a statically generated task is described by the statement

[schedule] ACTIVATE task-identifyer [WITH PRIORITY priority] ;
That is, activation is done according to a programmable schedule

and priority (for dynamically generated tasks, also the actual code is described here).

Further <u>tasking instructions</u> are:

[schedule] SUSPEND task-identifyer

for suspending an activated task;

[schedule] CONTINUE task-identifyer [WITH PRIORITY priority]

for continuing a suspended task and/or re-assigning priority;

[schedule] DELAY task-identifyer [DURING duration | UNTIL instant]

for delaying a task for a duration or until a time-instant;

[schedule] TERMINATE task-identifyer

for terminating an acitvated task;

[schedule] PREVENT task-identifyer

for cancelling pending schedules of a task.

The general (though slightly simplified) syntax of the "<u>schedule</u>" part of tasking instructions is:

{[empty | ON event] [empty | AFTER duration] | AT instant}
{empty | EVERY duration [empty | UNTIL instant | DURING duration]}

This syntax is self-explanatory. "Schedule" makes it possible

- to attach a tasking operation to a (symbolic) event;
- to have a tasking operation performed at a certain instant of time or after a specific delay;
- to have a tasking operation performed repeatedly with a specific frequency, either leaving the end of the sequence open or limiting it by setting, for the last performance of the operation, an instant or a duration (from the first operation);
- to prescribe any meaningful combination of the above, e.g.

ON event AFTER duration EVERY duration DURING duration;
(the 20 possible combinations are shown in Table 3.).

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
ON		x		x			x		x			x		x			x		x	
AFTER			x	x				x	x				x	x				x	x	
AT					x					x					x					x
EVERY						x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
UNTIL											x	x	x	x	x					
DURING																x	x	x	x	x

Table 3.

In addition to attaching tasking operations to events (as shown above), any other statement may be attached as well using the RESPONSE statement:

RESPONSE event : unlabeled statement.

Further, special instructions are available to ENABLE and DISABLE events and to generate software-type interrupts ("signals").

To synchronize tasks and to control usage of common resources by several tasks, semaphores (special integer variables) are used in both PEARL and PROCOL. Semaphores are accessible only for the special instructions REQUEST and RELEASE. A REQUEST operation decreases the value of the semaphore variable by one, should the result be non-negative; otherwise the task containing the REQUEST operation is suspended. A RELEASE operation increases the value of the semaphore by one, clearing the way for the highest priority request among the eventual pending ones to be serviced. Semaphores may be used, for example,

- to synchronize two tasks, that is, to ensure that a task does not proceed beyond a given point (instruction) before another task performes a certain operation;
- to block access by the other tasks to a common data area while one task is exclusively using it;
- to indicate whether or not there is free space in a limited-length buffer attached to some equipment jointly used by several tasks.

The way process input-output is handled is slightly different in the discussed two languages.

In PEARL, there is a special statement for process I/O, having the form

MOVE source TO sink.

In case of an input, "source" is the symbolic name of the communication register of a device and "sink" is the name of a memory location; in case of an output, vica versa. The MOVE statement does not imply any transformation of data. If such a transformation (conversion, coding, decoding or calibration) is necessary, a GAUGE option is attached to the MOVE statement. This contains the call of a previously declared procedure which, with the appropiate actual parameters, performs the required transformation.

In PROCOL, there are separate INPUT and OUTPUT statements. They include, in addition to the symbolic designation of the data-source and sink, reference to a formatting scheme. Formatting for an input consist of

- feasibility checking,
- filtering,
- conversion,
- logical checking.

For an output, formatting includes

- filtering (eliminating drastic changes),
- conversion,
- logical checking.

For each formatting item, the user may choose either the standard treatment (with specific parameters), or introduce new procedures of his own.

2.3 Application packages - problem-oriented languages

All major manufacturers provide with their process control systems several application packages. These packages are pre-written computer programs that

- operate in close connection with (and utilize several internal facilities of) the real-time executive system;
- take care of a particular functional area common in a class of process control applications.

When dealing with application packages from a user's point of view, one has to concentrate on two basic aspects:

a. What is the particular functional area it is intended for and, within this, what are the services it provides.
b. What is the programmer's interface to the package, that is, how to program the software-hardware system for a specific task. (In most cases, this interface is a special problem-oriented language.)

Functional areas

The major functional areas encountered in many process control systems are as follows:

- data acquisition and conditioning,
- direct digital control,

- supervisory control,
- sequence control,
- optimization.

Note that the border-lines between the separate packages of a particular vendor are not quite definit: data acquisition and processing is included in most control packages, also some higher level control packages contain elements of the lower ones and there are possibilities for inter-package referencing.

The systems to be dealt with here are general process control packages. Apart from these, also several special packages have been developed to meet the needs of particular industries (e.g. steam power generation [C 11]).

Data acquisition and conditioning packages [C 1,C 2,C 5,C 6,C 7,C 9] include, as basic steps, scanning, filtering, conversion and limit checking.

a. Scanning is acquisition of rough process data through the respective input devices. The user selects the appropriate scanning rate for each variable. A first limit-checking is performed on these data to detect faults of the measuring system.
b. Rough measurements are digitally filtered to reduce noise-effects. The user may choose between first and second order digital filters and specify his filter-parameters.
c. Conversion of the measured data is generally performed in two steps. First the non-linearities of the sensor are taken care of. Typical non-linear sensors are thermo-couples and flow-meters. In the latter case, temperature and pressure are also taken into account as correcting quantities. In the second step, the linearized (or linear) measurements are converted into the appropriate engineering units.
d. For limit-checking, most systems allow two upper and two lower limits. The user may prescribe different response actions to the violation of the inner and outer limits. Further, user defined dead-bands may be attached to each limit value to filter "return to normal" actions (messages). Also limit-checking for the rate-of-change of variables is available.

Direct digital control packages [C 1,C 2,C 5,C 6,C 7] are primarily based on the digital implementation of the conventional three-term (PID) control algorithm. The user may choose sub-algorithms (P, I, PI) and specify his control coefficients.

The input to the algorithm is either the control error or its signed square (e|e|). In some systems, the user may indicate if he wishes to have setpoint-changes neglected in the differential term. Also available is adaptive tuning with changing coefficients (or neglected terms) upon high error or significant setpoint changes [C 1].

The output is either position or incremental type. Upper and lower limits are specified for the absolute (position) value of the output and maximum-per-step for its increments. If a calculated output leads to violation of any of these limits, it will be reduced accordingly [C 7]. Also, a dead-band for the output increments may be defined to make control operation more quiet [C 3]. In some systems, incremental control is combined with position feed-back to base the calculation of increments and checking for position limits on real position instead of recursive computations [C 5].

A simple ratio-control algorithm is also available in most DDC packages. In addition, some systems offer special compensator algorithms like pure time-lag, sum of multiple inputs and lead-lag [C 1].

Supervisory control packages [C 3,C 9] are meant for computing setpoint values or changes for analog or DDC controllers in continuous processes. Supervisory control is generally done in a steady-state or quasy steady-state manner. There are two ways to describe the basic computation:

a. Using a standard adjustement equation [C 9]. This equation provides the necessary change of a manipulated variable, based upon the actual deviation of the controlled (feed-back) or some measured (feed-forward) variable. There is a possibility to consider deviations of three further variables. Up to four adjustement equations with common variables may be handled simultaneously.
b. Using special simplified procedural languages involved in the package [C 3,C 9].

Additional facilities [C 3,C 9] include:

- limit-checking on the inputs of the algorithm;
- minimum output deviation (dead-band) below which no control action will be performed;
- absolute or incremental limits for the output;
- special actions or programs to obtain initial values for the control calculations.

As far as timing of the control action is concerned, the user may specify [C 9]:

- a minimum time between two adjustements,
- a set-point movement rate,
- stair-function of up to four steps, expressed as fractions of the calculated change of set-point versus fractions of a specified delay-time.

<u>Sequence control packages</u> [C 7,C 12] serve for programming batch-processes or start-up/shut-down operations in continuous processes. Their basic feature is the evaluation of logic conditions, involving functions like AND, OR, EXOR, INVERT. Inputs to the logic equations are

- ON/OFF type status informations from the process or consol,
- logic results of process variable comparisons (to limit or each other),
- timing conditions (in logic form).

In addition to the special sequencing facilities, these packages include some reduced data acquisition and control features as well.

<u>An optimization package</u> [C 4] offers linearized solution to the general non-linear optimization problem. The objective function is either cost or profit. A user-written model of the system to be optimized provides, for a given input situation, either the dependent variables or the partial derivatives of the objective function. The standard program finds the optimum by the repeated application of the Simplex algorithm to locally linearized regions of the model. Hard (impassable) and soft (penalized) limits for all variables are taken into account as well as limits regarding the permissible change of system variables in each optimizing step.

Problem-oriented languages

To make an application package operable, it has to be programmed for the given job. The objective of this programming is

- to fill up the data-files of the package, that is, to inform the software of the actual numerical parameters;
- to specify the way of execution of the package, that is, to include/omit and link different program blocks;
- to describe non-standard operations.

Fitting packages to the particular job is implemented by means of problem oriented languages which are provided as part of the package. Those languages meant for file-building and program-linking are of specification type: their statements describe specifications for a previously programmed sequence of operations instead of the operations themselves. On the other hand, languages for describing non-standard operations are of procedural type, similar in this sense to the general-purpose process-control languages.

Looking at the formal aspects of these languages, they may be

- strictly formatted languages,
- "fill in the blanks" systems,
- assembly-like languages,
- high-level (English-like) languages,
- conversational systems.

Note that some packages include two languages: one for specification and another for describing non-standard operations.

The strictly formatted languages are meant for skilled programmers. They use low-level (numeric and alphanumeric) symbols. There are strict rules to govern the length and order of the symbols, the use of delimiters and the card-layout. Such a language was developed as means of specification for the OPO optimization package [C 4].

The "fill in the blanks" technique has been devised for unskilled programmers. Basically this is also a strictly formatted system, but the programmer need not care about formatting. He just has to fill in pre-printed forms where the sequence and format of the answers is fixed. Cards are then punched mechanically on the basis of the forms.

"Fill in the blanks" technique is used in the supervisory control packages BICEPS [C 3] and PROSPRO [C 9] for file-building and program linking. The forms contain blanks

- for the different numerical parameters of data acquisition and conditioning (like filter and conversion coefficients etc.) and control (like dead-band or time-delay);
- for the numerical codes of execution specifications (like type of filter and conversion equation, absolute or incremental output etc.);
- for references to programs describing non-standard operations.

In PROSPRO, the "fill in the blanks" technique is extended to some non-standard arithmetic operations. This is achieved by introducing a "general equation" and an "adjustement equation". The user may select his particular equation within the given scheme by specifying his own coefficients (that may also be zero). This again is performed by filling in blanks in some special forms.

Instructions in an <u>assembly-like problem oriented language</u> consist of a mnemonic operation code and up to two operands. An assembly-like procedural language is provided in the PROSPRO package [C 9] for programming non-standard operations ("general action"). The available instructions are

- arithmetic operations (variable equals variable/constant/equation, variable times/plus constant/variable, variable minus/devided by variable);
- comparison (variable to variable/constant);
- conditional branch (result minus/zero/plus, compare low/equal/high);
- unconditional branch;
- time operations (save real-time and calculate time difference);
- adjustement (feed-back or feed-forward, using the standard adjustement equation);
- program control operations (return to normal processing, etc.).

<u>High-level problem-oriented languages</u> have free-format English-like statements, similar to those of FORTRAN and other well-known languages.

A characteristic example of the use of high-level languages in programming process control packages is the DACS-AUTRAN system [C 1]. This comprises two free-format English-like languages, one for building files and linking program-parts of DACS (Data Acquisition System) and another for programming non-standard supervisory control.

The AUTRAN specification language has the following sorts of statements:

- group-specifications (scan and output groups);
- input-point specifications (digital, pseudo-digital, analog, external);
- output-point specifications (register, momentary, latching, control device, analog controller, control valve);
- control operation specifications (controller, compensator, calculator, switch);
- input processing (conversion, filtering, limit-checking);
- output processing (C-M relay, status sensor, output, output limits);
- control processing (parameters for control operations);
- alarm response specification;
- timing (cycling) specification.

The supervisory control (procedural) language involves a version of FORTRAN as a subset. Additional statements are:

- I/O variable list (with reference to DACS files);
- output statements (contact output, position and incremental analog output);
- semi-output statements (setting DACS variables);
- semi-input statements (getting value of DACS variables);
- equipment control statements (activate/deactivate points);
- tasking statements;
- logging statement.

Another high-level problem-oriented language is the procedural BPL (Biceps Programming Language) [C 3], added to the BICEPS supervisory control package to program non-standard operations. BPL is a very simple language consisting only of a few fundamental features. These are:

- constant and variable declarations,
- FETCH and STORE statement,
- basic arithmetic operations,
- conditional jump (with EQ, LT, LE, GT, GE as comparison operators),
- simple standard functions (like ABS),
- printing.

Conversational systems assume the least skill of the programmer. Programming is performed through an alphanumeric I/O device (typewriter) or display . After the programmer has indicated his intention to communicate, the system asks a set of questions. The programmer has to type in his answers

- either using an assembler-like language [C 6,C 7],
- or by making the right selection from the choice of answers offered, together with the question, by the system [C 2].

The conversational technique can only be applied for specification purposes. This approach has been taken in the OMNIBUS-DDC [C 2] and PM/C [C 5] packages as well as in the CONRAD [C 7] and CONSUL [C 6] systems.

Though very convenient, the conversational technique is too slow to handle large amounts of information. Therefore, most systems provide an optional punched tape or card input with an assembler-like specification language for system initialization, while conversational programming is mainly reserved for additions and modifications [C 5,C 7].

2.4 Man-Machine Communication Software

We noted in Table 1. that the existence of appropiate communications (man-machine and machine-plant) is one of the characteristic properties of industrial control systems. To accomplish these communications, we need an appropriate hardware (devices of the controlled plant, instruments, a control computer, terminals, etc.) and a good software. Man-machine communications form an important part of the activities mentioned in Section 1.3. and follow closely after system analysis. These activities are

- programming (flowcharting, coding, debugging)
- testing (static and dynamic)
- verification on the real plant (tuning and long-term operation).

Man-machine interface (serving the process operator, control engineer or programmer) should obey the following rules:

a. The communication system should be able to display numerical values (variables and technical parameters), messages, to register trends (also multiparameter) and to enter new plant parameters or values.

b. Each parameter to be entered into a control system must possess a functional specification (scan, alarm, control, log, etc.).
c. It should be possible to make visible all input data before entering them into the system.
d. All input data, changing parameters or conditioning operations, must be registered automatically. This registration should reflect contents of the respective memory address.
e. The possibility of the registration optionally of all the alphanumeric requests which are displayed within the communication system, should also exist.

Note that items c. and d. simultaneously perform checks for frequently occuring human errors.

Software for programming and testing will be discussed later in Section 3. Here let us pay out attention to the human activity within the industrial control system. This activity will be exercised within the user programs contained mostly in modules 1 and 2 (see Fig. 1.). These user programs are executed by:

a. basic software (real-time executives, assemblers, compilers, program development utilities e.g. editors etc., re-entrant routines, data-base manipulation software);
b. communication software (peripheral device control packages and drivers, terminal communication packages, software for multiplexors, buffering and core mapping for multiprogramming);
c. special programming systems for non-programmers (form or dialog based).

Man-machine communication in basic software

Communication between man and basic software is enabled by the use of special syntactical units of programming languages. For instance, a flexible macroinstruction language was chosen to accomplish this with DIRECTOR [A 1]. In addition to a language like this, the programmer's console is generally available with an access to a keyboard and a printer. These facilities mainly enable

a. to introduce new programs and names of entities (for data file handling, etc.),
b. to change the specification of programs (necessary core/bulk store, identification of source text, control of peripheral units, etc.),
c. to cause program activations,

d. to provide the system with operational data,

e. to initialize calendar and real-time clocks.

Let us present examples of some macroinstructions:

: TIME

This macroinstruction [A 4] causes printing out the current "real-time" of day.

$ ENTER AT CRISIS : PROG73;,,2,30,15,,:,4,,1,10,3,:,,,2,5,7,

This ENTER AT CRISIS macroinstruction [A 1] causes the activation of PROG73 at 30 minutes and 15 seconds after 2 o'clock. After 1 more minute and 10 seconds its priority changes to 3. Numer 4 (the first digit after the colon) indicates that there is a subsequent CRISIS time. Therefore after 2 more minutes and 5 seconds the priority of this program changes to 7. The simple syntax of this language is evident from the above example.

Communication software

In process control, especially the equipment listed below is used for communication:

- conversational typewriter (for programmers)
- conversational typewriter (for operators and technologists)
- output typewriter (or printer) for passive communication (alarm, log, various messages, trend records)
- pen recorders for operators
- pen recorders (trend recording)
- displays
- light panels
- punched tape and card I/O units.

Here the obvious proviso is that all these units are connected with the central processor unit through a unified interface. Then also software can be built in a unified way in order that the following three basic facilities can be provided:

a. Routines to handle and react to real-time events (like button pressings, light-pen interrupts, etc.) initiated by man through peripherals or terminals/modems.

b. Routines to build and manipulate data buffers and files (display file, printer file, card file, etc.) .

c. Routines to organize and present information for process-man communication (i.e., input-output routines for teletypewriters/ printers; routines to create standard layout plotting patterns necessary for graphical aids, e.g., points, lines, circles, characters, etc.).

Communication data structures realized by means of various I/O units can, of course, be diversified. With display interface for instance the following three properties of data structure are required [F 1]:

- It must represent the display sequence correctly, i.e., it must imply the order in which the patterns are to be displayed.
- It must imply the number of words in the display file corresponding to each pattern so that editing may be performed.
- It must have a way of associating a "name" with a pattern. Names may be assigned either by the user or by the system. These names are used for all communications about the pattern between the user and the system.

From another aspect, most displayable files are [F 2]:

- maps movable within a page;
- one-page displays which can be updated but not moved, magnified or rotated;
- scrolls (a scroll is a tabular list which may be too large for a screen and can therefore be scrolled backwards or forwards).

Software for non-programmers

Such software has been developed, for example, for continuous process control and is built of separate pre-programmed algorithms for industrial data manipulation, general control actions, operator logs and console displays and process output control. To accomplish man-machine communication, special control panels are used equipped with keyboards (functional and numerical) and simple displays to picture, as a rule, loop identifications, current values, new values and various visible signals. Let us present the facilities as provided by one of these software systems, CONSUL B (a subset of the CONSUL system [F 2], also cf. Section 2.3.):

a. Basic modular facilities
   - on-line assembly, modification and removal of control loops in the control system;
   - opening and closing cascade switches;
   - monitoring of process and control variables and checking parameters;
   - interface with auto/manual stations.

b. Optional facilities

- measured value logging for all the loops in the system;
- trend logging for selected loops;
- alarm logging for all loops in alarm state;
- loop logging of all loop variables for adjusting the loop;
- chart recording of selected measured values, set points and valve positions;
- background-mode running of standard compiler and editing programs and of user written background programs.

From the point of view of the future development of process control programming, we may say those systems based upon macroinstruction languages are closed. Their widespread use has proved their vitality and justified their place in programming.

3. SERVICING OF PROCESS CONTROL SOFTWARE

Once we have finished the analysis of the algorithm of an industrial control system from the programming point of view, we can write the program in a chosen language. Then there are two main activities awaiting us around the computer, namely

- removal of syntactical and semantical errors from the program (debugging);
- verification that the program actually realizes the algorithm (testing).

Of course, these two activities are very often interleaving. Formally, testing starts when debugging is "finished". However, the errors found during testing lead to program modifications which, having been performed, must go through "a new" debugging. Such a successively approximating activity is especially exacting in program development with real-time objectives. Unfortunately, there is no exact rule as to when to stop this and pass on to the phase of verifying the algorithm on a real plant. Here it is necessary to combine computer science results on one hand with general experience on the other hand. The around-the-computer preparation of the "right" program should be associated with an automatic creation of documentation on the program (including all the program modifications). For this reason, it is not surprising that the right, high-quality outfit of service programs brings a fair economic benefit during realization of control algorithms. However, relatively small attention has been paid to these questions in literature.

Let us mention now the services performed by the individual utility programs as they are successively met by the user:

- program preparation (PP)
- debugging (DB)
- testing (TS)
- tuning (TN).

A simplified diagram of links between these activities is presented in Fig. 4., together with a list of the subjects which these activities work with. The PP, DB, TS and TN acitivities are of course partially realized by means of the computer. For this reason, servicing software has its specific structure (see Fig. 5.). If S(PP) stands for software for PP and S(DB), S(TS), S(TN) alike, the following relation is valid

$$S(PP) \subset S(DB) \subset S(TS) \subset S(TN).$$

Let us now try to characterize, briefly and gradually, the traits of these parts of servicing software.

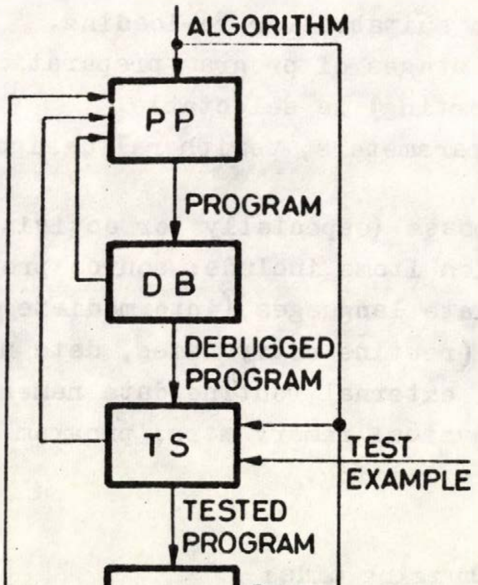


Fig. 4. SERVICING SOFTWARE FOR PROCESS CONTROL

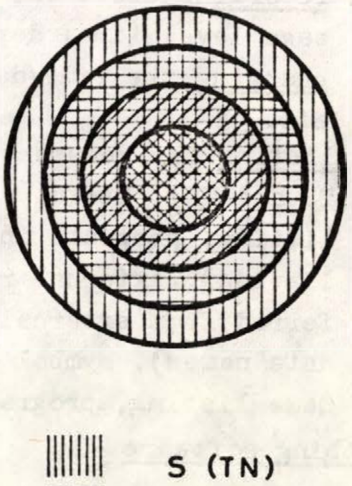


Fig. 5. STRUCTURE OF SERVICE SOFTWARE FOR PROCESS CONTROL

Program preparation software

During compilation and assembly, syntactical and semantical analysis of the source program text is important. Here it is convenient that both the usual messages about the errors found and the so-called directives [F 2], i.e., control commands about the service activity of translators, be contained in our programming languages. The individual service routines then provide especially the following services:

a. editing [F 2] and modification [A 5] of programs at any language level used. Editing programs developed in any programming language and for on-line or off-line use, can update the source text with the corrections required.
b. loading data or absolute or relative programs in binary format from any input device/file into core/bulk memory. During loading, error checking and global label linkage is performed.
c. comparison (for checking purposes) of information read from any input device or a file with the contents of core/bulk memory. The differences found are indicated for instance on the programmer's console/teletypewriter.
d. dump of selected core area or bulk memory on any output device. The information thus dumped is suitable for re-loading.
e. reporting about the different stages of program preparation. Message level (i.e. depth of reporting) is selectable.
f. changing the time/date, task parameters, peripheral designation, etc.
g. listing for documentation purposes (especially for activities DB, TS and TN). Useful documentation items include: source program listing, programs in intermediate languages (intermediate language listing), external references (routine entry names, data names referred from external programs, external routine/data names, common data names), symbol tables, physical memory maps, program unit name listing, program structure listing.

Degubbing software

We can distinguish the following debugging aids:

a. tracing which is provided for each programming language (similarly as editing). The interpretative software package usually enables the programmer to obtain various printout formats [F 2]. Tracing at the machine level language includes
   - instruction-only printout;
   - fully-executed-instruction printout including the resultant contents of all the registers affected by the instruction;
   - printout in either mode on program branching only;

- printout on tagged (pre-identified) instructions only;
- omitting n executions of some tagged instruction.

Similarly, formats are provided for higher level programming languages. The formats are obviously based on their syntactical elements (macro, statement).

b. examination of program and data storage contents. A selected sequence of core memory locations are printed out in a selectable format (program, octal number, integer, character, etc.).
c. changing the contents or type of numbers (FIXED → FLOAT, FLOAT → FIXED).
d. searching uses a breakpoint technique to check a program at selected points. The programmer can stop the program at pre-identified points and then use some of the previous aids.
e. hardware debugging for an integrated check of computer systems by test programs running in foreground.

Observations. 1. The debugging software may be used in foreground or background. 2. "Long-Term Procedure Language" [B] is intended to have syntactical units especially designed for
- setting the conditions for debugging operations,
- executable statements for debugging operations,
- auxiliary listings useful for debugging,
- reporting (error messages, etc.) for debugging.

Testing software

In accordance with the diagram shown in Fig. 4., during testing the algorithm of an industrial control system, a debugged program and a testing example form the subject of testing. The testing example contains some known data (generated by a small routine) and expected answers as well as selected program checking points. The testing example must also contain bases for evaluating time-parameters of tasks. Therefore, there are two stages of testing, namely static and dynamic (cf. Section 1.3.). Both stages of course require their specific software; they involve and utilize also the activities discussed earlier (see Fig. 5.).

a. Static testing software consists particularly of the following elements:
   - routines for generating example data
   - testing procedures for testing task modules, these procedures being able first of all to compare pre-identified results of the tested module with expected answers and then to halt the execution of the module under test if a specified condition

is met.

- core image software [A 1,G 2] provides additional information about the state of the program system at the time when the dump was taken. Thus we can receive a report which is not included in the dump. Among different types of messages are: system status, program (task) status, core map, trace history, peripheral status, bulk status, background (free-time) status, etc.
- query option facility, built into some macroinstruction systems [F 2], permits the programmer to insert in the tested program extra program instructions for monitoring intermediate data values at strategic program points. Subsequent translations can then progressively eliminate option items as knowledge of the correct performance is increased. This query option also permits several alternative sequences of code to be held on the same program file (great program modularization).

b. Dynamic testing software is closely connected with the real-time executive used. This testing software performs primarily the following activities:

- testing the interask cooperation (user programs), this cooperation being controlled under time conditions and priorities in a multiprogramming environment;
- testing the behaviour of the program with respect to time, with the registration of the actual path of execution (e.g., reporting on task's /routine's / subroutine's labels as they are passed).

Tuning software

This software operates mostly through real-time executive calls (cf. Section 2.1.). Some of the software pieces discussed previously in the present Section are also incorporated. Tuning itself can start when the installation testing is finished. By means of this software are realized the flexible modifications of the control algorithm as inferred from the application of this algorithm on a real plant. A choice of the necessary programming facilities to accomplish such a correction then depends on the extent and depth of the required modifications as needed for the industrial control system.

4. STANDARDIZATION

Standardization of industrial programming languages has for long been a desire of many people active in the field. Like in many other areas, standardization would result in a considerable saving of human efforts. The primary benefit of standardized programming techniques consist in transferability of software products from one system to another, but the advantage of having to learn only one language is also significant.

This desire and recognition led to the formation of the Purdue Workshop on Standardization of Industrial Programming Languages in 1969. The Workshop, the far most significant effort towards this direction, has been established with the very ambitious program of producing standard proposals for the different levels of industrial languages within a couple of years. Five committees were formed to start work in the fields of

- glossary
- functional requirements
- problem-oriented languages
- industrial Fortran
- long-term procedural language.

It should be noted that a Technical Committee on Industrial Computer Languages was also formed in Japan. Its three sub-committees (Problem Oriented Languages, Fortran, Long-Term Procedural Languages) maintain close cooperation with the respective Purdue bodies. Also a very active subcommittee of the Long-Term Procedural Language Committee exists in Europe.

To unify the usage of special terms of the field, the Glossary Committee of the Workshop developed a "Dictionary for Industrial Computer Programming" which was published by ISA (Instrument Society of America) in 1972 [D 1]. Now a second edition is being prepared.

The role of the Functional Requirements Committee was to prepare the way for the language committees, that is, to produce functional requirements for industrial computer systems to serve as a basis for the development of standard industrial computer programming languages. This work was completed and the results published in 1971 [D 6,D 7].

The main objective of the Workshop has been developing the proposed language standards. In this respect, however, the outcome is well behind the original expectations. One of the reasons is certainly the voluntary nature of the work: many people active for one period or another, drop their affiliation because of their changing working conditions and interest. The most serious reason, however, is probably the difference between company (and, in some cases, national) interests.

The Problem Oriented Language Committee has, for a long time, been attempting to find its way of operation. After studying functional requirements for and general features of problem oriented languages, they arrived at the intermediate result that these languages, or at least their procedural parts, should be considered as macro-forms of some general-purpose procedural language. Thus a wide class of problem oriented languages could be handled by translating them into the standardized long-term procedural language. A couple of suitable translaters are already available, but the lack of the definition of the object language prevents real progress towards this direction.

Perhaps the overwhelming popularity of Fortran is the reason why an industrial extension of this language proved to be most ripe, among the three levels, for standardization. Indead, the Fortran Committee of the Workshop succeeded in developing a proposal, containing special calls for process I/O, bit string manipulations and some executive functions, that was standardized by ISA in 1972 [D 2]. A second proposal, dealing with Fortran procedures for handling random unformatted files, bit manipulation and date and time information, is just being considered by ISA [D 3]. A third and last one on task management is under final development [D 4]. Note that the first of the above extensions has been standardized also in Japan [D 5], while the two others are being considered.

The Long-Term Procedural Language (LTPL) Committee was formed with the aim of developing a high-level general-purpose process control language that might replace industrial Fortran on the long run. The Committee first decided to base this language on PL/1, a choice later attracting much criticizm. This aspect of the work has since then been shifted to the X3J1.4. committee of ANSI (American National Standard Institute), explicity dealing with the definition of a process control version of PL/1. Meanwhile, the European subcommittee of LTPL has been formed and gained strenth gradually; now most of the LTPL work is being done in this group. They compare and evaluate existing process control languages to find the best mixture recommendable as an international standard. They have also established contacts with the respective committees of ISO (International Standard Organization). Unfortunately, conflicting national interests sometimes hinder also productivity of this group.

Just recently, the Purdue Workshop has been drastically re-organized. It was merged with the Purdue-ISA Computer Control Workshop (covering hardware and system aspects of computer control). Also, it was given an international structure with three regional workshops in North-America, Europe and Asia (Japan) and an international workshop (named International Purdue Workshop on Industrial Computer Systems) integrating the regional ones.

The Purdue Workshop is affiliated with ISA and IFIP. Similar affiliation with IEEE and IFAC is under negotiation.

ACKNOWLEDGEMENT

The authors would like to thank the

Brown, Boveri and Cie Actiengesellschaft
Control Data Corporation
Digital Equipment Corporation
Ferranti Limited
GEC-Elliott Process Automation Ltd.
General Electric Company
Gesellschaft für Kernforschung MbH
Hewlett-Packard Company
International Business Machines Corporation
Societe de Realisations en Informatique et Automatisme

for providing source-material to this survey. They also express their gratutude to their master-institutions, the Institute of Computing and Automation, Hungarian Academy of Sciences (Budapest, Hungary) and the Institute for Industrial Management Automation, INORGA (Prague, Czechoslovakia) for supporting this work. Special thanks to Mrs. E.Fazekas for the careful preparation of the manuscript.

REFERENCES

E 1. Sedlak, J.: Development of Programming Means in the Regions of Production Scheduling, Production Control in Real-Time. Ref.6.8., IFORS 70, Karlovy Vary, Czechoslovakia, Sept. 1970.

E 2. Holt, O.W.: General Purpose Programming Systems. Communications of the ACM, Vol.1.No.5.pp.7-12 (1958)

E 3. De Latourne, J.Y. et Garelly, H.: Tour d'horizon sur la programmation des calculateurs industriels. Automatisme, Tome XVI.No.11.pp.559-570 (1971)

X 1. Pike, H.E.Jr.: Process Control Software. Proc.of the IEEE, 58, (1)pp.(87-97) 1970

A 1. Director, Argus 500 Operating System. Ferranti Ltd.

A 2. RTMOS Real-Time Multiprogramming System for GE-PAC 4010 and 4020 Systems. Summary Manual. General Electric Co., Phoenix, Arizona, 1970. GET-6032A

A 3. RTMOS-30 For GE-PAC 3010/2 Computer Systems. General Electric Co., 1972. GET 6314

A 4. RTOS User Manual. GEC-Elliott Automation Ltd., 1973. SP-UM S 25 238

A 5. Real-Time Executive Software System. Programming and Operating Manual. Hewlett-Packard Co., Palo Alto, Cal., 1971. No. 02005-90001

A 6. Real-Time Executive File Manager. Programming and Operating Manual. Hewlett-Packard Co., Sunnyvale, Cal., 1973. No. 29033-98000

A 7. RSX-11D Concepts and Facilities. Digital Equipment Corp., Maynard, Mass., 1973. DEC-11-OXCDA-A-D

A 8. RSX-11D Programmer's Reference Manual. Digital Equipment Corp., Maynard, Mass., 1973. DEC-11-OXDPA-A-D

A 9. IBM 1800 Functional Characteristics. (IBM System Reference Library, 1800-01.) A 26-5918-5. 1966.

A 10. IBM System/7 Functional Characteristics. IBM Systems, GA34-0003-0, 1970.

B 1. Pike, H.E.Jr.: Procedural Language Development at the Purdue Workshop on the Standardization of Industrial Computer Languages. V.World Congress of IFAC, June 12-17, Paris, 1972. Paper 10.3.

B 2. Minutes Eight Workshop on Standardization of Industrial Computer Languages, Purdue University, October 1972.

B 3. FORTRAN reference manual. GE Process Computer Department. Manual YPG14M, May 1965.

B 4. Diehl, W. and Mensh, M.: Programming Industrial Control Systems in FORTRAN. IFAC-IFIP Symposium on Digital Control of Large Industrial Systems. Toronto, Canada, 1968.

B 5. Roberts, B. C.: FORTRAN IV in a Process Control Environment. IEEE Transactions on Industrial Electronics and Control Instrumentation, 15 (2) pp. 61-63 (1968)

B 6. Hohmeyer, R. E.: CDC 1700 FORTRAN for Process Control. IEEE Transactions on Industrial Electronics and Control Instrumentation, 15 (2) pp. 67-70. (1968)

B 7. Mecklenburgh, J. C. and May, P.A.: PROTRAN, a FORTRAN based computer language for process control. Automatica, 6, pp. 565-579 (1970)

B 8. An Introduction to CONTRAN. Honeywell Inc. Special System Division. Pottstown, Penna. 1965. SSD-20-ICMP 4/65-750

B 9. Schoeffler, J. D., Wilmott, T. and Dedourek, J.: Programming languages for industrial process control. IFAC-IFIP Second International Conference on Digital Computer Application to Process Control, Menton, 1967. Instr. Soc. of America, Pittsburgh, Penna., USA, ed.: W. E. Miller, pp. 371-388.

B 10. BCS Specialist Group: A language for real-time systems. The Computer Bulletin, pp. 202-212 December 1967.

B 11. Processalgol (PROGOL) I. SINTEF, AVD. Reguleringstechnikk, Trondheim, 1968. 68 17E 480 165

B 12. Boulton, P. I. P. and Reid, P. A.: A process control language. IEEE Transaction on Computers, C-18 (11) pp. 1049-1053. (1969)

B 13. Schoeffler, J. D. and Temple, R. H.: A real-time language for industrial process control. Proc. of the IEEE, 58 (1) pp. 98-111 (1970)

B 14. Gertler, J.: High level programming for process control. The Computer Journal, 13 (1) pp. 70-75 (February 1970.)

B 15. Official definition of CORAL 66. Prepared by the Inter-Establishment Computer Applications as a language standard for military programming. Her Majesty's Stationery Office, London, 1970.

B 16. Brandes, J. et al.: PEARL, The concept of a process- and experiment oriented programming language. Elektronische Datenverarbeitung, 12 (10) pp. 429-442 (1970)

B 17. Timmesfeld, K. H. et al.: A proposal for a process- and experiment automation rela-time language. Gesellchaft für Kernforsch. MbH. Karlsruhe, 1973.

B 18. Industrial Programming Language: LAI. CERCI-SESA. In: Minutes Fourth Workshop on Standardization of Industrial Computer Languages. Purdue University, October 1970, pp. 145-153.

B 19. PAS 1, Process Automation Language Description. BBC Actiengesellschaft, Mannheim, October 1971. ZEK-ED and ZPF/L.ED1004 E (872.ol)

B 20. An Introduction to RTL/2. ICI Corp. Laboratory, Reading, England. In: Minutes Eight Workshop on Standardization of Industrial Computer Languages. Purdue University, October 1972, pp. 217-257.

B 21. RTL/2 Language Specification. In: Minutes Eight Workshop on Standardization of Industrial Computer Languages. Purdue Univ., October 1972, pp. 259-320.

B 22. Ritout, M., Bonnard, P. and Hugot, P.: PROCOL: a programming system adapted for process control. V. World Congress of IFAC, June 12-17, Paris, 1972. Paper 10.1.

B 23. Systeme PROCOL T 2000. Notice Technique. STERIA Le Chesnay, France. Ref. 1 162 220/00 39 00

C 1. Control Data 1700 Computer System AUTRAN DACS, Software Reference Manual Version 1.o., Control Data Corporation, La Jolla, Cal., 1971. No.3964450

C 2. GE-DDC Direct Digital Control. Summary Manual. General Electric Co., Phoenix, Arizona, 1969. GET-3558A

C 3. BICEPS Supervisory Control. Summary Manual. General Electric Co., Phoenix, Arizona, 1969. GET-3559A

C 4. OPO On-Line Process Optimization. Summary Manual. General Electric Co., Phoenix, Arizona, 1969. GET 6033

C 5. Process Monitor and Control (PM/C). Users Manual. General Electric Co., Phoenix, Arizona, 1972. GET-6256

C 6. Ferranti Argus Consul. Flexible Software for On-Line Control. Ferranti Automation System Division, Ferranti Ltd., 1968. ASD 19 21

C 7. MARCH Industrial Software for GEC 2050 Computer. GEC-Elliott Automation Ltd., New Parks, Leicester, 1972. A.1001-80, A 1002-200 through 205

C 8. IBM 1800 Process Supervisory Program (PROSPRO/1800)(1800-CC-02X), Users Manual. IBM 1967, 1968. H20-0474-1

C 9. IBM 1800 Process Supervisory Program (PROSPRO/1800)(1800-CC-02X) Language Specification Manual. IBM 1967, 1968. H20-0472-1

C 10. Process Monitor and Control (PM/C) for GE-3010/2 Process Computers. General Electric Co., 1972. GEA-9643

C 11. SEER (Steam Electric Evaluating and Recording) System Manual. General Electric Co., Phoenix, Arizona, 1970. GET-3566-A

C 12. BATCH Sequencing System. Foxboro Co., Foxboro, Mass., 1968. TIM-R-97400A-5-4

F 1. Thornhill, D.E., Brackett, J. W. and Rodriguez, J. E.: A sample interactive graphics program. Course Notes for Two-Day Seminar on Programming Techniques for Interactive Computer Graphics, NEL, Glaslow, England, 1968.

F 2. Mc Laughlin, M. E.: Argus software for process control, message switching and general real-time applications. Ref.4.9., IFORS 70, Karlovy Vary, Czechoslovakia, Sept. 1970.

G 1. Gruenberger, F.: Program testing and validating. DATAMATION, 14, No.7., pp. 39-47 (1968)

G 2. DEACON/CIA User's Manual, General Electric Co., GET-6257, 1973.

D 1. Glossary Committee, Purdue Workshop on Standardization of Industrial Computer Languages: Dictionary of Industrial Digital Computer Terminology, Instrument Society of America, Pittsburgh, Pennsylvania, 1972.

D 2. Anon., "Industrial Computer System FORTRAN Procedures for Executive Functions and Process Input-Output", Standard ISA-S61.1,

D 3. Anon., "Industrial Computer System FORTRAN for Handling Random Unformatted Files, Bit Manipulation, and Data and Time Information", Proposed Standard ISA-S61.2, Instrument Society of America, Pittsburgh, Pennsylvania, 1972.

D 4. Anon., "Working Paper, Industrial Computer FORTRAN Procedures for Task Management", Proposed Standard ISA-S61.3, Purdue Laboratory for Applied Industrial Control, Purdue University, West Lafayette, Indiana, 1972.

D 5. FORTRAN Subprograms for Industrial Computer System. JEIDA-20-1973. In: Minutes Ninth Workshop on Standardization of Industrial Computer Languages, Purdue University, West Lafayette, Indiana,1973.

D 6. Functional Requirements for Industrial Computer Systems. In: Minutes Fifth Workshop on Standardization of Industrial Computer Languages, Purdue University, West Lafayette, Indiana, 1971.

D 7. Curtis, R. L.: Functional Requirements for Industrial Computer Systems. Instrumentation Technology, 18, No.11, pp. 47-50 (November 1971).