CAROLINE I. A new DDC-system for biotechnical process control

K. Schneider

Institute of Plant Biology, University of Zürich, Zollikerstr. 107, CH-8008 Zürich (Switzerland)

Summary. CAROLINE I represents a DDC software package utilizing to full advantage the power of modern microcomputer systems in a multitasking environment. The software in no way imposes restrictions on the number of configurable elements as I/O channels and function calls. More than 100 functions allow a rigorous process control concerning events depending on time and logic. The user interface is adaptable to any process setup by configurable menus whose number is limited only by the memory capacity of the computer. The menu configuration includes graphic layouts and the actions performed in response to operator input. Special data structures allow the integration of data acquired by analytical instruments such as mass spectrometers, IR-analyzers etc. The software is written in 'C' and easily portable to any multitasking operating system like OS9, real time UNIX or QNX. The tests were run using OS9 on VME computer systems.

Key words. Direct digital control; DDC; computer; process control; biotechnology.

Introduction

During the last 10 years I have had the pleasure of contributing to all the DECHEMA-Courses carried out at the ETH-Institute of Biotechnology under the direction of Armin Fiechter with the subject 'measuring techniques and computer control'. Comparing the notes of my first and of my last lecture shows that the progress in microprocessor-based computer systems is almost incredible. Also, the previous knowledge expected of the participants in a course has increased to a level that is no longer comparable to that in the days of 8-bit processors, memory limits of 64 kB and unattainable hard disk drives. Today sophisticated cultivation techniques are based on computer coupled bioreactors, together with numerous peripheral analytical instruments. High speed 32-bit processors together with arithmetic coprocessors, fast interfaces and communication lines, virtually unlimited memory of many megabytes and cheap mass storage devices meet the requirements for solving practically all calculable problems in fermentation under real time conditions. However, to take advantage of the high hardware performance adequate software must be available. Of course the well-known process control systems, used primarily in chemical engineering, have been available for a long time, but for a lot of smaller applications in biotechnology their price is beyond the financial possibilities of many working groups. Additionally, their complexity makes it difficult or costly to adapt the configuration to continually changing process environments. Smaller software packages, in most cases using PCs, often suffer from restrictions concerning the configurable elements and are more usually suitable for simple data acquisition and evaluation. To overcome these difficulties CAROLINE I was developed, taking advantage of the power provided by modern microcomputer systems and fulfilling the requirements of logical and time-dependent control for complex biotechnical processes. Besides the possibility of an easy and flexible configuration of the proper DDC-machine, the development of a suitable user interface, adaptable to any process environment, was of primary importance. To study the software requirements and especially the required data structures, two extreme biotechnical installations were investigated theoretically and later used for real test purposes:

Example 1: A large production street including medium tanks, bioreactors of different sizes and downstream equipment. Assuming automatic sterilization and transfers between all vessels and machines the installation needs dozens of analog inputs and outputs and hundreds of controlled valves. The control strategies are rather simple, the resulting complexity has only a quantitative aspect.

Example 2: 6 small bioreactors in a research environment are each connected to a mass spectrometer for gas analysis. A total of 18 balances allows the control of substrate feeding depending on the results of the mass spectrometer data.

The system analysis outlined in the following section is based on the usual terminology in chemical engineering and on concepts kept in DIN standards [1]. From the beginning of the development any software restrictions such as the number of allowed interface channels or configurable function calls were forbidden, the restrictions should be imposed only by hardware. Programming conflicts between speed and memory requirements were always solved in favor of speed.

System analysis

General remarks. From the engineering viewpoint the problem may be solved by defining three hierarchical operation levels: unit operations, process units [1] and plant unit. On the lowest level, unit operations such as heating, stirring, transportation etc. are executed in a controlled manner concerning time, logic and process hardware correlations. Primary software requirement is high execution speed of all required fundamental functions such as input/output drivers, logical switchers of

program flow, arithmetic operations, control algorithms, rule checkers etc. Usually that level is called a direct digital control- or DDC-system. On the second level, the unit operations are grouped into process units. Logistically they represent autonomous working units defined by process hardware boundaries. Such a process unit may be a reactor, a storage vessel, a transfer pipe, any piece of downstream equipment etc. Execution speed is of secondary importance, primary requirements are parameterization by recipes, recipe management, documentation and automatic locking. On the highest level, the process units are summarized to a plant unit allowing the coordination of all plant equipment for a defined purpose. Recipe management and documentation are the main tasks on that level.

For a transparent software structure three program packages should be developed according to the operating levels proposed above. This partition also allows the representation of the levels by networked but independent computers. The above-mentioned example 1) requires four DDC-computers and one supervisor computer all connected to a local Ethernet. CAROLINE I now represents the software package on the DDC-level. For the two higher levels CAROLINEs II and III are currently being developed.

To start the detailed analysis of the DDC-level a simple temperature control loop is assumed: the temperature is measured by a Pt-100 sensor in a stirred vessel, an amplifier converts the transmitter signal to a voltage output in the range of 0 to 10 V connected, via an isolation amplifier, to an A/D converter. The A/D-converter is mounted on a computer board and via the bus backplane is accessible by the CPU. Depending on the difference between setpoint and actual value, the control algorithm acts either on a cooling or heating device. The cooling device consists of an ON/OFF valve switching the cooling water on or off, flowing through a heat exchanger mounted inside the stirred liquid. The heater inside the vessel is powered by a current switched on or off by a solid state relay. This means that both devices are driven in pulse modulation mode, therefore the computer system, additionally to the A/D board, needs some digital outputs, e.g. on a relay board. Any digital computer works in a discrete manner; it performs its task step by step. Therefore the temperature control outlined above must be solved stepwise. In consequence a DDC-system principially represents a high speed stepping machine triggering step actions in a systematic way concerning time and logic. The procedure must be known to the machine and for a multipurpose system it is fixed in a changeable configuration.

It is obvious that in our example the actual temperature is required as a first step. The value obtained must be stored somewhere for display purposes. It is then taken as input for the next step calling the controller algorithm. In addition to the actual value, the controller needs the setpoint stored in a register as a second input. As soon as the controller has calculated its output the value should be available in a display register. How can the controller now act, either on the heater or on the cooler? A controller of this type is usually called a split range controller, and has a dead band avoiding under all circumstances simultaneous heating and cooling. Therefore in the next step some kind of dead band checker is required. Depending on the sign of the controller output, and on the dead band, it directs the controller output to the relay driver activating the heater or cooler, or it switches both devices off if the controller output is inside the dead band. Additionally it selects between the two controller parameter sets for heating or cooling. The controller algorithm for the usual PID control must be called in constant sampling intervals chosen according to the time constant of the loop. Since the ON/OFF outputs are pulse modulated, the system must call the output drivers asynchronously to the loop sampling interval. These calls are required at short time intervals.

This small example of temperature control permits a deeper insight into the required step actions and data structures. Step actions represent parameterized function calls. Therefore each step is accompanied by a small data structure called a step descriptor. It should be as short as possible since large systems consist of several thousands of steps. Because many step functions need information concerning the physical environment (measuring range, alarm limits) and registers for data transfer and data storage, the step descriptor is not suitable. In addition, this information always accompanies a succession of step actions. Such a succession is called a phase. In our temperature control example all the step actions are related to the same phase. The registers mentioned above are included in the phase descriptor. With regard to the temperature controller it is apparent that it is necessary to be able to switch the loop on or off, to select the manual mode, and to define the sampling interval. These data are defined in a data structure called a sequence descriptor.

Summing up the analytical results achieved, the situation shows the following picture: A sequence consists of a succession of step actions and is executed depending on its state and on the sampling interval. Consecutive step actions inside a sequence are accompanied by phase descriptors. The total number of sequences, step actions and phases is unlimited. In any one sequence, any number of phases is possible and the same phase can be used in more than one sequence. One problem still remains unsolved: the asynchronous call of pulse modulated digital outputs. Additionally the possibility for pulsing outputs in a constant measure and for switching an unlimited number of valves at the same time according to configured output patterns would be very useful, especially for automatic sterilization. To solve this problem, each device is fixed in a device descriptor offering the possibility to act asynchronously in any desired manner on the output and to fetch inputs at any time. More data

structures have been added to the explained descriptors for sequences, phases, step actions and switching patterns: a unique general system descriptor containing the basic system sampling interval (usually of the order of 100 ms), pass numbers, information concerning the system state and global system timers; flag and mask registers for sophisticated logical operations. A short description of each descriptor type is given below. Finally the user interface is described.

Step actions. A step action is configured in the following way:

Step «number» «step action»

All configuration is done writing text files. They are transformed off line by interpreters to machine code files. There are two available kinds of step actions:
1) Step functions: They are direct, unconditional calls of functions. Depending on the function some arguments are required. Therefore a step function is configured as:

Step «number» «step function»
«step function»: «function name» (arguments)

All user relevant data can be used as arguments.
2) Rules: A rule is a conditional call of a step function consisting of a rule head and a rule tail. All rule heads start with an IF followed by a comparison of two arguments. For flag and mask registers all boolean operations are available. The rule tail consists of one function call executed if the rule result is true. The syntax of rule configuration is shown below:

Step «number» «rule head» «step function»

The step functions can be classified as follows:
1. Interface drivers: They handle the interface boards according to the information found in the device descriptor. The device descriptor number is a mandatory argument for each driver function.
2. Digital filters: Calculation of mean values or first or second order low pass filters.
3. Controllers: ON/OFF, ON/OFF 3 state and PID controllers. The second order PID control algorithm is based on Isermann[2]. Concepts for adaptive controllers after Kraus[3] have been programmed by G. Burri, but up to now exist in an experimental state.
4. Flow control: Change of the stepping succession inside a sequence. The dead band checker belongs to that function class.
5. Stack operations: Any argument can be pushed onto a stack. Subsequently the mathematical operations are done stepwise in the manner of reverse polish notation. All arithmetic, transcendental and trigonometric functions usually defined for a C-compiler are available.
6. Direct execution of mathematical operations: The very important assignment functions or value transfers, e.g. between phases, belong in this class. Depending on the problem and on the user's taste, stack operations or directly executed functions can be selected.
7. Output setting: Switching of single outputs or according to pattern tables. The behavior of the output depends on the declared type (pulsing or ON/OFF).
8. Manipulation of mask and flag registers for logical operations.

Phases. They are configured in tabular form and referenced by unique logical numbers starting at 0. Each step or any succession of step actions may be correlated to any phase. It represents a rather complex data structure. The defined variables accessible as step action arguments are classified as follows:
1. Physical environment: Measuring range, name and dimension of the engineering unit involved, registers to save actual values.
2. Alarm definitions: 2 upper and lower alarm limits, alarm priorities for each limit pair.
3. Predefined registers named with single characters: x: input and output of each step function, w: setpoint, m: output values in manual mode, y: disturbance superposition.
4. General purpose user registers: a,b,c: floating point values; h,i,k: integer values.
5. Controller related parameters: PID-parameters, dead band limits.
6. Graphic parameters: upper and lower scaling for trends related to the phase-correlated engineering unit.

Sequences. The sequence data are configured as a header for a following step succession. It is characterized by a name, a type (initializer, one shoot or looping), a start mode taken into account when the system is switched ON and a sampling interval.

Devices. They are configured in tabular form, referenced by logical numbers and characterized by the following data: name, type (analog input, analog output, digital input, digital output: ON/OFF, pulse modulated, constantly pulsing, alarm output), address of the interface board, physical channel number on the interface board, safe output state, pulse period, pulse width for constant pulsing devices, a correlated alarm level taken into account by the alarm handler and offset to compensate the usual 4 mA offset in I/O current loops.

Flag registers. It is only necessary to indicate the number of the total 16-bit-wide flag registers required. The single bits are set dynamically by step actions referencing the registers by logical numbers.

Masks. 16-bit-wide dynamic flag registers can be compared logically to static mask registers. They are configured by tables and referenced by logical numbers.

Output patterns. They are defined in tables containing the logical pattern numbers, the devices involved and the output state of each device needed by a single pattern. The number of patterns and pattern tables is not limited. These tables offer the only convenient way of handling several hundreds of valves.

General system data. They are declared as a header at the top of the configuration file. It essentially contains the basic system sampling interval, the pass numbers for 9 operator levels, the global alarm mode, print interval and host transfer interval for periodical outputs, information for the trend graphics and global alarm handling.

The user interface

The acceptance for any software depends to a high degree on the possibilities of the user interface. A sophisticated user interface technique is a primary requirement in the field of process control, especially when this involves an enormous variability of configurations. Under real time conditions, and especially in stress situations, the operator depends on adequate support by displays and input possibilities. Therefore all menu techniques using non-configurable masks and operating with nesting are unsuitable for efficient system handling. The development of the user interface presented was initiated taking the following requirements in account:

- The layout of each display (position, color and height of static or dynamic text; windows for bar and trend graphics) is configured in a text file.
- Each input key can be covered in each menu with any selected function. The necessary prompt texts and correlated error messages are configurable too.
- Each possible input is optionally protected by pass numbers arranged in 10 levels.
- It should be possible to switch alternatively from one menu to any other without any nesting hierarchy.
- Menus are grouped into classes representing different languages or operator levels.
- Conditional display of dynamic text should improve the survey of special system states.
- Alarms are automatically displayed and fixed in a history file.
- Simple ASCII-terminals should be suitable for menus without any full graphics.

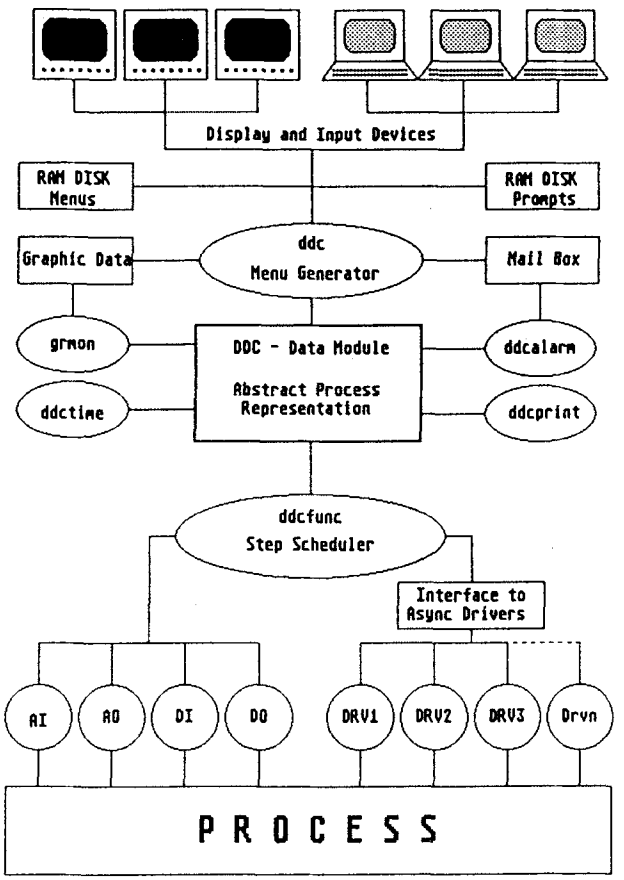
The extreme speed of today's microprocessors allows the interpretation within milliseconds of RAM-disk-residing, menu configuration text files during each menu switch. Different directories on the RAM-disks represent the different menu classes (languages, operator levels). The number of configurable menus is only limited by the capacity of the RAM-disk (usually one megabyte). For each menu a maximum of 200 actions and 11 graphic windows can be configured. They are always newly read during each menu switch.

Implementation

Taking into account all the requirements outlined above, a multitasking and multiuser realtime operating system is an absolute prerequisite. It allows a flexible and transparent software structure correlating the main software functions to different tasks running in the fore- or background. The single tasks communicate by shared data allocated dynamically according to the configuration requirements. All solutions based on a single task operating system finish in a dead end.

CAROLINE I is programmed in 'C'. Up to now all tests have been made using OS9 from Microware [4] suitable for Motorola processor based computer systems. As computer systems, VME-systems and the ATARI MEGA ST 2 were used. The general software structure is shown by the figure.

After the booting of the operating system, including the creation of a RAM disk, a shell procedure performs the following actions: The data modules DDCDA, MAILBOX and INTERF are loaded into the memory. DDCDA is shared by all tasks and contains all information



General structure of CAROLINE I. Rectangles: Memory resident data structures except PROCESS, representing the 'outside'world. Ellipses: Concurrent running task. 'ddc' can run in the foreground, called by operators from remote ASCII-terminals. Circles: Interface devices: Boards on the computer bus driven directly by 'ddcfunc' or special drivers which transfer data from remote devices to a configurable interface data module.

about the DDC machine. Its size depends on the configuration and can reach several hundreds of kilobytes. MAILBOX is used for error and alarm information exchange between back- and foreground tasks. INTERF allows the communication with any special drivers usually serving serial connections to external analytical instruments or data loggers. The number of these interface channels of special type is not limited; they can also be used as additional registers.

After all date modules have been loaded, the shell procedure copies all menu and prompt text files from the hard- to the RAM-disk. This is done only for higher execution speed during menu switches.

Now all background tasks are started: *ddctime* organizes all timing requirements, it controls all the global system and local sequence timers; *grmon* updates the graphic data in configured time intervals for all analog inputs available to the display programs for the generation of trend graphics; *ddcalarm* surveys the alarm limits according to the wishes of the operator and performs the necessary actions in case of an alarm; *ddcprint* optionally transmits periodically data to a printer or a host computer; *ddcfunc* executes the configured step actions starting at every system sampling interval with highest priority, it checks sequence states and sequence sampling intervals and it fires a step sequence if all the necessary conditions related to a specific sequence are reached.

The task *ddc*, responsible for the user interface, runs in the background for process field stations equipped with color monitors and membrane shielded keys. It is not possible to leave *ddc* at these work stations. Contrary to that mode an operator at a remote ASCII- or color-terminal starts *ddc* in the foreground and can stop and restart it as required.

Performance and reliability

The DDC scheduler measures the number of time slices needed for each scan. The display program can indicate the resulting value. The example 2 outlined in the introduction was tested over a long time. As processor the Motorola 68020 together with the coprocessor 68681 was used, running at 16.5 MHz. The configuration allows the simultaneous operation of 48 control loops with sampling intervals in the magnitude of seconds. The scheduler task *dcfunc* starts in intervals of 100 ms. Given a system tick length of 10 ms the scheduler needs 1–2 ticks, corresponding to 10–20 ms, for one scan. No retardation of the display program effected by *ddcfunc* running with maximum priority was ever observed. It seems that the performance of CAROLINE I is high enough for any hardware configuration imposed onto one system.

The reliability of a system is almost wholly dependent on the reliability of the interface boards, assuming correctly assembled computer hardware and a battery-driven backup power supply. The price should never be a criterion for the selection of interface boards, especially for systems running at a high clock frequency. The boards should be tested with high frequency access over a long time to detect the very rare errors generated by the address-decoding hardware. During our tests we lost months for the localization of errors generated by interface boards. (We are kindly concealing the name of the well-known manufacturer.) Otherwise, the reliability of well-manufactured present-day computer systems is on a much higher level than that of the process hardware (valves, motors, seales, probes etc.).

Software stability is quite another problem. But modern software engineering concepts and tests under extreme stress conditions can warrant the assumption of practically 100% stability.

Acknowledgments. The DDC-system might never have matured without the constant use, suggestions, and constructive criticism of Georg Burri, Ettore DaPra and Markus Oberholzer who tested the system at MBR Bioreactor AG, Wetzikon (Switzerland) with very complex configurations on real systems.

1 Uhlig, R. J., Erstellen von Ablaufsteuerungen für Chargenprozesse mit wechselnden Rezepturen. atp *29* (1987) 17–23.
2 Isermann, R., Digitale Regelsysteme, Vol. I: Grundlagen, Deterministische Regelungen. Springer, Berlin 1987.
3 Kraus, F. J., Das Vergessen in rekursiven Parameterschätzverfahren. Thesis ETH Zürich 8012 (1986).
4 Microware Systems Corporation. OS-9/68000 Operating System. Technical Manual. Ed. W. Miller. Des Moines, Iowa 1987.

0014-4754/89/11-12/1030-05$1.50 + 0.20/0