

PC-based direct control flattens control hierarchy, opens information flow

Personal computer (PC) technology, while already entrenched in office environments, is just beginning to have an impact on industrial automation. For the most part, PCs are used in factories and plants to handle the operator interface (OI) function for programmable controllers (PLCs) or distributed control systems (DCSs). In this capacity, their involvement is limited to indirect control or, at best, Supervisory control functions (perhaps by detecting and organising alarms).

But few expect this trend to continue. Most see PCs getting heavily involved in discrete control - that is, the gathering of real-world inputs and use of control algorithms to provide control outputs directly back to the machine or process. Implementing direct control with open PC technology gets rid of an intermediate level of proprietary hardware, thereby reducing costs and eliminating a barrier to information flow. In addition, it flattens the traditional automation hierarchy that segments control functions into layers. In this hierarchy the control layer typically includes a PLC or other dedicated control hardware. The supervisory control layer is responsible for collecting and translating the PLC data into a human readable form as well as into information that can be shared with the rest of the organisation.

In the direct control model built on open PC-based technology, these functions are performed by software modules that coexist on one or more industry standard PCs. This eliminates the need for multiple real-time

databases containing process information.

Industry standard PC hardware technology can be used to produce a wide range of open control platforms, ranging from embedded controllers to control room workstations. Open control greatly improves the buyer's price/performance ratio by enabling him or her to use off-the-shelf components available from multiple vendors. Several bus architectures can be used to build open control systems - for example, PIC, PC104, ISA (and passive backplane ISA), EISA, STD32 and VME. Many different interfaces for I/O are available with different capabilities, in addition to boards that connect to all of the various device networks. (Fig 1) Open control products are also becoming available from PLC vendors that use the proven PLC I/O architecture while replacing proprietary controllers with PC-compatible processors.

Open control implies that hardware can be selected from different vendors to build industrial controls and logically this should apply to software as well. However,

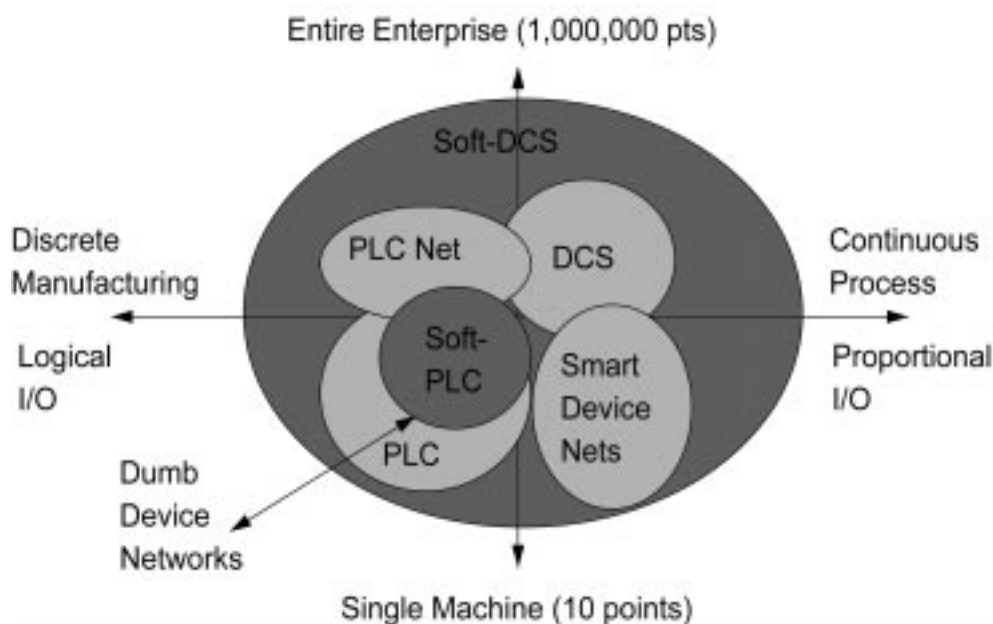


Figure 1. Many different interfaces for I/O are available with different capabilities, in addition to boards that connect to all of the various device networks.

DISTRIBUTED CONTROL

for reasons explained later in this article, compatibility standards for industrial control software have been slower to develop. Thus, software compatibility still is limited.

The following examines a number of software issues that must be addressed when applying PC based direct control. It looks at the concept of control itself as defined by S88 (an ANSI/ISA standard for batch control), discusses real-time system considerations, covers real-time operating systems (OSs) and explains how various reliability factors are affected by OS architectures.

DIFFERENT KINDS OF CONTROL: S88 TERMINOLOGY

At the heart of many arguments about implementing direct control via PC-based technology are misunderstandings about the term control. Part one of this standard defines models and terminology for control that include batch processing, but are also generally applicable to continuous processes. The intent of this standard is to provide a method of organising and stating control requirements. The primary philosophy embodied in the S88 standard is that process objectives are achieved by applying elements of procedural control to elements of a hierarchy of equipment used in manufacturing.

ISA's S88 defines three kinds of control: basic control, procedural control and co-ordination control. A description of each follows.

Basic control comprises the control dedicated to establishing and maintaining a specific state of equipment and process. It includes feedback controls, interlocks and exception handling (alarms), as well as repetitive discrete or sequential control.

Procedural control directs equipment-oriented actions to take place in an ordered sequence to carry out a process-oriented task. While procedural control is an essential characteristic of batch control, it can also be applied to start-up and shutdown tasks for any continuous process. S88 defines four hierarchical levels of procedural control: procedures, unit procedures, operations and phases.

The lowest of these, the phase level, is concerned with enabling, disabling or altering basic control functions of all kinds. Examples of activities carried out in a phase might include changing regulatory control modes, setpoints or outputs, changing controller constants, clearing or setting alarms or changing alarm limits.

Co-ordination control directs, initiates and/or modifies the execution of procedural control, but is not structured along a specific process-oriented task. Examples of co-ordination control include supervising availability or capacity of equipment, allocating equipment to batches and scheduling procedural elements so that production meets specific targets.¹

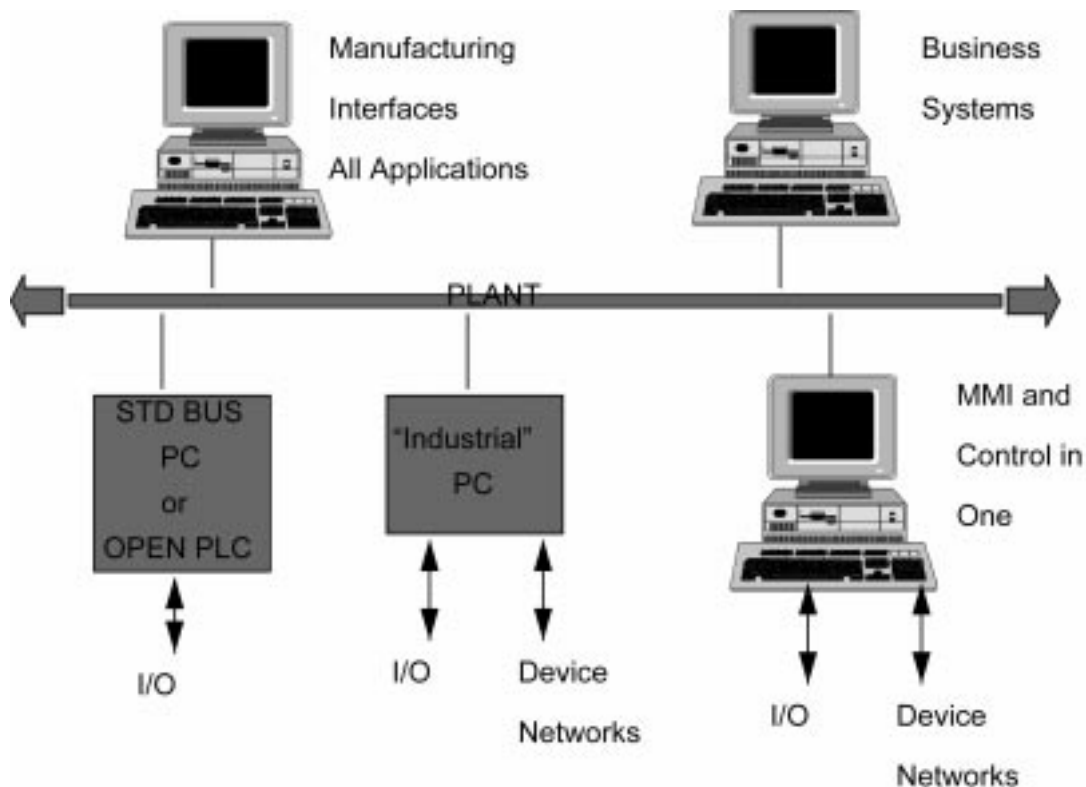


Figure 2. Basic control and some procedural controls traditionally have used dedicated electronics.

**Ad
Green Hills**

DIFFERENT SOFTWARE FOR DIFFERENT KINDS OF CONTROL

Basic control and some procedural controls traditionally have used dedicated electronics. Using the direct control concept, all three different kinds of software in either a single or network of open control systems (fig 2). However, the three kinds of control lead to very different kinds of software designs. Basic control functions require repeated software functions that occur on a regular time interval typically on the order of 1ms to 1s.

Software for procedural control must repetitively evaluate certain process conditions to determine when a new phase must begin, but the functions of implementing a phase typically take place only at the beginning of a new phase. Procedural control software may be thought of as being driven by external events. Coordination control software operates in response to changing business or plant design conditions and will typically be driven by interactions with the user, perhaps in response to entering a new number in a spreadsheet or creating a new database record.

Direct control is concerned with enabling basic control functions to coexist with other types of control in open control systems.

DIFFERENT PARADIGMS FOR CONTROL PROGRAMMING

Because we often mean different things when we talk about control, there is a diversity of views about how control programming should be performed and organised. Ladder logic was design to emulate relay circuits commonly used to automate discrete manufacturing applications. In ladder logic, lines represent logical conditions, true and false corresponding to energised and de-energised wires. In the process industries, instrument diagrams display control functions as blocks and use lines to represent the instrument signals carried between various kinds of control equipment. Function blocks and ladder logic are still the predominant methods used to program DSCs and PLCs.

In the PC world, general purpose programming languages, particularly C and C++, dominate the scene. Basic, flow sheets and database query languages (especially SQL) are also commonly used PC-programming tools. The high cost of learning a new language and adapting it for use with controls has been no barrier to the acceptance of PC-based control technology. This barrier has come down with the introduction of a rich variety of programming methods specifically designed to take advantage of direct control.

Two new control paradigms have been introduced alongside of function block and ladder logic methods: flowchart programming; using blocks to represent procedural phases in which basic control functions take place, while lines represent logical flow of control from one phase to another. This approach to programming has a strong foothold in the discrete manufacturing industries and seems well suited to implementing procedural control as outlined in S88.

There is one open standard for flowchart program-

ming, referred to as SFC (sequential functional control) in the IEC 1131-3 standard. The structure of the SFC language neatly matches the definitions for procedural controls in the S88 standard. There are also other proprietary flowchart programming methods embodied in specific software products. Flowchart programming is easily and efficiently mapped to high level languages like C.

Virtual device programming creates control logic by adding and configuring devices in a database. Each database record corresponds to a virtual instrument with predefined capabilities and a unique tag name. Relationships between virtual devices are represented by references to other tag names in the configuration record. Virtual device programming doesn't feel like programming at all, because it can be performed incrementally and can deal with on-line changes. Direct control enables the virtual device paradigm because it makes it possible to interleave changes in the control logic of a single virtual device in between execution cycles. This style of control programming is ideally suited for basic control functions that must not be interrupted.

REAL-TIME SYSTEMS: WHAT ARE THEY?

Real-time systems must meet deadlines. This is a pretty good definition of what we mean by real-time. Deterministic real-time systems always meet deadlines, whereas non-deterministic systems might meet deadlines part of the time, depending on circumstances.

Deadlines are typically defined in terms of response to a particular event within a certain length of time. Both procedural and basic controls introduce deadlines.

The ability of a system to meet deadlines - reliably - depends on: How fast the CPU and its interfaces are, how many different events it must respond, to how the software is devised and also the design of the OS. If any one of these design aspects is inadequate, deterministic real time performance is impossible.

Real-time OSs makes it possible to create real-time control systems, but they can't guarantee that performance will be acceptable. On the other hand, unless an OS is designed properly, even modest real-time performance may not be achievable.²

Real-time systems are sometimes further classified as having hard or soft real-time requirements, The properties of a hard real-time system are: No lateness is accepted under any circumstances, unless results occur if late, catastrophic failure occurs if deadline is infinitely high. A good example of a hard real-time system is the fly-by-wire control system of an aircraft. A soft real-time system is characterised by the rising cost for lateness of results and acceptance of lower performance for lateness.³

An OS must meet certain minimum criteria to be used to build a hard real-time system. When these requirements are met, only then can we call it a real-time OS (RTOS).

WHY A REAL-TIME OPERATING SYSTEM IS NEEDED.

Control systems must meet the needs of many potential end users, including control engineers, process engineers, operators and also the managers of business processes. It is this last customer group that is likely to demand compatibility with Microsoft® Windows NT™, since NT is clearly poised to satisfy business process computing needs. What could be simpler than to adopt Windows NT as the control system platform as well?

Of course, this simple evaluation ignores other issues that are important to people closer to manufacturing. The designs of computer systems for business processes and those of control systems must meet very different requirements. One way to appreciate just how different is to look at how the business system user and process control system resolve common computer system design issues.

As you can see in the table on page XX, in many cases management policies developed for business users are clearly unacceptable for process control users. With these highly divergent viewpoints about how to address the most common computer issues, it is easy to see why control system users need a system that is designed specifically for them, while simultaneously staying compatible with systems designed for business users.

The capabilities of a real-time OS enable control applications to meet the deadlines common to basic and procedural control functions. However, neither Windows™ 95, Windows NT, nor UNIX are real-time OSs. Either modifying a general-purpose OS, or choosing a real-time OS for control with connectivity to general-purpose systems may provide real-time capabilities.

It is usually best to have control processing take priority over everything else, so the performance of the PC is somewhat degraded when using the multitasking open-control approach. The degree of performance loss depends on how much control processing is done and the overhead associated with real-time OS functions. Context switch time - i.e., the time it takes an OS to recognize an interrupt request and to act upon it - is important.

HOW AN OS AFFECTS RELIABILITY

The reliability of a PC-based control system is measured in a way similar to that of any instrument - as MTBF (mean time between failures). What is less obvious is that the most common mode of failure for a PC-based system is not in any way related to the PC hardware itself, but rather to the OS. Thus, the selection of an appropriate OS can have a very large impact on the perceived reliability.

A control system has failed any time it is not operating as designed for PC-based systems, this happens every time a reboot occurs. When a PC-based system requires a reboot, it is not merely an inconvenience, but also can cause the loss of control over a portion of a manufacturing process. This will likely shut down the

process - or worse. The time to restart the process is often substantially longer than that required to restart the computer and typically costs in wasted raw materials and ruined product. How do re boots affect PCs that are only used as OI stations or for supervisory control? While a reboot may not quite be as critical in this case, there are few processes that can be operated safely for very long without a functional OI.

The need for reliability goes beyond the requirement of a real-time system. High priority, real-time tasks should continue to operate even when non-real-time portions of the system do not. For example, if a Windows NT system provided with real-time extensions should encounter a disk drive fault, Windows NT will fail. Even if the real-time tasks are still functioning correctly, the process control data will be inaccessible. Furthermore, before the operator can restore his view of the process, it will be necessary to reboot the computer, also halting the real-time tasks.

PCs have come a long way in the last 15 years. Some of you will recall that most software for early Apple PCs was designed so that you had to insert the program disk and reboot the computer to load each program. Contrast this with today's multitasking environment, in which many programs can run simultaneously. One of the principal benefits of PC-based control is that the computer can provide several functions simultaneously - for example data logging, trend display, an animated process view, various alarm handling services, all while reserving a small portion of CPU time for direct process control.

In multitasking process control applications, reliability is enhanced by using an OS that allows various maintenance procedures to be performed without rebooting. Consider the following common procedures that may require a reboot to be performed and thus may be considered to have caused downtime, depending on the OS:

- Loading new software, including the OS
- Changing a network address or changing from one file server to another
- Starting new network services
- Starting or changing I/O system drivers
- Hot-swapping a keyboard or mouse, particularly with a different model
- Replacing the monitor, perhaps with one that requires a different synch rate or supports a high resolution
- Starting or stopping database services
- Stopping a runaway program
- Stopping a runaway program that monopolises all console input devices (possible from another node in the network, or a dumb terminal on some systems)
- Changing the system to redirect all file operations (including OS files) away from a local (failed) disk drive to a remote one
- Accessing OS functions, even after a complete disk drive failure involving OS files
- Allowing full remote access to the system while key-

DISTRIBUTED CONTROL

board and monitor have failed, through either network or serial port.

This list makes it easy to score an OS in terms of its need to be rebooted under various conditions. Based on our experience, systems designed using DOS or Windows 95 score poorly and those designed using Windows NT score only marginally better. Operating systems specifically designed for real-time applications score well. For example, we've found that QNX scores a perfect 12 on this checklist. As a result of the ability to perform maintenance over a network connection and the highly modular nature of the OS (and thanks to an uninterruptible power supply) several QNX-based process control systems at Olin have been in constant service for longer than two years without a reboot.

RECOGNISING OPEN VS. PROPRIETARY SOLUTIONS

We've seen the benefits of real-time systems, but today the concept of openness is just as important in the overall solution. The philosophy of open systems is that by agreeing to comply with published standards, vendors can produce interoperable products. These products should be more attractive to customers because they provide a wider range of choices, along

with capabilities that could not be achieved using any one product alone.

Is a control system product that conforms to the published specifications for Windows NT necessarily an open control product? When this software claims to perform direct control, the answer is usually no. As already explained, all versions of Windows must be modified to provide real-time functions. When these modifications are themselves proprietary to the software vendor, they create a barrier to compatibility with software products from other vendors. Even when these real-time modifications are made available to other vendors most of these systems will not allow two direct process control programs to run simultaneously. Windows NT, itself, does not define a standard interface between programs and real-time OS functions, because these functions do not exist. However, there is a subset of the POSIX standard (derived from UNIX systems) for real-time OSs, which does define such a standard interface. (Currently, Windows NT supports POSIX at the core application programming interface [API] for C language.) Control software that uses the POSIX standard is not necessarily open, because the standard applies to source code only and does not guarantee that the compiled binary programs you purchase will be interoperable either. However, the real-

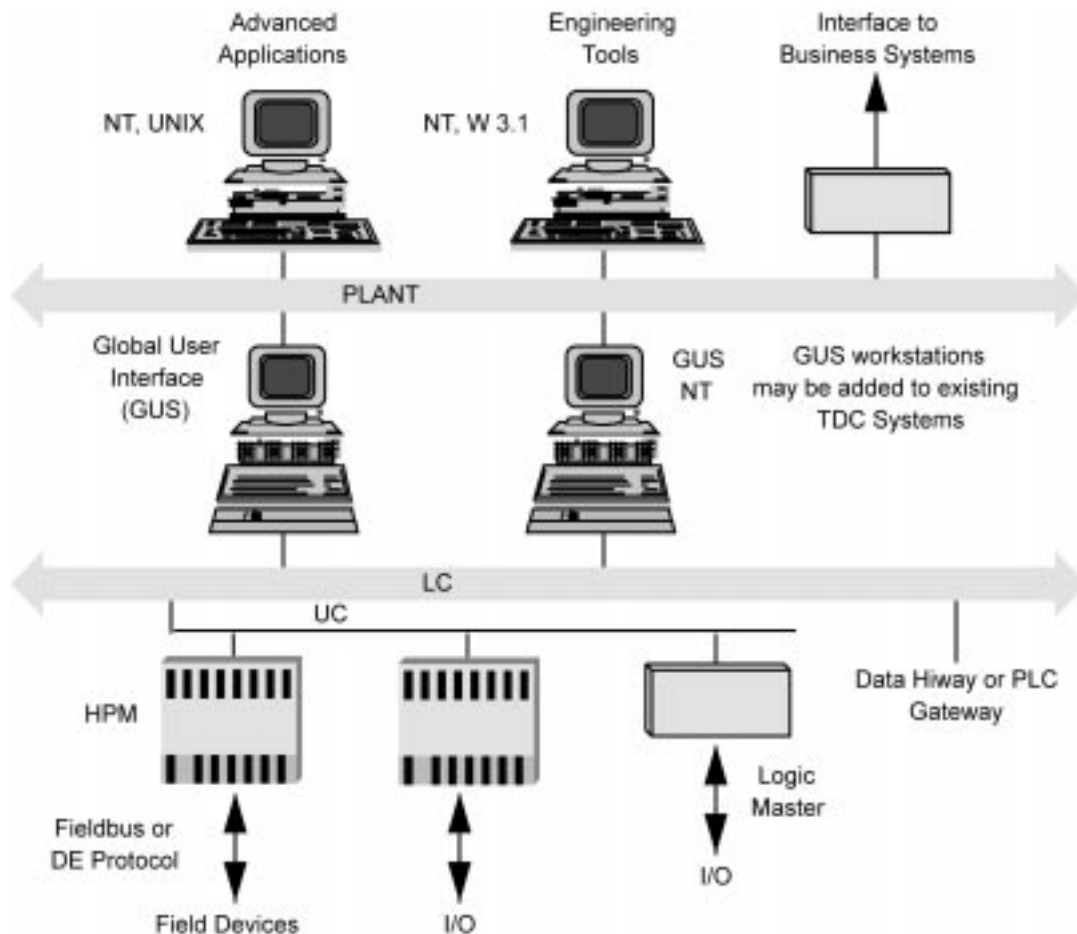


Figure 3. The emerging OPC (OLE for Process Control) standard offers a different strategy to create an environment for open control software based on Windows NT.

time POSIX standard is at least a step in the right direction.

The emerging OPC (OLE for Process Control) standard offers a different strategy to create an environment for open control software based on Windows NT (fig 3). This is really a misnomer since OPC really uses the COM (Component Object Model) portion of the OLE (Object Linking and Embedding) standard developed by Microsoft. But no matter what you call it, OPC is just a way to allow programs to communicate real-time information, particularly process data, efficiently.

In the OPC paradigm, programs that generate real-time information, such as direct control programs, data acquisition programs or drivers for any instrument with a PC interface, are Servers. Programs that receive process data, such as operator interfaces trending packages, data historians or any program that can accept OLE objects, including spreadsheets, databases, visual basic programs and even word processing programs, are Clients. OPC compliance does not guarantee that direct control software from different vendors can coexist, but it will help to ensure that the real-time data it generates will be available to other Windows programs.

**Ad
ES 1**

REFERENCES

1. ANSI/ISA-S88.01-1995 "Batch control Part 1: Models and terminology," ISA 1995.
2. Dr Martine Timmerman, "Windows NT as a Real-Time OS?" Real-Time Magazine 1996.
3. Ibid.

David Cawfield began development of PC based process control software in 1983, while working as a process development engineer for Olin's Chlor-Alkali division. Cawfield has more than 60 patents for various chemical and electrochemical processes and equipment, as well as for methods of process control and high-speed, model predictive control. Prior to employment with Olin in 1981, he worked in R&D for Proctor and Gamble and Monsanto. Cawfield has a BS in Chemical Engineering from the University of Missouri at Rolla (1977) and is a member of AIChE, ISA and ICS.