Journal of Computing and Information Science in Engineering

Technology Review

Real-Time Computing in Open Systems for Manufacturing

Harry H. Cheng
Mem. ASME, Department of Mechanical and Aeronautical Engineering, University of California, One Shields Avenue, Davis, CA 95616

Frederick Proctor, John L. Michaloski, and William P. Shackleford
Intelligent Systems Division, National Institute of Standards and Technology, 100 Bureau Drive, Gaithersburg, MD 20899

This article reviews various mechanisms in languages and operating systems for deterministic real-time computing. Open-architecture systems will be defined and their applications in manufacturing will be addressed. Market directions for open-architecture manufacturing systems will be surveyed. Performance issues based on real-time, reliability, and safety will be discussed relating to manufacturing factory automation designed and implemented with component-based, plug-and-play open-architecture. [DOI: 10.1115/1.1351819]

1 Introduction

Machine controllers for factory automation have evolved from their origins in mechanical devices for regulating and transforming engine power, into computers ranging in size from tiny embedded chips to networks of high-performance workstations. However, the physical constraints of factory floor operation, including harsh environment, continual operation, high performance, and long capitalization life expectancy have limited machine controller's advancements so that they have historically lagged behind mainstream hardware and software technology. As special purpose systems, such factory automation is costly to support and maintain, and often, the provider is the sole supplier of parts and service. Moreover, such systems cannot easily adapt to meet new manufacturing demands or technology innovations.

Manufacturers realize the shortcomings of proprietary solutions and are eager to leverage commodity PC hardware and software in machine control. This has resulted in the trend in factory automation for open, standards-based computer products by opting for general-purpose off-the-shelf PC hardware and software technology wherever possible, and adapting it to the more stringent needs of the factory floor. Machine controllers used in factory automation built with this open, standards-based premise have been termed open-architecture controllers [1–3]. The open-architecture strategy is part of an effort to improve the agility of manufacturing by integrating the shop floor directly into the enterprise business systems. Numerous benefits to factory automation can be achieved by using commodity hardware and software from the computing and communication markets—e.g., improved time to market, increased production yield, lower product cost, increased production capacity, reduced inventory, and optimization of production workforce.

The trend toward adopting open, desktop technology on the shop floor means that the factory automation market may be poised to benefit but there are obstacles blocking immediate success. Many factory automation applications have strict safety, reliability, and real-time performance requirements not faced by the desktop computing products. This paper will study the impact of reliability and real-time performance requirements on using or adapting desktop open architecture tools and services for factory automation. Market surveys for available open desktop products based on applicability to factory automation will be presented. Typical factory automation that will be considered includes computer numerical control (CNC) for milling, material handling robotics, and coordinate measuring machine (CMM) for part inspection, Programmable Logic Controller (PLC), and General Motion Control (GMC) found in the packaging industry.

The paper will begin with a characterization of open-architecture and then review the implications that open means to factory automation on the shop floor. Next, the real-time requirements of factory automation will be discussed as well as the impact of these requirements on systems issues, such as operating system, communication and platform. A look at the concept of hard and soft real-time will the reviewed as an industry benchmark concept and as a more formalized correctness requirement. The paper continues with a look at programming technologies including programming languages and software development strategies, as they apply to developing open-architecture, real-time, machine control applications. The importance of component-based technology and the impact on factory automation will be examined. Finally, a summary will be presented considering the influence that real-time requirements have on the development of open architecture factory automation.

2 Open Architecture

An open-architecture is the definition of a set of components with well-defined behavior and well-defined relationships, and for which the definition is known to all. Often the definition is standardized, either through a standards body or through de facto adoption by a wide community. In some cases, component vendors provide the definition. Desktop computing is one of the most widely known examples of an open-architecture, where components include display monitors, input devices, storage devices, and software functions provided by the operating system or applications. Here, there is a combination of formal standards (e.g., VGA, IDE, SCSI, POSIX, CORBA) and de facto standards (e.g., BIOS, Win32, COM). Table 1 details the names and purpose of

Contributed by the Embedded Systems Committee for publication in the JOURNAL OF COMPUTING AND INFORMATION SCIENCE IN ENGINEERING. Manuscript received Sept. 2000; revised Dec. 2000. Editor: P. Khosla.

Table 1 Desktop open formal and de facto standards

Acronym/Name	Meaning	Purpose
ATM	Asynchronous Transfer Mode	Higher speed network technology
BIOS	Basic Input Output System	OS booting kernel on ROM
COM	Component Object Model	Microsoft Component Model
CORBA	Common Object Request Broker Architecture	OMG Distributed Component Model
DOS	Disk Operating System	Basic PC OS
Ethernet		Local Area Network(LAN) specification, now the IEEE 802.3
FDDI	Fiber Distributed Data Interface	ANSI network standard
Firewire	Apple trademarked name for IEEE 1394	IO external bus/network supporting plug-and-play hot plugging, and power.
IDE	Integrated Drive Electronics	ATA Disk Drive specification
PCMCIA	Personal Computer Memory Card International Association	Standard for small, devices, called PC Cards, to PCs.
POSIX	Portable Operating System Interface for UNIX	IEEE and ISO standards
SCSI	Small Computer System Interface	Parallel Interface
USB	Universal Serial Bus	IO external bus/network supporting plug-and-play, hot plugging and power.
VGA	Video Graphics Array	De facto PC display standard
Win32	Windows 32-bit API	Microsoft C API

many of the desktop open standards mentioned in this paper. In an open-architecture, in theory, any vendor can provide any component. With de facto standards, in practice, some components may only be available through a single vendor. However, as long as no onerous barriers exist that prevent any component from being provided by any party, the architecture is open.

Open-architecture systems differ widely in the degree to which they are ''open,'' in the intuitive sense of the word. In some cases, systems may be comprised of commodity hardware and open-source software, so that virtually every aspect of all components is publicly visible. In other cases, systems may consist of proprietary components with de facto interfaces provided by the vendors. In practice, open architecture systems balance the need for public disclosure of component and interface specifications with the need to protect intellectual property, so that the resulting system has market viability.

By definition, an open-architecture provides open access to real-time data and information that can be used to allow vertical integration of machines within the enterprise, and is critical to agile manufacturing. Middle-level information systems, called Manufacturing Execution Systems (MES), bridge the information gap between the upstream and downstream systems. MES systems can collect data from the machines to continually monitor and improve the manufacturing process leading to increased productivity and quality. MES track and manage all aspects of a job on the factory floor in near real-time. MES play essential supervisory and monitoring roles that link all levels of manufacturing and business operations. For example, they identify bottlenecks and material shortages on the shop floor, and they provide near real-time performance of manufacturing systems along with comparison with the past performance and projected results. Through this vertical integration, factory-floor information systems, which focus on the operation of mechatronic systems and on the control of processes, can communicate both directly and regularly with front office information systems, which are used for design, process planning, quality control, accounting, forecasting, and other resource planning activities. As a result, upstream information systems are aware of important manufacturing details such as the availability of appropriate tools; records of past process; maintenance schedule; or the status of work in progress. A growing push among manufacturers is to move to an Internet-enabled business requiring build-to-order vertical-integration manufacturing model that emphasizes low inventories, short cycle times and quick order execution. With enterprises faced with these time-to-market pressures, factory automation must be implemented quickly and modified easily. The open architecture ability to reconfigure or extend existing equipment to meet new needs is particularly powerful in meeting these challenges.

The potential benefits from open-architecture controllers include improved time to market, increased production yield, lower product cost, increased production capacity, reduced inventory, and optimization of production workforce. The potential productivity benefits are not just for end-users and OEMs, but the control vendors can also benefit from open-architecture and component-based technology. Using a standard framework in which to develop components, control vendors can concentrate on application-specific improvements that define their strategic market-share—as opposed to spending valuable programming resources reinventing and maintaining software ''plumbing.'' With the shortage for skilled software professionals on the rise, the necessity to leverage commodity software technology is even more compelling.

Open-architecture system development raises some issues. When systems are assembled using an open-architecture approach, the responsibility to make systems work shifts to the user or a systems integrator. When failures occur, it may be hard to determine which vendor's component is causing the problem, and can lead to vendor ''finger-pointing.'' Open systems are also often not tested to the same level of robustness that mature proprietary control systems have been, resulting in higher risk for some applications. Further, many factory automation applications have strict safety and reliability requirements not faced by the more widely known desktop computing architectures. Failure to control equipment like machine tools or robots can damage expensive parts, fixtures, and tooling, and can cause injury or death. Big automation end users are beginning to hold vendors contractually liable for downtime cost, which can reach tens of thousands of dollars a minute. Additionally, unlike the desktop PC industry where the hardware components are commodities, with the underlying operating system software dominated by a single company, the manufacturing industry is much more diversified with different requirements for various specific applications. Therefore, it is difficult to have a single model with real-time constraints which can be copied exactly to solve different manufacturing problems. In reality, a plug-and-play open-architecture, combined with component-based reconfigurable software, is the only solution to these various manufacturing problems.

Where safety and reliability are critical, the augmented architecture has emerged as an alternative to purely open approach, in which an open system acts as a front end to a proprietary real-time controller, through a controlled communication link. All access to the real-time controller is done via software provided by the vendor that runs on the open system. This prevents any modification of the real-time controller, except as allowed by the vendor's software interface on the open system. The augmented architecture is intended to support the integration of controller and factory data, for applications such as data logging, file sharing, resource or consumable status monitoring, and diagnostics. It is also intended to allow third parties to customize the graphical user interface. Some access to real-time controller data may be allowed, such as configuration parameters. In these cases, the open system is serving as a replacement for the traditional proprietary interface given to systems integrators for configuring the machine for installation or resale to customers.

The use of open-architectures is seen as vital to improving manufacturing within numerous industry groups, including the Open Modular Architecture Controller (OMAC) Users Group, OLE for Process Control (OPC), the Robotic Industries Association (RIA), and the Metrology Automation Association (MAA). OMAC is focusing on computer-numerical controllers (CNCs) for machine tools, and general motion control for the packaging industry. OPC is an industry consortium focusing on standard interface for the exchange of data between hardware field devices and automation/control or enterprise business applications. The RIA is focusing on robotics, and the MAA is focusing on metrology equipment such as coordinate measuring machine and portable inspection arms. Internationally, European companies have formed the Open System Architecture for Controls within Auto-

mation Systems (OSACA) consortium, and the Japanese have formed the Japan Factory Automation Open Systems Promotion Group (JOP).

OPC has been the leader in the realization of open systems products with over 250 manufacturers producing OPC compliant products. OPC replaces custom drivers used to connect different manufacturer's IO field device products. Before OPC, a different driver was required to connect Intellution's FIX to Allen-Bradley PLCs, or Siemens PLCs or GE Fanuc PLCs. At one time, Wonderware advertised over 200 such drivers. Now, one OPC driver is required to connect to any one of these PLCs running an OPC server. Moreover, the Intellution FIX can be replaced by a USDATA or Wonderware product as these conform to the standard OPC client interface.

3 Performance Requirements for Open Factory Automation

General-purpose commodity software is often difficult to use ''as-is'' in factory automation systems as it may not offer a fine enough degree of programmability, or offer enough performance, or it may hide system and resource policies and mechanisms and make them inaccessible to the software developer. These underlying features are considered part of a black box, where a best-effort is provided to all users. For most programming applications, fair allocation of resources works adequately, however, this policy is not sufficient for factory automation, which needs finer access and control in order to attain real-time, high-performance, deterministic operation. This section will review the real-time requirements of factory automation as seen from an open-architecture perspective.

3.1 Real-Time Measures. Factory automation reliability demands a stronger threshold of assurance than just logical program correctness. Factory automation is based on numerous internal control loops to manage the equipment. A control loop consists of a repeating cycle of operation: sample commanded setpoint, compare commanded to the sensor-measured actual setpoint, compute a goal-directed setpoint, and output setpoint to an actuator. This control loop cycle runs repeatedly at a fixed time interval, or period. All control loop timing constraints must be satisfied in order to guarantee system reliability and program correctness.

The formal definition of hard real-time is in terms of time-bounded, reliable and deterministic execution, and differs from the industrial ad-hoc notions of hard, firm, and soft real-time. The formal approach divides real-time systems into tasks with hard real-time deadlines or soft real-time deadlines. We define a factory automation task as either a process or thread, where a process is an executing instance of an application on the computing platform, and a thread is a path of execution within a process. A periodic task t_i is characterized by a worst-case computation time C_i and a period T_i and must finish by the end of its period. Tasks are independent if they do not need to synchronize with each other. A real-time system typically consists of both periodic and aperiodic tasks. The hard-real-time factory automation tasks are the control loops. The case where a task must finish by the end of its period is a hard deadline, and a system must meet all hard deadlines, otherwise the system is not correct or deterministic. For soft upper bounds, the system attempts to meet all upper bounds, but the system is still correct if some deadlines are missed.

In practice, factory automation is judged based on how fast it can perform control loops. The terms hard real-time, firm real-time, and soft real-time are ''rule-of-thumb'' timing qualifiers used to characterize the real-time performance of a system. Hard real-time is the most stringent control timing requirements, performance on the order of less that 1 millisecond resolution. Firm real-time means control timing requirements on the order of 1–10 millisecond resolution. Soft real-time means meeting control timing, at greater than 10 millisecond resolution, with the potential to run into seconds or minutes. Non-real-time refers to background tasks that can accept a laissez-faire scheduling policy with no catastrophic side effects should a deadline be missed. An important distinction should be made. The factory automation use of the hard, firm and soft real-time terms misrepresents the fact that each is in fact hard-real-time and all factory automation systems must maintain all deadlines, and instead indicate control loop timing responsiveness, and should be termed, ''hard real-time response,'' ''firm real-time response,'' and ''soft real-time response.''

To evaluate real-time correctness, tasks cannot be judged solely as independent entities, as tasks need to communicate and synchronize with each other. Communication is a major aspect of factory automation in determining real-time correctness, where synchronization is a form of communication and data is a signal. Several metrics exist to measure the effectiveness of communication. Latency is defined as the elapsed time before a message is acknowledged. Throughput is defined as the rate one process can send information to another process, especially important for systems that share large amounts of data. Factory automation demands real-time, deterministic communication so that latency, not throughput is the fundamental determinant of communication correctness. All communication latencies must meet or better an upper bound time interval. Another measure of communication correctness demands that communicating tasks do not deadlock.

Communication between tasks in factory automation ranges from IO subsystems in machine controllers to remote communications for monitoring operations on the shop floor. IO integration is a major task in developing factory automation and the trend in factory automation IO integration is towards the networking of the IO to simplify wiring, programming, and maintenance based on open, standard protocols and equipment. IO networks offer simplified cabling, which alone eliminates much of the cost and greatly simplifies troubleshooting as diagnostics can now be applied at once to all equipment on the network, instead of hunting for the broken component. Network research focuses on using statistical, algorithmic and neural network approaches for maximizing bandwidth while insuring real-time performance [4–6].

Although high-speed performance is important in a real-time control system, reliability and deterministic behavior are the ultimate system measures that cannot be sacrificed at the expense of optimizing performance. Factory automation will be considered real-time correct if and only if every hard real-time task executes deterministically bounded by fixed response time and does not deadlock. Prevention of deadlock is an important part of programming multi-threaded applications. Historically, simple OS like DOS were single threaded and could deadlock only by entering an infinite loop. Modern OS are multithreaded, and as more complex systems require priority-based synchronization mechanisms such as, semaphores, timers, events, message queues, mutex, etc., plus ability to handle priority inversion. To avoid deadlock in a multi-threaded environment, many OS kernels support either a software or hardware watchdog service to monitor operation. The watchdog operates like a countdown timer, and as long as the system is functioning properly, it will reset the timer before it reaches zero. Should the watchdog timer reach zero, appropriate action to resolve the deadlock can be taken. In summary, a system is real-time correct if and only if the following requirements are satisfied:

- processes execute deterministically within upper bound response time,
- processes do not deadlock or can mediate and resolve deadlock,
- interprocess-communication, interrupt latency, and context switching execute within upper bound response times.

Methods to improve design correctness and improve the safety and reliability of manufacturing controller operation through detection of real-time faults is an ongoing research area. The use of better modeling techniques in early system design can result in time and cost savings in system integration due to better real-time

correctness analysis and early problem identification [7–11]. Better run-time capabilities could improve operation in spite of failure by embedding into the programming languages, OS, or OS kernel the ability to detect and handle timing errors caused by variations in processing determinism due to interrupt service, caching, pipelining, and bus arbitration within the system [12–14].

3.2 RT Scheduling. Scheduling specifies how access to one or more reusable computational resources will be granted. Requirements for scheduling within factory automation equipment will be reviewed, while higher-level shop floor production scheduling will be considered outside the scope of this paper. The primary scheduling within a factory automation controller is for dispatching tasks to run on the processor(s). General-purpose commodity OS only provide time-sliced and round robin processor scheduling, which are non-deterministic algorithms and thus unacceptable for factory automation. Factory automation requires pre-emptive priority-based and/or deadline-based scheduling of tasks to guarantee real-time determinism.

Scheduling must insure stability under transient overload, such as when a system is suddenly overloaded by arrival of new events. Factory automation should be able to specify real-time control loop timing requirements with scheduling parameters such as deadlines and periods. Unfortunately, much of the real-time scheduling is still ad-hoc and manual within factory automation. Rate monotonic scheduling (RMS) is an example of a scheduling paradigm that could be used to provide deterministic real-time scheduling [15,16]. RMS would be of great help, but RSM is not widely available on many RTOS. Instead, most RTOS only support task priorities as a means of defining scheduling criteria.

To this day, many factory automation systems handle potential transient overload by using excess capacity and exhaustive testing. For these reasons, real-time systems are expensive and difficult to maintain. There is ongoing research, with tools slowly emerging, to advance the state of the art in factory automation for automating the support for execution time and schedulability analysis and support to allow independent real-time software components to determine their resource needs and to negotiate for the resources they require [17–19].

3.3 Real-Time Operating Systems. Based on the deterministic criteria, factory automation has a set of processing requirements that are different from those provided by a general purpose operating system. For instance, fairness is not an issue in real-time control. Instead, hard-real-time factory automation tasks are permanently dedicated to the hardware and are guaranteed to run at fixed periodic intervals. Thus, factory automation must run on Real-time Operations System (RTOS) that provides deterministic, high performance and complete access for processor allocation, memory management, communication, and thread scheduling. To insure that a controller or manufacturing applications operates in hard-real-time, a RTOS would be expected to be optimized to provide the following services:

- High resolution timers
- Multi-threaded and preemptive with support for thread priority and/or predictable thread scheduling such as rate-monotonic
- A system of priority inheritance must exist for the detection and prevention of priority inversion
- Support for predictable process and thread synchronization mechanisms
- Support for a virtual address space, but allows real-time threads to be locked into memory to prevent non-deterministic paging delays. Paging has a lower priority level than the real-time priority process levels so that virtual memory can still be used without interfering with any real-time processing
- Each thread preemption is based on a priority scheme, while threads at the same priority level run in a round-robin fashion with each thread provided a quantum or slice of execution time with time quantum potentially in microseconds
- Fast context switching, small interrupt latency

4 Factory Automation PC Platform Technologies

Open control for high-performance factory automation, such as a CNC, typically requires some modifications to desktop PC computing platforms. Figure 1 illustrates the three main PC-based control architectures used to handle high-performance, hard-real-time, factory automation where response times/control loop performance is under 10 milliseconds. The first architecture is software servoing based that runs 100% on the operating system, usually under a RTOS. Another architecture is still 100% PC control running a general purpose OS, but has real-time extension kernel. The third architecture entails putting the control on a separate PC co-processor board.

4.1 PC-Based Real-Time Control. PC-based hard-real-time control depends on insuring that hard-real-time tasks have higher or RT priority and that the task and data are loaded and locked into on-board physical memory, to guarantee against any delay caused by page faulting. Soft real-time tasks can have lower priorities and may be allowed to page fault depending on the timing constraints.

One of the primary considerations when using a PC that have hard-real-time threading is interaction with the related hardware and device drivers. The behavior of these devices can vary greatly, and compliance and performance must accurately be tested. Microsoft manages its hardware certification testing with its Windows Hardware Quality Labs (WHQL), which provides vendors with published specifications and standardized automated test suites for testing and validating their hardware [20]. Although stringent, these tests measure compliance and do not necessarily test performance. Because video and network cards have observable delays of 30–40 milliseconds during network stress testing, Rockwell Automation developed additional certification tests for their products to further certify performance when using the Windows NT for soft-real-time control applications [21]. Technology based on this architecture are used in place of traditional proprietary PLC-based systems for soft-real-time plant floor control.

Microsoft Windows CE is also a RTOS as well as a commodity OS product. However, its application is intended for the embedded market, such as the handheld, diskless PCs, and is not widely available on general-purpose PCs. Since there is no de facto hardware platform, it is difficult for third party factory automation providers to develop products, such as IO boards or visions sys-

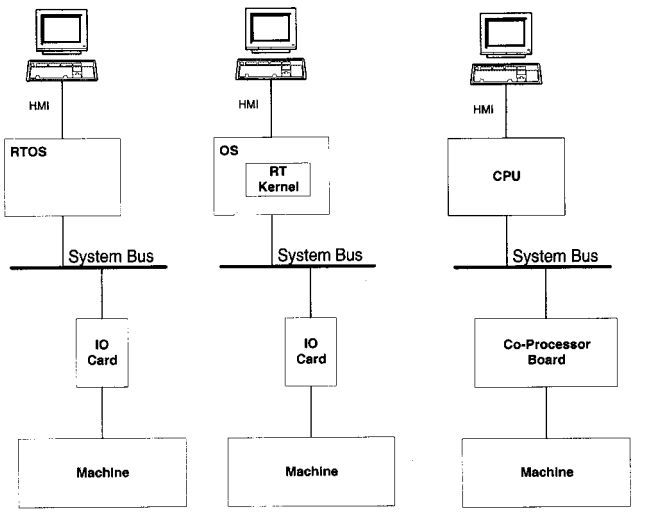

Fig. 1 Primary PC-based control system architectures

tems. Some factory automation vendors are developing PC-compatible platforms with Windows CE as the installed OS, but these are not in the commodity volumes as in the desktop industry, so the prices are not as competitive. Further, device drivers developed for mainstream Windows don't work with Windows CE, which limits the selection of peripheral equipment, as well as increases the costs.

4.2 PC-Based Real-Time OS Extensions. Both Linux and Windows NT have extensions available to make it a RTOS, usually based on a strategy of masking the fundamental PC timer interrupt. This works by modifying the OS kernel so that all calls to enable/disable interrupts and the interrupts themselves are intercepted by a real-time scheduler and interrupt handler. User RT threads are first installed as dynamically loaded kernel modules. Upon every timer interrupt, the RT scheduler runs real-time user threads according to their priority. Only when all real-time threads are idle does the scheduler pass control back to the normal OS kernel. There are problems with this approach. Many devices can only tolerate interrupts being blocked for a short amount of time before losing data. The general-purpose services and API of the primary OS are not usually available, thus eliminating the use of the greater selection of PC programming and diagnostic tools.

RT-Linux was the first such popular extension to Linux based on the timer masking to allow real-time OS features [22]. The DIAPM Real-Time Linux application Interface (RTAI) provides similar capabilities. It is based on the DIAPM Real-Time Hardware Abstraction Layer (RTHAL) [23]. KU Real-Time Linux (KURT) provides normal Linux user process with better timing resolution and control over scheduling than the standard Linux kernel. Although KURT is not strictly deterministic, it may be a better choice for ''firm real-time'' applications [24]. The NIST Enhanced Machine Controller (EMC) is an example of an open-architecture CNC controller exploiting real-time Linux, open source, and community software development [25]. When running actual equipment, real-time Linux is used to achieve the deterministic hard-real-time computation response rates required (200 microseconds is typical).

Windows NT is suitable for firm-real-time control, but interrupt service for some devices can be protracted, and jitter can occur within the running control system. Windows NT can run a real-time kernel like TenAsys INTime or VenturCom RTX or Hyperkernel from Imagination Systems, Inc. These systems add a real-time kernel that intercepts the interrupt from the PC clock before the Windows operating system receives it for scheduling real-time tasks local to the kernel. TenAsys INtime provides a memory protection scheme between its real-time kernel and the Windows NT operating system kernel. Proprietary real-time kernel extensions are also used to streamline system development by allowing PC-hosting of real-time targets. An example of this type of real-time PC development system is Mathwork's Matlab running Real-Time Workshop or xPC.

4.3 Co-Processor Board. This design alternative uses a traditional PC and couples it with a coprocessor board in the same platform backplane. The platform runs a typical window-based Operating System to provide the controller a human-machine-interface that can potentially offer other typical desktop processing chores running in the background or when the machine controller is not operating. This yields benefits of real-time control while using the database, communications, and HMI capability of the PC. The use of bus-mastering coprocessor boards are popular because they provide extremely high-speed communication control across the bus transparently to the operation of the general-purpose host operating system. The use of this design provides an instant ''firewall'' against secondary impairment from the host to insure real-time controller operation on the coprocessor board. Numerous coprocessor boards are available in the market, such as, Delta Tau, Aerotech, LabView products from National Instruments, or Galil products. The main disadvantage to this approach is the increased cost for the additional hardware.

The coprocessor board does not have to be an actual motion control product, but instead can be a full or half-sized industrial single board computers that is loaded with an embedded real-time operating system. Examples commercial and research RTOS suitable for embedded coprocessor platforms include: VxWorks from WindRiver Systems Inc.; LabView RT from National Instruments; Embedix RealTime, a hard real-time Linux OS; DR-DOS a real-time DOS OS also from Lineo; QNX from Software Systems Ltd.; Chimera [26]; and Microsoft Windows CE; and Microsoft Windows Embedded NT with real-time extensions products including: INtime from TenAsys Corp., RTX from VenturCom, Inc., Hyperkernel from Imagination Systems, Inc.

5 Real-Time Programming of Manufacturing Systems

An open programming language environment is critical for rapid integration of mechatronic devices that include stepper motors, DC and AC motors, laser and vision sensors, force and torque sensors, tactile sensors, pneumatic grippers, etc. These mechatronic devices are essential components of many manufacturing systems such as robot manipulators, CNC machines, and automatic assembly systems. For example, according to its pre-programmed software implementations, a robot can make intelligent decisions and actions based on external sensory information. Experience indicates that a large portion of a typical robot application program is non-motion related [27]. Programs deal mainly with initialization, communication, synchronization, sensory information processing, and error checking and correction. Much of the intelligence of a manufacturing system is derived from these sophisticated software implementations. Requirements for programming of manufacturing systems such as robot manipulators, CNC machines, and measurement machines are in general much higher than those programming in desktop computers because of real-time and special constraints with machinery. Some important features for programming of manufacturing systems are listed in Table 2.

Research and application experience of industrial automation and manufacturing systems indicate that an ideal programming language for manufacturing systems must be a sophisticated computer programming language. The language should appeal to both sophisticated expert programmers and novice users. In most cases, sophisticated users will write high-level functions that can be readily used by less experienced users. The language must be deterministic for real-time programming. The language should be interpretive with a quick system response. The program compilation presents a serious problem for real-time manipulation of mechatronic systems. For a real-time manufacturing system, the

Table 2 Important features in languages for programming of manufacturing systems

Deterministic for real-time computing
Interpretive
Interactive user interface
Ease interface with binary objects
Concurrent programming
Superset of an established language
Based-on an open language with international standard
Object-oriented
Graphic user interface
Supported in heterogeneous platforms
Code portable across-different platforms
Secure network computing
Advanced numerical computing with matrix library

external environment may be different at each execution, so the testing scenario may not be repeatable. During debugging and testing, it is impractical to restart a program from the very beginning every time a change is made or a problem is diagnosed. The programming environment should support command or function execution interactively. Although the language is interpretive, some time critical code such as control algorithms for servo loop might be compiled for fast execution. Therefore, interpretive scripts should be able to interface to binary objects. It is typical that multi-tasks or threads are executed for servo update, IO checking, user interface, etc. The language should support concurrent multi-task processing. In addition, the language should be a superset of an established computer programming language rather than a subset. Programming of manufacturing systems can then draw upon a large body of existing user and code base. The base language should be an open language with an international standard so that it will keep abreast advance of the new technology. The language should be easy to learn. It will be used not only for programming of manufacturing systems, but also for daily programming tasks. The object-oriented nature of the language will ensure that the code is relatively easier to develop, maintain, and reuse. Programming of manufacturing systems should be similar to very high-level shell programming. Application programs are created not by writing large programs starting from scratch. Instead, they are combined by relatively small components. These components are small and concentrate on simple tasks so that they are easy to build, understand, describe and maintain. The language should support modular programming. A set of high-level commands and functions developed by experienced researchers and engineers then can be readily used by novice users and factory personnel. The programming environment should support user-friendly graphical interface and visual programming as illustrated in LabView. However, an entirely visual based programming environment without a base of procedural programming language is difficult to program for complicated manufacturing tasks, especially for sensor fusion. The language should be supported in different platforms such as Windows and Linux so that application programs will be portable. The language should be manufacturing-system-independent so that high-level application programs can be developed to relieve programmers of the task of learning different programming languages for different manufacturing systems. It will enable integration of mechatronic devices from different vendors for consistent interfaces external to machinery. An integrated manufacturing system often consists of several subsystems such as a workcell with multiple robot manipulators. The language should support programming of several manufacturing systems concurrently. To ease integration with a manufacturing executive system and other information systems, manufacturing systems are often networked. The language should support secure network computing with industrial standard networking protocol TCP/IP. It is desirable that a program can be dynamically downloaded through the network and executed securely so that the manufacturing system can be adaptive to the external sensory information. Many complicated algorithms are used in manufacturing systems, so it is desirable that the language supports advanced numerical features such as matrix computations similar to matrix features in Matlab. It is also desirable to be able to perform function block-based programming. IEC 61131-3 is visual, function block-based, programming standard for industrial control. In this paradigm, function blocks are the software equivalent of integrated circuits that are wired together to build control systems [28].

Development of a language environment for real-time operation of manufacturing systems takes a deliberate effort. General-purpose compiler-based system languages such as C and C++ are not suitable for this purpose. The Java programming language is more portable because a machine independent intermediate bytecode is generated from a Java program. The bytecode then can be executed by a Java virtual machine. However, garbage collection is typically non-deterministic. The early version of Java with unrestricted garbage collection presents a serious problem in real-time computing for applications in manufacturing systems. Study of memory-allocation behavior of embedded applications written in Java is still an active research topic [29]. General-purpose interpretive languages such as Perl, Python, Tcl/TK are also not desirable for programming of manufacturing systems because of their lack of interactive user interface. The code written in these glue languages are difficult to read and maintain.

Most programming languages for manufacturing systems currently in use are associated with proprietary commercial manufacturing products with specific hardware configurations. For example, like the pre era of Unix/C computing environment, every robot has its own programming system or language with different notations and symbols. Integration of different robots in the same production line is difficult and the training of personnel is costly. No single robot-independent language has prevailed as a de facto standard for robot programming. In general, the evolution of robot programming languages follows the evolution of general-purpose computer programming languages. Robot languages based on Algol, APL, Lisp, Basic, Fortran computer languages can be found in the 1970s and early 1980s. Robot languages developed in the 1980s are primarily based on Pascal that was popular then. In the late 1980s and 1990s, C, C++, and Ada are the prevalent computer languages. Although most robotic systems are now coded in C, C++, or Ada, no existing robotic system behaves like C, C++, or Ada from the user's point of view. The RCCL (Robot Control C Library) is such an example [30]. In this system, the robot motion-related C library is linked when an executable object code is generated in a Unix environment. Open-architecture robot controllers and standard interfaces for sensors and actuators are recognized as critical research issues for emerging applications of robotics and intelligent machines [31]. The popular commercial robot languages include KAREL, V++, VAL II, UNIVAL, V++. CNC high-level languages are APT or a CAD representation that is post-processed into machine-executable languages, including RS274D or BCL. The prevalent CMM language is DMIS. For programming of manufacturing systems, a Ch language environment has been developed and used to control a manufacturing workcell with two industrial robot manipulators under real-time LynxOS [32–34]. Ch is a superset of C interpreter with all features of C90 as well as new features in C99, objected-oriented features in C++, and salient features from other programming languages and software packages.

6 Real-Time Component Technologies

Since its inception, the first and easiest step to achieving an open-architecture was to provide open access to controller components, which is now a widely-established paradigm available in most commercial controllers. The next step in the open-architecture technology evolution is towards a ''plug-and-play'' component-based model. A component is a self-contained unit of software, possibly combined with hardware, which provides well-defined functionality and well-defined connections or interfaces exposed for communication. Components are designed for repeated use in developing systems, either with or without customization. By reusing existing components, applications can be produced more quickly, thus reducing the time to market. As product lifecycles continue to shorten, component-based development has the potential to help improve productivity and reduce software costs.

To be effective, components must provide the following features:

- Location independence so that components can transparently run in the same process, same platform, or across a distributed network.
- Platform independence so that components interact as if there were a single type of platform.

- Programming language independence so that components can interact regardless of the component implementation language
- Ease of use for deployment and customization
- Support for multiple languages and environments
- Less-skilled programmers must be able to assemble applications
- Tools must be available to catalog the various components and for composibility to assemble and test new applications built from components
- Interchangeable to scale performance, depending on the application

There exist several component-based controller development products, e.g., ControlShell from Real-Time Innovations, and LabView from National Instruments, that achieve many of the desired goals of component-based technology. Unfortunately, these products are proprietary and the resulting components are not interoperable so interchangeable ''plug-and-play'' is impossible [35]. Although neither component model was originally intended to be used in the real-time machine control, both are evolving into this application domain. Because of the enormity of the COM market-presence, many controller products, such as LabView, will accept COM components within some aspect of their development environment.

COM is the Microsoft architecture for local interaction of components and the Distributed Common Object Model (DCOM) provides the methods for remote interaction of components. COM has evolved from its origins as a mechanism to let applications dynamically perform Object Linking and Embedding (OLE). COM provides many services to facilitate component technology including, location transparency, security, registry, naming, and type information are included in any 32 bit Windows operating systems, at no additional charge. The primary benefit to COM is zero sacrifice on performance for within-process component interaction. For integration of factory automation with the business enterprise, COM+ adds the Microsoft Transaction Server (MTS) that provides services for transactional guarantees, concurrency control, instance lifecycle management, database session management, and security. DCOM is the basis for distributed communication, and is designed for use with multiple network transports, including TCP, UDP and HTTP. DCOM is based on Open Software Foundation's DCE-RPC specification, and is not real-time. OPC has been the leader in the realization of COM-based open-system manufacturing communication products. Wind River's VxDCOM provides a third-party implementation for developing real-time DCOM applications.

CORBA is a specification from the Object Management Group (OMG) that defines a software bus, the Object Request Broker (ORB), for platform-independent and language-independent communication and execution between software objects. CORBA was designed by the OMG consortium to specifically to handle the problem of integrating distributed applications running on a mix of operating systems running on mainframes, workstations and PC desktop machines. CORBA was not originally intended for real-time applications, but can be used in soft-real-time factory automation with certain caveats [36]. The Real-Time CORBA specification was developed to overcome the non-deterministic behavior of CORBA and provide real-time functionality [37]. This standard tackles the key issue of end-to-end predictability across a CORBA system, and provides its solution in terms of priority control, synchronization, protocol selection and other forms of resource control [38]. Examples of Real Time CORBA products include HARDPack from Lockheed-Martin, ORBExpress, Vertl/Expersoft Orb, e*ORB, and the Highlander port of VisiBroker to an RTOS.

7 Summary

The next generation of manufacturing systems must be agile to better adapt to changing markets and evolving technologies, while still focusing on cost and quality. Open architecture control systems are a key enabling technology for realizing agile manufacturing through increased flexibility and reduced automation life cycle costs. The open architecture ability to reconfigure or extend existing equipment to meet new needs is particularly powerful in meeting these challenges. Life cycle costs, including purchasing, integration, customization, training, maintenance, diagnostics and repair, will be positively affected by the competitive pressure that arises when proprietary barriers are eliminated and interoperability standards are introduced. This paper reviewed a wide set of real-time open-architectures and component-based tools and technologies that are on the forefront of agile manufacturing.

On the horizon, there remain important business issues and technical problems to be solved before open architecture manufacturing systems are to become a prevalent technology. Technical problems include achieving deterministic execution of critical control tasks while leveraging desktop computing hardware, operating systems, and software not ideally suited to this purpose; attaining high levels of reliability when moving from a monolithic model to one in which system integration plays a major role; and increasing the communication efficiency that impairs the design of truly distributed real-time systems. There are several significant business obstacles. Product differentiation becomes more difficult with the loss of unique (albeit proprietary) interfaces. Skill sets need to shift away from in-house development of soup-to-nuts turnkey systems to systems integration, where software engineering becomes more important. Perhaps most significant is the vendor's loss of control over deployed products, whose original nameplates now cover untested combinations of aftermarket catalog components. Who is responsible for the injury or death of machine operators when something goes wrong? To meet these challenges, industry must adopt new practices when dealing with open architecture controllers. Conformance and testing are critical to successful operation. There is a need for public conformance and test suites to be developed for which users and providers of open architecture technology could test products against the standard open architecture specifications, and hence ensure product fidelity as well as interoperability.

Today, the evolution of open architecture controllers and its application in manufacturing is still in its infancy. However, the advent of PC-based open architecture control has already produced opportunities and exposed latent markets that previously went unfulfilled. One market that has benefitted is small manufacturing enterprises (SMEs), companies with fewer than 500 employees. Most of these companies are quite small, employing less than 30 people. These companies have traditionally been left out of the CNC user market due to high costs of entry. The impact of this technology insertion is expected to be significant. In the U. S. alone, there are 381,000 SMEs, which employ 12 million people (2/3 of the manufacturing force) and account for half of the total U. S. manufacturing value [39]. Some of these small manufacturers have begun open-source software projects to develop free programs for CAD, CAM, and CNC [40]. These groups have set up Internet sites where developers can collaborate and post new contributions. Software is released to the user community through development snapshots, and users are encouraged to submit new material and requests for features or improvements. Aside from the obvious advantage of free software, advantages include the ability to customize the software for specific needs, or port it to new platforms. Disadvantages include the higher degree of ownership users must accept, due to the explicit lack of warranties, and the developmental nature of the work.

Ultimately, open-architecture technology is but a first step on the path to new and more intelligent manufacturing control. Research into the use of smarter cooperative components, or agents, is being applied in a variety of architectures to design and implement distributed intelligent manufacturing systems [41]. Much research remains to resolve key issues related to component-based open-architecture systems, such as system configuration and inte-

gration, communications, legacy issues and standardized interfaces. However, the prospect for these controllers based on derivatives of component-based open-architecture technology is especially promising, offering unlimited potential for intelligent controllers that are self-configuring, learning and self-organizing.

References

[1] Proctor, F., and Albus, J., 1997, ''Open Architecture Controllers,'' IEEE Spectr., **34**, No. 6, pp. 60–64.

[2] Wright, P. K., and Greenfeld, I., 1990, Open Architecture Manufacturing: The Impact of Open-System Computers on Self-Sustaining Machinery and the Machine Tool Industry, in *Proc. Manufacturing International '90*, **2**, pp. 41–47.

[3] Wright, P. K., 1995, ''Principles of Open-Architecture Manufacturing,'' J. Manufac. Syst., **14**, No. 2, pp. 187–202.

[4] Hong, S. H., and Kim, W. H., 2000, ''Bandwidth Allocation Scheme in CAN Protocol,'' IEE Proc.—Control Theory Appl., **147**, No. 1, pp. 37–44.

[5] Zuberi, K. M., and Shin, K. G., 1997, ''Scheduling Messages on Controller Area Network for Real-Time CIM Applications,'' IEEE Trans. Rob. Autom., **13**, No. 2, pp. 310–316.

[6] Cavalieri, S., and Mirabella, O., 1996, ''Neural Networks for Process Scheduling in Real-Time Communication Systems,'' IEEE Trans. Neural Netw., **7**, No. 5, pp. 1272–1285.

[7] Lin, E. Y.-T., and Zhou, C., 1999, ''Modeling and Analysis of Message Passing in Distributed Manufacturing Systems,'' IEEE Trans. Syst. Man Cybern., **29**, No. 2, pp. 250–262.

[8] Stoyenko, A. D., Marlowe, T. J., and Laplante, P. A., 1996, ''A Description Language for Engineering of Complex Real-Time Systems,'' Real-Time Syst., **11**, No. 3, pp. 245–263.

[9] Fidge, C., Kearney, P., and Utting, M., 1997, ''A Formal Method for Building Concurrent Real-Time Software,'' IEEE Software, **14**, No. 2, pp. 99–106.

[10] Ancilotti, P., Buttazzo, G., Di Natale, M., and Spuri, M., 1998, ''Design and Programming Tools for Time Critical Applications,'' Real-Time Syst., **14**, No. 3, pp. 251–267.

[11] Bradley, S., Henderson, W., Kendall, D., and Robson, A., 1994, Designing and Implementing Correct Real-Time Systems, in H. Langmaack, W-P. de Roever, and J. Vytopil, Eds. *Formal Techniques in Real-Time and Fault-Tolerant Systems FTRTFT '94, Lubeck, Lecture Notes in Computer Science 863*, pp. 228–246, Springer-Verlag.

[12] Stewart, D. B., and Khosla, P. K., 1997, ''Mechanisms for Detecting and Handling Timing Errors,'' Commun. ACM, **40**, No. 1, pp. 87–94.

[13] Kenny, K., and Lin, K-J., 1991, ''Building Flexible Real-Time Systems Using the Flex Language,'' IEEE Computer, **24**, No. 5, pp. 70–78.

[14] Kligerman, E., and Stoyenko, A. D., 1986, ''Real-time Euclid: A Language for Reliable Real-Time Systems,'' IEEE Trans. Software Eng., **12**, No. 9, pp. 941–949.

[15] Liu, C. L., and Layland, J. W., 1973, ''Scheduling Algorithms for Multiprogramming in a Hard Real Time Environment,'' JACM, **20**, No. 1, pp. 46–61.

[16] Sha, L., Rajkumar, R., and Sathaye, S., ''Generalized Rate-Monotonic Scheduling Theory: A Framework for Developing Real-Time Systems,'' Proc. IEEE, **82**, No. 1, pp. 68–82.

[17] DiNatale, M., and Stankovic, J., 1994, ''Dynamic End-to-End Guarantees in Dist. Real-Time Systems,''in *Proc. 15th IEEE Real-Time Syst. Symposium*, pp. 216–227.

[18] Garcia-Fornes, A., Terrasa, A., Botti, V., and Crespo, A., 1997, ''Engineering Tool for Building Hard Predictable Real-Time Intelligent Systems,'' J. Eng. Appl. Artif. Intell., **14**, pp. 369–377.

[19] Deng, Z., Liu, J. W.-S., Zhang, L., Seri, M., and Frei, A., 1999, ''An Open Environment for Real-Time Applications,'' Real-Time Syst. J., **16**, No. 2/3, pp. 155–185.

[20] Microsoft. *Windows Hardware Quality Labs*, http://www.microsfot.com/hwtest/default.asp.

[21] Rockwell Automation—Allen Bradley, 1998, *Using the Windows NT Operating System for Soft Reqal-Time Control-Separating Fact from Fiction*, White Paper.

[22] REALTIMELINUX.ORG., *The Real time Linux Portal*, 2000. URL: http://www.realtimelinux.org.

[23] Mantegazza, P., Bianchi, E., and Dozio, *DIAPM RTAI*, 2000. URL: http://www.aero.polimi.it/projects/rtai/.

[24] Hill, R., Srinivasan, B., Pather, S., and Niehausc, D., 1998, *Temporal Resolution and Real-Time Extensions to Linux*. URL: http:///www.ittc.ukans.edu/kurt/.

[25] Proctor, F., 2000, *The Enhanced Machine Controller*, URL: http://www.isd.mel.nist.gov/projects/emc/emc.html.

[26] Stewart, D., Schmitz, D., and Khosla, P., 1992, ''The Chimera II Real-Time Operating System for Advanced Sensor-Based Control Applications,'' IEEE Trans. Syst. Man Cybern., **22**, No. 6, pp. 1282–1295.

[27] Cheng, H. H., and Penkar, R., 1995, ''Stacking Irregular-Sized Packages by a Robot Manipulator,'' IEEE Robotics and Automation Magazine, **2**, No. 4, pp. 12–20.

[28] International Electrical Commission, IEC, Geneva, 1993, *IEC 1131-3, Programmable Controllers—Part 3 Programming Languages*.

[29] Petit-Bianco, A., 1998, ''Java Garbage Collection for Real-Time Systems,'' Dr. Dobb's Journal, No. 290, pp. 20–29.

[30] Hayward, V., and Paul, R., 1986, ''Robot Manipulator Control Under Unix RCCL: A Robot Control ''C'' Library,'' Int. J. Robot. Res. **5**, No. 4, pp. 94–111.

[31] Bekey, A. G., 1997, ''Needs for Robotics in Emerging Applications: A Research Agenda,'' IEEE Robot. Autom. Mag., **4**, No. 4, pp. 12–14.

[32] Cheng, H. H., 1996, ''Plug-and-Play Open Architecture Integration of Mechatronic Systems for Agile Manufacturing,'' in *Proceedings Nov. 20–21, SPIE, Open Architecture Control Systems and Standards*, **2912**, pp. 136–145, Boston, MA.

[33] Cheng, H. H., and Hu, X., 2000, ''Plug-and-Play Open-Architecture Object-Oriented Real-Time Mechatronic System Integration and its Applications in an Automatic Manufacturing Workcell,,'' in *Proc. of NSF Design and Manufacturing Grantees Conference*, Vancouver, Canada.

[34] Cheng, H. H., 2000, *The CH Language Environment*, URL: http://iel.ucdavis.edu/CH.

[35] Chung, E., Huang, Y., Yajnik, S., Liang, Deron, Shih, C., Wang, C.-Y., and Wang, Y.-M., 1998, ''DCOM and CORBA Side by Side, Step by Step, and Layer by Layer,'' C++ Report, **10**, No. 1, pp. 18–30.

[36] Polze, A., Plakosh, D., and Wallnau, K. C., 1998, ''CORBA in Real-Time Settings: A Problem from the Manufacturing Domain,'' in *First International Symposium on Object-Oriented Real-Time Distributed Computing*.

[37] Object Management Group, 1999, *Real-Time CORBA 1.0 Specification*, ORBOS/99-02, and errata, ORBOS/99-03-29.

[38] Schmidt, D., and Kuhns, F., 2000, ''An Overview of the Real-Time CORBA Specification,'' IEEE Computer., **33**, No. 6, pp. 56–63.

[39] Manufacturing Engineering Partnership (MEDP), www.mep.nist.gov.

[40] Shackleford, W., and Proctor, F., 2000, ''Use of Open Source Distribution for a Machine Tool Controller, in *Proceedings of the SPIE Conference on Sensors and Controls for Intelligent Machining*, **4191**.

[41] Shen, W., and Norrie, D., 1999, ''Agent-Based Systems for Intelligent Manufacturing: A State-of-the-Art Survey,'' Int. J. Know. Infor. Syst., **1**, No. 2, pp. 129–156.