PROCEEDINGS

IASTED

COMPUTER APPLICATIONS IN INDUSTRY

Cairo, Egypt
February 1-3, 1988

EDITORS: I. Lefkowitz, M.F. Sakr

A Publication of
The International Association of Science
and Technology for Development - IASTED

ACTA PRESS

ANAHEIM * CALGARY * ZURICH

TABLE OF CONTENTS

FOREWORD

This conference on computer applications in industry is one of a series of international conferences on scientific and technical subjects sponsored by the International Association of Science and Technology for Development (IASTED).

The main purpose of IASTED is to further economic development by promoting science and technology with particular attention given to the less developed areas of the world.

The Conference on Computer Applications in Industry was held in Cairo, Egypt, during the period February 1-3, 1988. Hosts for the Conference were the University of Cairo and the Development Research and Technological Planning Center.

There were 37 papers presented covering a broad spectrum of theory, methodology, techniques, and applications representing the very rapidly growing ubiquitousness of the computer in industry. In addition to the large Egyptian contingent, the conference attracted participants from a number of countries, including England, Italy, Jordan, Qatar, Spain, U.S.A., and Yugoslavia.

The history of computer applications in industry is as old as the history of the computer itself. Progress during the past decade has been very rapid. The continuing advances in computer hardware and software technology, e.g., increases in computer speed, memory capacity, and sophistication coupled with ever-decreasing costs assure that this progress will continue unabated in the foreseeable future.

The application of computers in industry is very important because of its direct impact on production and productivity. One area of application is computer control, which is now firmly established in a large variety of industries throughout the world. Although the major developments have taken place in highly industrialized economies, the contributions of advanced control to improving product quality, increasing energy efficiency, decreasing losses, etc. should be of particular relevance to developing economies. Indeed, the state-of-the-art of computer control is such that plans for new industrial plants invariably call for some level of computerization with integrated plant control as an ultim[illegible]. Although the computerization of an old plant is generally far more difficult than designing a new co[illegible]ed plant, the information and need to do so is there, especially in developing countries.

The conference covered a number of other aspects of com[illegible]uter applications in industry including computer-aided design, artificial intelligence, modeling and simu[illegible]ation, etc. All are important components in the pursuit of improved processing efficiency, product quality, [illegible]nd productivity.

The papers in this *Proceedings* are organized in the following categories:

Invited Paper: State-of-the-Art Survey
Software
Artificial Intelligence-Based Systems
Computer-Aided Management
Computer-Aided Design
Computer Process Control
Control Systems Analysis and Design
Power Systems Applications
Modeling and Simulation
Measurements
Testing and Fault Detection

Irving Lefkowitz
M. Fahim Sakr
Conference Co-Chairmen

State of the Art of Computer Control Applications in Industry

Irving Lefkowitz
Case Western Reserve University
Cleveland, Ohio, U.S.A.

Abstract

The state of the art of computer control applications in industry is examined in the context of the goal of maximizing overall plant performance via an integrated systems control approach. Practical realization of systems integration depends on concepts of hierarchical structuring of the decision-making/control problem and of distributed, computer-based control. Recent developments in these areas, along with contributing advances in sensor technology, advanced control methods, process diagnostics, and expert systems methodology are reviewed. Finally, some thoughts are expressed concerning technology transfer and the relevance of the above developments to industries in developing countries.

Introduction

The past decade has seen a very rapid development of computer control applications to industrial systems – in oil, chemical, steel, paper, and electric power – to cite a few prominent examples [3, 6, 16].

There are three fundamental issues driving this development.

1. Industry (and government) is becoming increasingly aware of the critical importance of improving the performance of their processes and plants. The major factors underlying this awareness are the following: (a) the need for more efficient utilization of resources (e.g., energy, water, labor, materials) because of increasing cost, limited availability, or both, (b) demands for higher productivity to meet more intense competition from both domestic and foreign sources, and (c) more stringent requirements concerning product quality, environmental impact, and human safety because of government regulations and greater consumer awareness.

2. Industrial systems are typically large-scale and very complex, characterized by multivariable, nonlinear and time-varying relationships, and subject to disturbances and constraints of various kinds. Performance of these systems depends on a variety of factors, including (a) technological design of the system and its components, (b) the nature of resources available and environmental constraints, and (c) the choice of operating conditions, allocation of resources, scheduling of operating sequences, etc. This has two consequences. First, recognition that the concept of control must be broadened to embrace, not only the traditional process control functions, but also real-time applications of information processing and decision making, e.g., production planning, scheduling, optimization, operations control, etc. Second, recognition that the control objectives can only be achieved by sophisticated computer controls employing advanced systems concepts and techniques.

3. Developments in computer technology over the past fifteen years have resulted in tremendous reductions in hardware costs accompanied by dramatic increases in computation speeds and storage capacities. User-oriented programming languages have greatly eased the man-machine interaction problem, e.g., in programming, debugging, and updating computer control algorithms. Also, system reliability has improved substantially as a result of more reliable components and the increased feasibility of fault-tolerant designs, redundancy, and diagnostic routines – enhanced by low hardware costs and more sophisticated design techniques. As a result, the means for achieving high performance through advanced computer control are not only currently available, but they are becoming increasingly more powerful and cost effective.

Integrated Systems Control

A consequence of the explosive advances in computer technology has been a vast broadening of the domain of what is technologically and economically feasible to achieve in the application of computers to control of industrial systems. The ability of the computer system to gather, process, and store large quantities of data, to carry out complex computational tasks at high speeds, to interact effectively with the human component of decision-making functions, and to adapt readily (via software) to changing system requirements has meant that all aspects of information processing, data gathering, process control, on-line optimization, operations control – even real-time scheduling and production planning functions – can be included in the range of tasks to be carried out by the computer control system. This has made possible the realization of integrated systems control in which all factors influencing plant performance (including the couplings, interactions, and complex feedback paths existing in the system) are taken into account in an integrated fashion to achieve an overall optimum performance [7, 9].

Early implementations of the integrated system control approach were by the Japanese steel industry. Results were spectacularly successful with the following benefits claimed [6]:

- Improved efficiency of operating units and increased plant productivity as a result of better quality and the ability to control to optimum conditions.
- Better utilization of resources, e.g., energy, scarce materials, manpower.
- More effective compliance with technological and environmental constraints, e.g., ensuring that air and water effluents meet government standards.
- Adaptability to time-varying conditions such as those induced by changing product demands, costs, availability of raw materials, equipment obsolescence, etc.
- Capability of responding safely and securely to contingency events, e.g., equipment breakdown, delayed delivery of needed supplies, etc.
- Capability of providing effective and immediate updates on the status of the system, e.g., orders in process, inventories, equipment maintenance, etc.

Hierarchical Control

The problems of realization of implementation of an integrated system control are generally formidable because of the complexity of the production processes, the variety of constraints to be satisfied, the nonlinear, time-varying dynamics, etc. The hierarchical control approach provides rational and systematic procedures for resolving these problems. In effect, the overall systems control problem is decomposed into more easily handled subproblems; the subproblem solutions are coordinated by a higher level controller so as to assure compliance with overall objectives and constraints.

Two conceptual approaches to the control hierarchy have been formulated; these are identified as the multilayer control hierarchy (vertical decomposition) and the multilevel control hierarchy (horizontal decomposition).

In the *multilayer control* approach [7, 8, 9], four classes of control functions are identified, namely: direct, supervisory, adaptive, and self-organizing control functions (see Fig. 1).

(a) The first or direct-control layer constitutes the interface between the controlled plant and the decision-making and control aspects of the system. It includes three subfunctions: data acquisition, event monitoring, and direct control.

(b) The second-layer or supervisory function is concerned with the problem of defining the immediate target or task to be implemented by the first layer. In the normal mode, the objective may be control of the plant for optimum performance according to the assumed mathematical model. Under emergency conditions, different objectives may take precedence through implementation of appropriate contingency plans.

c) The third-layer or adaptive function is concerned with updating the algorithms employed at the first and second layers, reflecting current operating experience.

(d) The fourth-layer or self-organizing function is concerned with decisions relevant to the choice of structure of the algorithms associated with the lower layers of the hierarchy. These decisions are based on overall considerations of performance objectives, priorities, assumptions of the nature of the system relationships and input patterns, structuring of the control system, coordination with other systems, etc.

In the *multilevel control* approach [10, 12, 13], the overall plant system is decomposed into subsystems, each with its own local controller. In this scheme:

(a) The first-level controllers compensate for local effects of the disturbances, e.g.,maintain local performance close to the optimum while ensuring that local constraints are not violated.

(b) The second-level controller modifies the criteria and/or the constraints for the first-level controllers in response to changing system requirements so that actions of the local controllers are consistent with the overall objectives of the system.

In effect, the subsystem problems are solved at the first level of control. However, since the subsystems are coupled and interacting, these solutions have no meaning unless the interaction constraints are simultaneously satisfied. This is the coordination problem that is solved at the second level of the hierarchy. A schematic of a two-level multilevel structure is shown in Fig. 2.

A hybrid of the two hierarchical approaches is represented by the structure of Fig. 3, developed by the group at Purdue University, USA [15, 16]. In this formulation, there is explicit identification of man-machine interfaces, inter-computer communication links, and the management information function. This formulation has served as a blueprint for implementations of computer control in the steel industry and in the pulp and paper industry.

Microprocessor-based distributed control

Distributed systems are now generally accepted state-of-the-art for the process industries and, increasingly, for the manufacturing industries also. Indeed, all of the major vendors of control systems feature distributed, microprocessor-based systems as key components of their product line (e.g, Foxboro SPECTRUM, Honeywell TDC 3000, Bailey Network 90, Siemens SIMATAC, and Toshiba TOSDIC distributed control systems). A generic schematic of a distributed system is shown in Fig. 4.

The important common features of these systems are [2, 16]:

- Modular, building block design based on microprocessor components; this contributes to design flexibility and easy modification in response to changing needs and conditions.
- Operator interface featuring color graphics and a centralized display and command capability with complete access to all aspects of plant operation ranging from details at the individual loop level to overall plant status.
- Preprogrammed menu type instruction system that provides configurability capabilities and the ability to select from a large variety of prepackaged control algorithms.
- A high-speed data highway to provide communication between process units and microprocessor controllers, among controller units, and with larger (main frame) computers for supervisory and higher-level information processing functions.
- Provisions for high reliability through use of redundant components, fail-safe techniques, and self-diagnostic schemes.

Commercial systems are still in process of evolution. Some current trends are [16]:

- More extensive incorporation of relay and logic functions with the blending of programmable logic controller (PLC) technology with traditional process control technology.
- Expansion of the communications network to embrace a global data base covering all process variables in the plant. This greatly facilitates data collection for use at the upper levels of the plant control hierarchy (i.e., for satisfying management information and plant integration functions).
- Incorporation of general purpose microcomputers (e.g., the IBM PC) into the system to perform special, tailored functions and tasks.

The overall consequences of using these modern distributed control systems are (a) to simplify and speed up the design and implementation of a new control system, (b) to facilitate plant operations and system maintenance, (c) to give the operator far more command over the system through much more effective man-machine interfaces, and (d) to greatly increase system reliability.

Process Diagnostics

A consequence of the trend toward computer control and systems integration is that the system under control becomes increasingly more complex and large-scale, thus becoming increasingly vulnerable to the effects of contingency events, e.g., abnormal changes in process conditions, a fault in a control computer or communication link, a sensor failure, etc. Hence, it becomes increasingly imperative that the system be designed to be reliable and robust; in particular, that the occurrence of a contingency event not have a catastrophic effect on the system. Traditional approaches to the problem include the use of ultra reliable components, redundancy of critical system elements, and conservative design features. As we move toward whole plant integration, such approaches become increasingly costly and inadequate. It is necessary, therefore, to build into the system active mechanisms for assuring reliable, fail-safe operation or, at worst, graceful degradation in the event of a malfunction or other abnormality. Process diagnostics provides the means for detecting fault and failure occurrences and identifying the source so that the operator can take remedial action. An obvious extension in this direction is where the computer implements the necessary corrective actions automatically.

There are important applications of process diagnostics in the electric power industry and the chemical industry; the approach is perhaps most advanced in nuclear power plant applications [11].

Sensor Development

It is axiomatic that the ability to control is limited by the quality of the information available to the control system. A large variety of sensors are commercially available for on-line measurement of the common physical variables (pressure, flowrate, temperature, speed, etc.) encountered in industrial processes. However, sensors for measuring chemical composition, product quality, and many other variables are very restricted with respect to cost, accuracy, speed of response and applicability in the process environment.

The development of improved sensors has been a continuing activity engaged in by both vendor and user companies. We mention three directions of effort that characterize recent advances in sensor technology:

Application of newly discovered physical principles or properties. An example here is the potential development of measuring devices for chemical concentration based on ion-sensitive solid state elements.

Smart sensors. This couples a micro-processor with one or more sensing elements for the purpose of (a) computing the value of a process variable, which is not directly measurable, as a function of other variables which are measurable; (b) improving the accuracy of a measurement by correcting for ambient conditions, e.g., temperature variations; (c) performing smoothing, filtering, linearizing, discretizing, and other functions on the output data; (d) carrying out automatic calibration and self-checking procedures to improve accuracy and reliability.

Measurement systems. This may be looked upon as an extension of the smart sensor, but on a much larger scale involving computers, multiple sensors, servos for accurate positioning of the sensors, etc. This kind of system is being increasingly used in product quality measurement and control, e.g., devices which generate two-dimensional profiles of thickness and/or moisture content in paper production.

Advanced Control Techniques

As noted earlier, one of the important contributions of computer-based control is the ease with which special-purpose and advanced control algorithms may be implemented (i.e., via software rather than hardware changes). Some approaches that are finding increasing industrial application are:

Internal model-based control (IMC). This approach employs a dynamic model of the controlled process to predict the response of the system to a control action and thereby provide the means of selecting the best action to satisfy objectives and constraints. Variants of this approach, including dynamic matrix control and inferential control, are being applied in the chemical and petroleum process industries. One application of particular interest is in critical multivariable control loops where the variables may be subject to constraints.

State feedback control and Kalman filtering. Applications in the process industries continue to lag those in other areas, e.g., aerospace systems; however, there are continued efforts to exploit the power and generality of state-space methods particularly since the problem of computer implementation of the algorithms is no longer a consideration.

Adaptive and self-tuning control. The range of effectiveness of the direct control function is extended, under changing process characteristics by updating the controller parameters. This is another concept that has been around for a great many years, but has only recently entered the realm of technical and economic feasibility. Thus, more and more applications are being reported [1].

Robotics

There has been a prodigious level of effort in robotics development in the United States, Japan, and other countries. Much of this is directed to applications in the manufacturing industries – a sector of industry we have not addressed thus far in our discussion. However, robotics will play an increasingly important role in the process industries. We note that even continuous processes are punctuated by discrete events – startup and shutdown procedures, periodic roll changes in the steel rolling mill, scrap additions in the steel converter,product packaging and shipping, etc. In addition, batch processing is an important and growing feature of pharmaceutical, fine chemical, and biotechnology-based products with many potential applications of robotics.

In general, as we expand the domain of the computer control system, we will increasingly employ robotics – somewhat analogous to, but much more flexible and general than actuators – in the role of implementing first-level decisions of the control hierarchy.

Manufacturing Systems

Although we have focussed attention on process-type systems, equally exciting developments have been taking place in the manufacturing sector. Basically, the problems, objectives, and control concepts previously discussed apply almost directly to manufacturing systems or, more generally, discrete event type systems [5]. An important point is that manufacturing systems are being designed to achieve ever higher levels of performance by introducing automated machine tools, flexible manufacturing centers, "just-in-time" policies, minimal in-process storage facilities, robotics, etc. However, these developments make performance more and more critically dependent on the computer control system exercising effective integrated control of the entire plant system. This poses a challenge which is being addressed by many research groups in industry and in universities.

Expert Systems

There is a very rapidly evolving interest in expert systems as an approach with broad implications for expanding the scope and range of effectiveness of computers applied to the control of industrial systems. There are already a number of applications of the methodology proposed – and even some scattered practical realizations – in such diverse areas as adaptive control, process diagnostics, production scheduling, computer-aided design, smart sensors/automated inspection, and batch process control. A few motivating factors behind this direction of interest are: (i) improving the range of effectiveness of the control system, (ii) extending the domain of automatic control, i.e., automating some of the activities and operations currently carried out by humans, (iii) facilitating the process of system design, startup, and updating to new knowledge and new conditions, (iv) increasing overall system reliability and robustness [1, 4, 9, 14].

An expert system has been defined as a machine or computer program that performs an intellectually demanding task at least as well as most (human) experts. The system consists of a knowledge base which contains the knowledge and rules needed to make decisions, a data base which contains the data (e.g., sensor readings, alarm signals, etc.) used in the decision process, and an inference engine which manipulates the knowledge to arrive at and explain decisions.

Some commonly identified attributes of the expert system are:

- It captures the judgemental, experimental, and intuition aspects of a good operator (or "good" designer or "good" decision-maker).
- It incorporates heuristics, reasoning, and rules of inference in its reasoning process.
- It makes the chain of reasoning transparent to the user (i.e., it provides the "why" for its decision).

An interesting new idea being examined is the incorporation of expert systems methodology at the higher levels of the control hierarchy where time scale, problem complexity, and degree of uncertainty tend to increase with respect to level in the hierarchy. Thus, as expressed in [4], "Knowledge of a domain takes many forms. When that knowledge is firm, fixed, and formalized, algorithmic computer programs that solve problems in the domain are more appropriate than heuristic ones. However, when the knowledge is subjective, ill-codified and partly judgemental, expert system embodying a heuristic approach are more appropriate."

Technology Transfer

Some comments are in order concerning the relevance of advanced computer control technology to developing countries.

(a) While labor-saving is often an important factor in highly industrialized economies, it may be very unimportant, and even a negative factor (for political reasons) in developing countries. However, the contribution of advanced control to improving product quality, increasing energy efficiency, decreasing losses, etc., should be of particular importance to the developing economy.

(b) Developing countries may often "leapfrog" their technology gap by purchasing "turnkey" systems that incorporate state-of-the-art technology. To make this technology transfer viable, however, local personnel must be trained in the new concepts and methodologies so they have the background understanding necessary to maintain, update, and modify the system to accommodate new circumstances and priorities as they arise.

As a case in point, the industrial control groups at Purdue University and Case Western Reserve University were recently commissioned by the government of India to put on an intensive training program for a group of engineers from the Indian steel industry. The purpose of this program was to prepare the Indian engineers to take responsibility for an advanced hierarchical computer control system to be installed in one of their plants. Thus, although all the hardware, software, and systems design are being imported, the know-how to manage, maintain, and evolve the system are to be locally established.

(c) Industry-university collaboration on research and development projects is a very effective means of keeping in the forefront of advancing technology. This is just as important in developing countries as it is in more highly industrialized economies. At Case Western Reserve University, for example, we have a long history of collaborative research with a consortium of companies in the area of control of industrial systems. I am pleased to note that here at Cairo University there are also joint projects with industry – specifically, with the steel and textile industries – to study the application of computers for improving plant performance.

(d) Another avenue for facilitating technology transfer is via the expert system. We consider two features of the methodology: the learning capability and the explaining capability.

In the learning process, the expertise of the "expert" or trained operator is encapsulated within the computer program, embracing all of the facts, behavioral characteristics, rules, and heuristics necessary to achieve a desired level of performance out of the production system. The computer then translates this "knowledge" into appropriate code that can be utilized by the operator trainee in a training or operator guidance mode.

In the explanation process, the expert system diagnoses the current state of the process and makes recommendations to the operator for actions to take. On request, the computer can then explain the reasons underlying the recommendations; i.e., how it interpreted the available information, what assumptions were employed, and perhaps even what weightings or priorities were assigned to conflicting observations or inferences. In this way, the operator is able to learn from the experiences and understanding incorporated into the expert system. Equally important, the operator can question the bases under which conclusions were drawn, inputting his own judgement or experiential factors to arrive at a more acceptable computer output.

This area of application is still very much in its infancy; it seems clear, however, that there is considerable potential for significant contributions to technology transfer as the associated artificial intelligence methodologies become more cost effective and efficient.

Conclusions

Remarkable progress has been made in computer control applications to industrial systems. This progress will continue, driven by (a) continuing pressures for improved performance of production facilities, (b) continuing advances in computer technology, and (c) continued developments in systems and control theory and practice. This progress is relevant to industries in developing countries in the context of such control objectives as improved product quality, increased efficiency of scarce resource utilization, etc. While state-of-the-art technology can be imported via "turnkey" commissions, it is important that in-house expertise be developed so that local, resident personnel can maintain, manage, and even update these imported systems to meet changing needs.

References

1. Bristol, E.H., "The Design of Industrially Useful Adaptive Controllers", *ISA Transactions*, **22**, (1983).
2. Buchner, M.R. and I. Lefkowitz, "Distributed Computer Control for Industrial Process Systems: Characteristics, Attributes and an Experimental Facility", *Control Systems Magazine*, Mar. 1982.
3. DyLiacco, T.E., "Control of Power Systems via the Multilevel Concept", Ph.D. thesis, Case Western Reserve Univ. (1968).
4. Hayes-Roth, F., D. Waterman and D. Lenat, eds., *Building Expert Systems*, Addison-Wesley (1983)
5. Lefkowitz, I., and J.D. Schoeffler, "Multilevel Control Structures for Three Discrete Manufacturing Processes," *Proc. Fifth IFAC Congress*, Paris (1972).
6. Lefkowitz, I. and A. Cheliustkin, "Integrated Systems Control in the Steel Industry," *Report of the International Institute of Applied Systems Analysis*, Laxenburg, Austria, CP 76-13 (1976).
7. Lefkowitz, I., "Integrated Control of Industrial Systems," *Trans. Royal Society*, **287** (1977).
8. Lefkowitz, I., "Hierarchical Control in Large Scale Industrial Systems", chapter in *Large Scale Systems*, Y.Y. Haimes, ed., North Holland Publishing Co. (1982)
9. Lefkowitz, I., "Integrated Control of Industrial Systems", *Proc. of the International Seminar on Distributed Control*, New Delhi, Sept. 1986.
10. Mesarovic, M.D., "Multilevel Systems and Concepts in Process Control," *Proc. IEEE*, **59** (1970).
11. Pao, Y.H., and T.E. DyLiacco, "Artificial Intelligence and the Control of Electric Power Systems", *Proc. IFAC World Congress*, Budapest (1984)
12. Schoeffler, J.D., "On-line Multilevel Systems," in: *Optimization Methods for Large-Scale Systems with Applications*, D.A. Wismer, ed., McGraw-Hill, New York (1971).
13. Singh, M.G., and A. Titli, *Systems – Decomposition, Optimization and Control*, Pergamon Press, Oxford (1978).
14. Taylor, J.H., and D.K. Frederick, "An Expert System Architecture for Computer-aided Control Engineering", *Proc. of the IEEE*, **72** (1984).
15. Williams, T.J., editor, *Analysis and Design of Hierarchical Control Systems*, Elsevier Science Publishers, Amsterdam (1985).
16. Williams, T.J., "Distributed Digital Computer Based Industrial Control Systems in the Western World and Japan – A Status Report", *Proc. of the International Seminar on Distributed Control*, New Delhi, Sept. 1986.

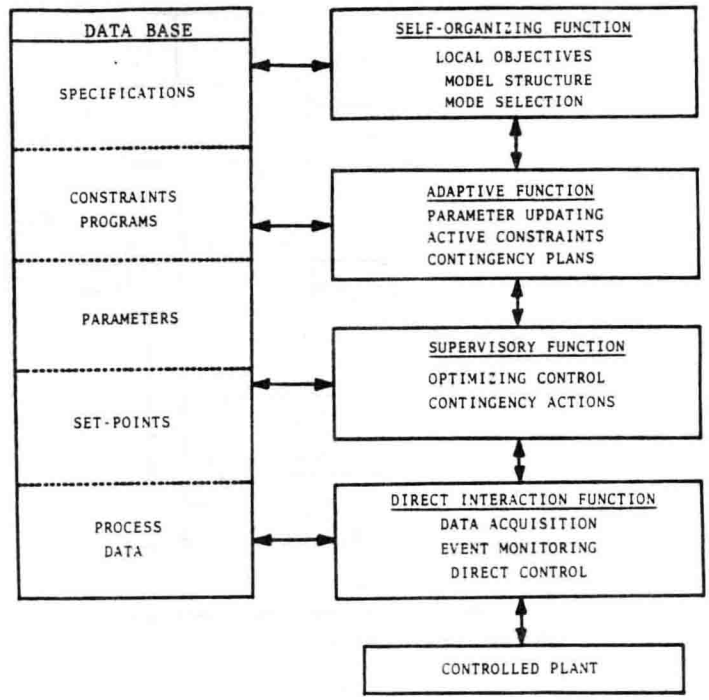

Fig. 1 Functional multilayer control hierarchy.

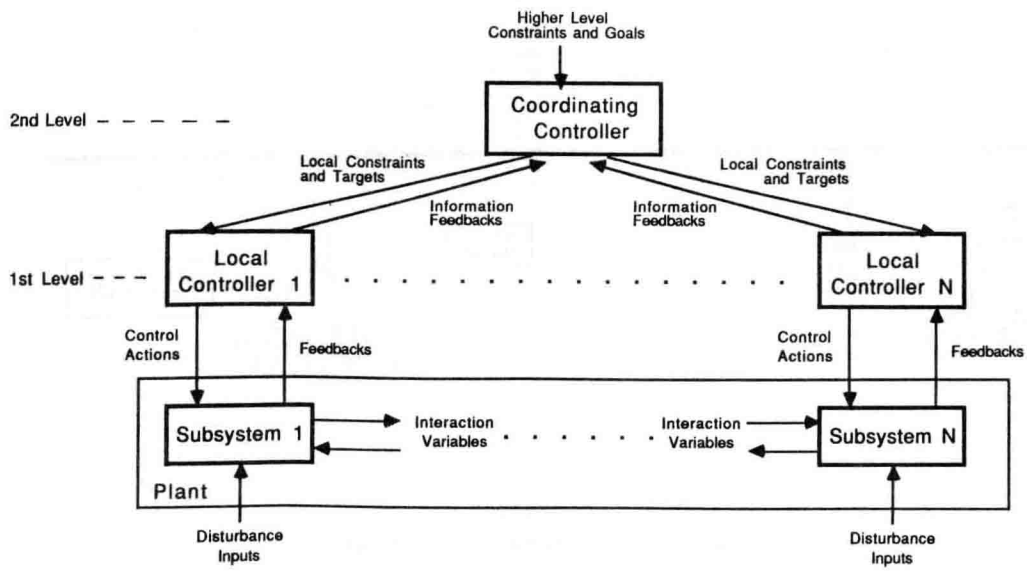

Figure 2 Multilevel Hierarchical Structure

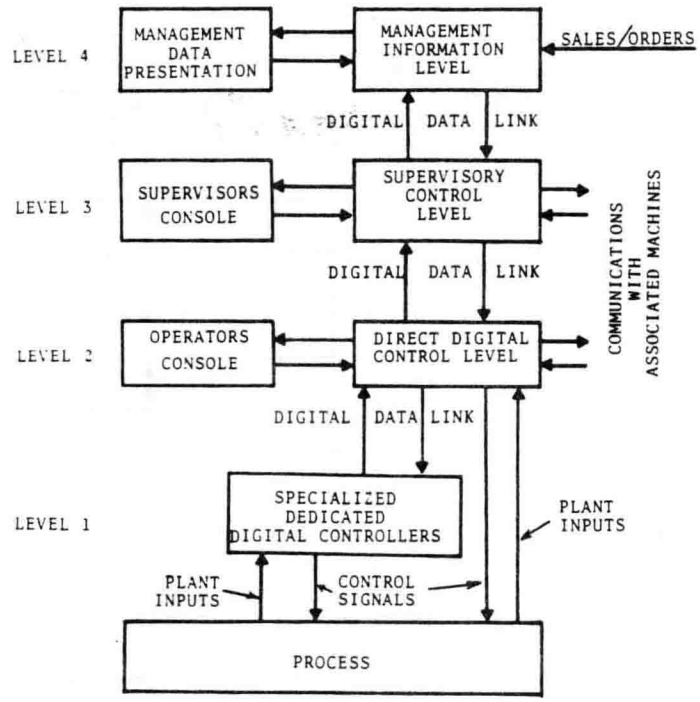

Fig. 3. Alternative configuration of control hierarchy.

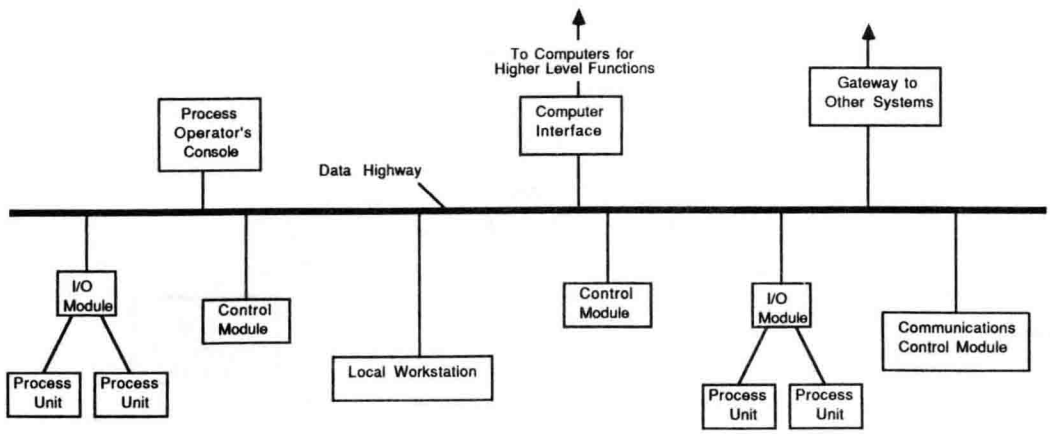

Figure 4 Basic Features of Distributed Control System

A COMPARATIVE STUDY OF SOME PROGRAMMING LANGUAGES IN REAL-TIME APPLICATIONS

J.A. Cerrada (**), M. Collado (*), R. Morales (*), J.J. Moreno (*)

(*) Departamento de Lenguajes y Sistemas Informáticos e Ingeniería de Software.
Universidad Politécnica de Madrid.
Km. 7 Carretera de Valencia
28031 Madrid - Spain.

(**) Departamento de Automática, Ingeniería Electrónica e Informática Industrial.
Universidad Politécnica de Madrid.
José Gutiérrez Abascal, 2
28006 Madrid - Spain.

ABSTRACT

This paper is devoted to a comparison among several notations for describing real-time programs. Some of them are well known languages, and others are original of the authors of this paper. The main goal is to measure the relative adequacy of every language for programming industrial real-time applications.

The set of compared languages includes Ada, Occam and CHILL, as well as extensions of Pascal and Modula-2 developed by the authors. Language features for concurrent or real-time programming are taken into account, and also their expresive power and their facilities for using good methodologies.

Conclusions are derived about the adequacy of every language. A comparative table summarizes the characteristics of every language, allowing an easy comparison.

KEYWORDS: Real-Time Programming Languages, CSP, Ada, Occam, CHILL, D-Pascal, CC-Modula.

1. INTRODUCTION

Real-time programming is at present an open field. Real applications are usually programmed by using a sequential programming language from which several timing, communication and synchronization primitives are programmed as "system calls" to the operating system. Only few languages include some of these primitives in their own ones.

The purpose of this work is to compare some of these languages. The main goal is to measure the relative adequacy of every language for programming industrial real-time applications.

The set of compared languages includes Ada [2, 7], Occam [8, 9], CHILL [4, 11], D-Pascal [5] and CC-Modula [6]. The last ones are extensions of Pascal and Modula-2 developed by the authors.

The comparison is carried out by analyzing the relevant features used in previously developed real-time software projects. These features have been grouped in three categories: general language features, concurrency mechanisms, and real-time facilities.

Every feature is analized separately. A comparative table summarizes these features for all the languages, thus allowing an easy comparison. This table could be used as a guide for language selection.

2. APPLICATION SCHEMA

Before evaluating a programming language it is necessary to stablish what kind of programs will be considered. In the present work, the objective is to develop software for industrial processes supervision and control systems.

Previously developed projects in this field have led to several versions of a common schema. The software had been developed in FORTRAN for historical reasons. The motivation for the present analysis is to find better languages candidates in order to build new sofware starting from scratch.

The general schema appears in Figure 1. The whole application has been decomposed into processes that cooperate trough message exchanges. This schema shows the program structure of a single node in a distributed network.

In addition to communication by messages, it is also interesting to consider the possibility of having common data areas, in order to improve efficiency.

The schema is general enough as to serve as a basis for analyzing the adequacy of a given language for programming this kind of applications. The main features that appear in this schema are:

- Parallel execution of several processes.
- Communication by message exchanges (buffered and unbuffered).
- Communication by controlled shared data.
- Synchronization by message or signal exchanges.
- Computation and recording at specific time instants.
- Multiple instances of a given process in order to manage several instances of a given resource (terminals, printers, etc.).
- Peripheral error detection using a time-out schema.

These and other features will be taken into account in the following sections.

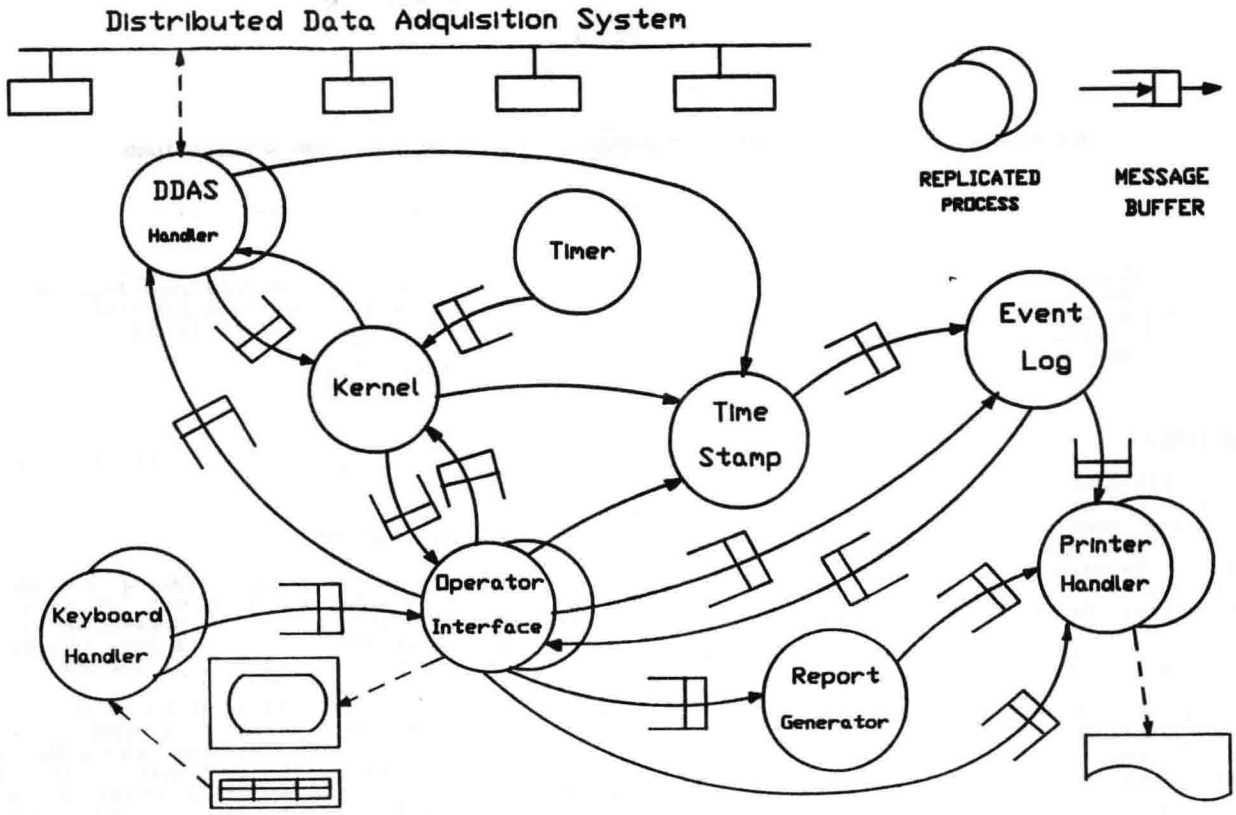

Figure 1. Data flow diagram of the prototype application.

3. GENERAL LANGUAGE FEATURES

Actual trends in programming methodology mark clear preferences towards a set of language features. Perhaps the most desirable ones are notation clarity and object-oriented features.

The clarity degree shown in every considered language is similar. All of them are high level structured languages with very similar notation for structured statements and subprogram definition.

The only exception is, perhaps, Occam. In this language a control structure is denoted by indentation. In addition, the general taste of the language is rather low-level, lacking of powerful data structure handling, and using very short keywords and unusual operator symbols.

Facilities for object-oriented programming rely mainly on the module definition mechanism [3]. This feature is found in Ada, CC-Modula and CHILL, but not in Occam or D-Pascal.

Other concepts also influence the adequacy of a language with respect to the nature of the applications. In the present case, genericalness is a very desirable characteristic, applicable, for example, to modules that implement similar operations on different data types. Only Ada and (in some sense) CC-Modula allow generic modules definition.

Another interesting feature is exception handling, that can simplify the programming of error recovery. Only Ada and CHILL provide specific program structures to handle exceptions.

4. CONCURRENT PROGRAMMING

Every considered language provides facilities for process or task declaration.

Parallel execution of the defined tasks is implicit in Ada and D-Pascal. In other languages it must be explicitly programmed. Occam provides the PAR statement, and CC-Modula uses a COBEGIN..COEND structure.

From another point of view, we can consider whether processes are or not created or destroyed dynamically. In Ada, CC-Modula and Occam, processes are dynamic objects. In D-Pascal, tasks are static and declared in the global program structure.

The application schema described above match very well the conceptual model of concurrent processes that cooperate via message exchanges. All the considered languages provide facilities for implementing this model. In addition, it is also possible to have shared data when necessary.

Occam, CC-Modula and D-Pascal use variants of the CSP schema [1]. Ada implements a "rendez-vous" mechanism at the procedure call level, also based on the CSP policy. CHILL posesses a variety of communication and synchronization features, including buffers for communication, signals and regions for communication and synchronization, and events for synchronization.

The CSP schema combines message passing and non deterministic selective receptions by means of "guarded commands". A SELECT or similar

statement in Ada, Occam, D-Pascal and CC-Modula implements a guarded command. In CHILL, a receive case statement implements non deterministic reception, but without the selective capability of a guarded command.

Task termination is hard to program if the task uses non deterministic delayed receptions. The CSP schema specifies an automatic termination when all the cooperating tasks are waiting for messages at the same time. This model is easily programmed in Ada by using the TERMINATE clause in SELECT statements, but must be explicitly programmed by passing termination messages in the remaining languages.

There are several distinct ways to specify message exchanges. Ada implements a kind of remote procedure call in which the caller specifies the name of an entry point in the invoked task. In D-Pascal and CC-Modula messages are routed through named channels, usually bound to specific tasks. A similar schema is used by Occam, but without an explicit relation between channels and processes. In CHILL, buffers and signals are used the same way as Occam channels.

Another interesting characteristic is the facility for executing multiple copies of a defined process. The application schema uses this policy to handle several peripherals of the same class connected to a single processor.

Some languages provide this facility. In Ada it is possible to define arrays of tasks. CHILL and CC-Modula allow activation of multiple instances of a given process. D-Pascal and CC-Modula don't support arrays of tasks, but we can declare arrays of channels.

The better way to start several instances of the same process is found in Occam, that contains an optional replication clause in PAR or ALT statements.

5. REAL-TIME FEATURES

Concurrent programming languages are not enough by themselves to adequately develop real-time applications. Some facilities for real-time handling must be added.

In general, real time handling has been taken into account as a normal operation in special purpose languages designed for writing computer control applications. But only very few general purpose languages include this kind of operations as primitive ones. This is the case of languages oriented toward "embedded computer systems" software development.

The kind of applications covered in this analysis need the following set of facilities:

- Real date and time reading.
- Compute interval lengths.
- Execute actions at fixed intervals.
- Execute actions after a given delay.
- Execute alternate actions after a time-out period.
- Clock synchronization in distributed systems.

In fact, real-time needs are more complex than explained. In some applications, limits for the response time to specific events should be specified easily, but this is seldom faced at the programming language level, and it is usually solved by craftmanship, by tuning the internal scheduling policy or by buying or developing a convenient hardware.

5.1 Real time and date.

This feature is available in every system, but not always in the same way. Only few languages include primitives for getting date and time. This is the case of Ada (which has the predefined package CALENDAR) and the D-Pascal proposal (which has a set of predefined procedures).

In the remaining cases, the language lacks of primitives for time and date reference, but in every support environment there is a "system" library providing this functionality in a machine-dependent way.

Computations involving time can be programmed in a well organized way in Ada and D-Pascal. In these languages the time is a predefined data type and only a set of consistent operations can be applied to it.

In other languages, like CC-Modula and CHILL, the time must be explicitly mapped into numerical values and handled the same way as other magnitudes.

A special case occurs in Occam, in which the time is handled as a cyclical integer counter, accesed by the predefined element NOW, with machine-dependent range and resolution. Modular arithmetic must be used in time computations. The AFTER operator is used specifically to compare two time instants in chronological order.

5.2 Timed operations

The execution of a given operation at a given time is usually programmed by using a delay or wait primitive. Depending on the language, the waiting time is expressed in absolute or relative form.

In Ada, the DELAY statement causes a wait for a given relative duration. In Occam there is a similar WAIT statement, but it specifies an absolute time. In D-Pascal, a Wait_time predefined procedure operates also in an absolute time basis.

A delay or wait statement inside a loop can be used to program a periodic operation in every language.

In our experience, there is little difference in clarity and expressive power between the absolute and relative expression of the time. However, it seems that absolute time is potentially better in order to obtain execution at precise instants.

Another specially interesting timed operation appears in the classical time-out schema used to recover communication failures and other kind of malfunctions. At the language level, only special-purpose languages include this facility directly.

In Ada and Occam, a time-out schema can be

programmed by using a DELAY/WAIT as a guard in a SELECT/ALT structure. The timed branch is selected after the specified time if no other action has been selected before.

In D-Pascal and CC-Modula the time-out mechanism is associated to message exchanges. Every send or receive operation can be assigned a time limit to wait for. This arrangement matches the requirements of the selected applications in a natural way.

5.3 Multiple clock synchronization

In multiprocessor systems, and specially in distributed ones, a separate clock can exist in every processor. In this case, a problem arises if clocks run asynchronously because local time can vary from site to site and events cannot be chronologically ordered in a consistent way [10].

At present, clock sychronization is an open research field. Commercial systems usually don't address this problem neither at the language level nor at the operating system level.

6. CONCLUSIONS

All the languages are more or less based on the CSP schema and contains facilities for programming the kind of considered applications. Differences are found in the way used to program specific mechanisms.

	Ada	CHILL	Occam	D-Pascal	CC-Modula
Module definition	Yes	Yes	No	No	Yes
Generic modules	Yes	No	No	No	Partially
Exception handling	Yes	Yes	No	No	No
Parallel execution	Yes	Yes	Yes	Yes	Yes
Dynamic processes	Yes	Yes	Yes	No	Yes
Process comm.	Rendez-vous	Buffer Signal	Mesg.	Mesg.	Mesg.
Process synchr.	Rendez-vous	Signal Region	Mesg.	Mesg.	Mesg.
Guarded command	Yes	Partially	Yes	Yes	Yes
Real-time and date	CALENDAR	System lib.	NOW AFTER	Predef proc.	System lib.
Timed operations	DELAY	System lib.	WAIT	Wait & tmeout	time-out

Figure 2. Summary of language features.

As a general conclusion, a distinction must be made between features for concurrency and for timing. Time specification and management is a much more difficult task than expressing and controlling paralellism. Design of true real-time programming languages and tools is at present an open field, and a very important necessity in practice.

Because of the variety of analized features, a ranking of the languages cannot be established. Instead, a table comparing every feature in every language is included in Figure 2.

REFERENCES

[1] G.R.Andrews, F.B.Schneider: "Concepts and Notations for Concurrent Programming". ACM Computing Surveys, March 1983.

[2] J.G.P.Barnes: "Programming in Ada". Addison-Wesley, 1982.

[3] G.Booch: "Object-Oriented Development". IEEE Trans. Softw. Engin., Feb. 1986.

[4] CCITT: "CCITT High Level Language (CHILL). Recommendation Z.200". CCITT, 1985.

[5] J.A.Cerrada, M.Collado: "Implementation of a CSP-based extension of Pascal". Microprocessing and Microprogramming (to be published).

[6] M.Collado, R.Morales, J.J.Moreno: "A Modula-2 Implementation of CSP". ACM SIGPLAN Notices, June 1987.

[7] DoD: "Reference Manual for the Ada Programming Language". ANSI/MIL-STD 1815A, 1983.

[8] J.Geraint: "Programming in Occam". Prentice-Hall, 1987.

[9] Inmos, Ltd.: "Occam Programming Manual". Prentice-Hall, 1984.

[10] L.Lamport: "Time, Clocks and the Ordering of Events in a Distributed System". Comm. ACM, July 1978.

[11] C.H.Smedema, P.Medema, M.Boasson: "The Programming Languages Pascal, Modula, CHILL and Ada". Prentice-Hall, 1983.

Developing Multimicro Embedded Applications Using a Concurrent Language

M.Ancona(*), A.Clematis(+), G.Dodero(*), V.Gianuzzi(*)

(*) Dipartimento di Matematica dell' Universita' di Genova
(+) Istituto per la Matematica Applicata del C.N.R.
Via L.B. Alberti,4 - I 16132 Genova, Italy

Abstract

This paper describes an experience in the development of a control system for a set of cooperating machineries; the system consists on a network of Intel 8088 microprocessors and its software has been designed using the concurrent language MML. The purpose of our experience is to verify how practical is to employ a concurrent language and its development system in an industrial environment.

Keywords: Multimicroprocessor systems, Process control, Concurrent programming languages, Embedded distributed systems.

1. Introduction

In the application domain of distributed systems, an emerging position is assigned to the embedded multimicroprocessor systems especially suited for industrial applications. They usually consist on a relatively small number of processors, on which concurrent activities are executed; the hardware architecture of such systems may be composed of heterogeneous processors, and the application can be partitioned into a number of statically defined processes. Many industrial applications, indeed, constitute a set of problems which naturally lead to a distributed solution, which is usually simpler than the corresponding non- distributed one. In fact, such systems provide the following features:

- Distribution: it is preferable that each machine has its own control equipment, physically close to the controlled machine;
- Flexibility: it should be easier to upgrade a system to handle a greater workload by adding processors, rather than by replacing existing processors by larger ones;
- Coordination of the various components: such coordination is usually achieved by exchanging a limited amount of information, such as for instance, speed, alarm conditions and the like.

Usually, this kind of computer applications do not involve a large number of delivered units (from a few units to some hundreds per year), while the features of the controlled machineries can vary with a relatively high frequence. This consideration suggests, rather than an optimization on unit cost of the control system, the choice of a multiple microprocessor system, which at first glance may even seem overdimensioned for the original problem, but which shall provide advantages in installation and maintenance (human costs) throughout product life cycle.

To this purpose, the features provided by the software development system have a considerable impact. The increasing complexity of control algorithms leads to the use of high level languages, which is also indispensable to increase software reliability. Unfortunately, most commercially available software development environments for this latter type of multimicro systems (e.g. [Int83]) do not support a 'true' high level concurrent programming language, with separate compilation and object oriented programming features. Hence, application programmers are charged with the burden of expliciting the operations such as process synchronization, mutual exclusion and interprocessor communication. This is accomplished by invoking basic kernel services; however their names, functions, parameters etc., vary from one implementation to another, and no standardization proposal has been accepted until now; although the TRON project [Saka87] is now affording the problem.

On the other hand, our approach has been to analyze, with a case study, the possibility of the development of these applications with a concurrent language, thus avoiding low-level kernel details: the study involves sketching of the sample application in the concurrent language MML [Boar84] and evaluating its development in the MMDS programming environment [Anco86].

Therefore, the paper contains a short description of a possible modular hardware architecture, which seems to be well suited for the intended application; then, we outline a sample application involving a number of machineries receiving items to be processed from a conveyor belt and delivering them back to it. Allocation of software processes to hardware resources completes the discussion.

2. Architectural features

A possible architecture for the embedded multimicro system is that of a loosely coupled distributed system, in which the processors communicate by links, which may be point-to-point, serial or parallel lines. On the other hand, there may be situations where more than a single processor is required for the control of a single component. This is the case when speed problems arise, or when the system is replicated to achieve a better fault tolerance, or when debugging and self-diagnosis have to be performed in parallel with the usual control cycle. To this purpose, a tightly coupled distributed system may be selected, in which processors communicate via a shared memory, which may be dual (or multiple) port, or on a global bus. This cluster of processors is the basic 'control unit', for designing control systems.

In the case study discussed hereafter, we employed a hardware architecture based on Intel microprocessors, which provides two different inter- and intra-control units connections possibilities.

The system has a high degree of modularity and can be tailored to specific application requirements by changing:

- the number of clustered units,
- the number of processors in each cluster.

Figure 1 shows a typical cluster configuration with a master processor, which takes care of I/O bound processes, and two slave processors, devoted to intensive computations. Dual port memories are used for interprocessor messages; each processor has its own private memory for local data. Each I/O peripheral unit contains two RS232 serial and 48 parallel lines.

Three considerations lead to the selection of 16-bit microprocessors, with a large addressing space, with respect to 8-bit micros:

- The production cost of hardware for a 16-bit micro is acceptably higher than that for a 8-bit micro, especially when compared to the unavoidable cost for all peripherals and control devices. Any organization which develops and maintains such control systems has to face internal standardization problems. To this purpose, the cost of human resources may well overrun that of hardware, and it is often desirable to support only one architecture for all aplications of this type.

- Many tools available for the debugging of single processors (hardware emulators, tracers etc.) lose a great deal of their efficiency in multiprocessor environments. The debugging of cooperating processes at the multiprocessor level requires a run-time "high-level debugger" to interface the operator directly with those processes[Baia83, Bate83, Garc84]. Such a function must be supported by a distributed software layer, which is logically "above" the basic run-time, and "below" the application software. Again, such a separation is better obtained where a larger addressing space is available, together with support features for memory management and segmentation, as for 16-bit micros.

- If the overall architecture seems to be over-dimensioned for the required control task, its additional computational power can be saved for future expansions of the system, or it can be employed to guarantee increased availability. For instance, self-diagnostic programs can be automatically executed during idle time periods, and fault tolerant features can be added [Anco87].

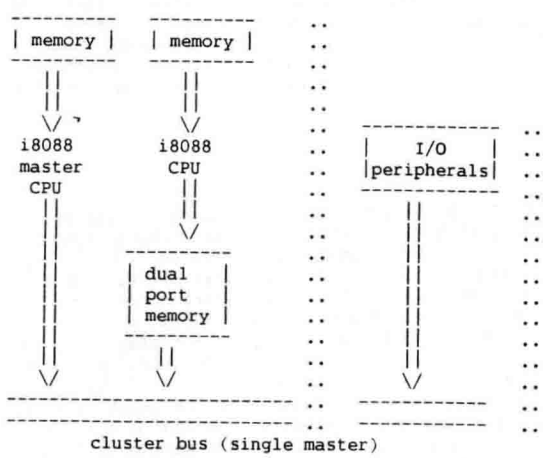

figure 1

3. A case study

Most industrial control problems can be expressed in the producer- consumer schema. The simplest expression of this classical problem comprises a producer process (which supplies information to others), a consumer process (which has access to such information for further processing), and a buffer (a process which allows asynchronous computations between producer and consumer processes). This basic schema can be replicated "serially", that is, with a pipeline of processes where each consumer behaves as a producer for processes to elaborate afterwards the partially processed information. The same schema can be replicated in "parallel", that is in a system where n producers supply information to m consumers through a buffer or pool of buffers allowing a greater parallelism in the computation (see figure 2).

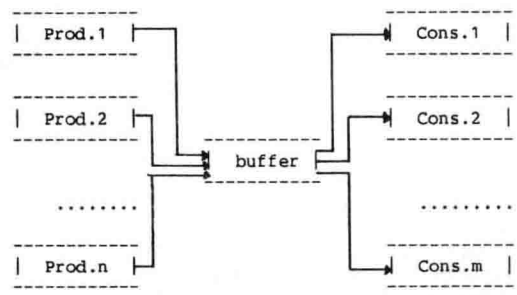

Figure 2

The case study detailed in the following lies in the above classification, namely, three producers operating in parallel with one consumer only. This case is typical of those applications where one phase of manufacturing requires a longer time than the following phase (for instance, manufacturing of single items followed by their packaging in groups of 10's).

Producers shall deliver the item to the conveyor belt only if the actual position on the conveyor belt is free; while they shall be kept waiting until the consumer frees the position by starting to process the item. Conversely, the consumer is inactive until an item is made available by some producer.

Let us assume also that the conveyor belt operates at constant speed; therefore each machinery must adapt its own processing speed to the presence (or absence) of items on the corresponding conveyor position. The most obvious solution would be to develop a separate control system for each machinery, each one connected to a number of sensors to analyze the physical environment, and to I/O lines for interprocessor signals.

This simple coordination among the various independent computing elements (i.e. non concurrent programming) implies a limited amount of information exchange among machineries, such as acknowledgements or error flags. As a consequence, the configuration of the assembly line has to be decided before the actual software can be written, and the amount of information to be exchanged is drastically limited. Also, the various machineries must have separate operator panels, since no centralized software for operator interface can be written for a variable number of input/output lines.

Possible problems arising by this "sequential" view of the problem are:
- in the case of variable features of the item, errors may arise if the same item is programmed by the operator in different ways on different machineries;
- with limited amount of information exchange among machinery controllers, it is difficult to establish their optimal processing speed;
- availability problems may arise, when the whole system is halted, for instance after fault of one producer only, which might instead be recovered by increasing processing speed of the remaining producers.

In terms of concurrent processes, the above problem would be expressed by defining also a buffer process,

expressing the logical behaviour of the conveyor belt, and possibly its own control algorithm, if any. Also, one of the concurrent processes may be dedicated to the operator interface functionality handling the inputs and outputs of the control panel, and dispatching relevant information, when altered by operators, to producers and consumers. We remark that this latter feature, which is often executable in non-operational phases only (namely, items cannot be varied on-line), usually requires a lot of programming effort, and it is subject to frequent modifications during product life cycle, for instance to meet new safety regulations (see figure 3)

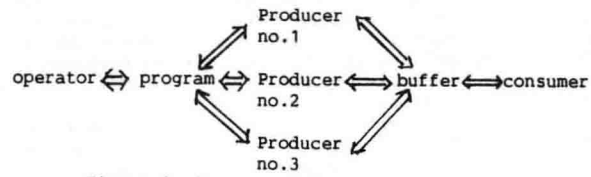

Figure 3: Concurrent Processes Structure

Let us now describe in short the three main processes, Producer, Consumer and Buffer, in the programming language MML. An MML process is called "sequence" and a family of processes can also be written, where the basic algorithms are the same, but they only differ in low- level details such as peripheral addresses. Processes commutate by means of remote procedure calls, called controlled procedures, with input and output parameters, and there is no possibility of data sharing. The number of processes composing a family need not be specified at compile time, it is only needed by the linker and by the kernel to allow the execution of processes. Therefore we can sketch the following MML portion of program for the family of producer processes, where in our application the index "i" will vary from 1 to 3.

```
Prod[i] family
global  vel count  word     !initial value!
                 velmax word := ... !some value!
Main procedure
entry
   do
   ...parameters... :=program.p(i)
   vel:=eval(...parameters...)
   do
      if alarm then vel:=0  stop  exit
            else   execute    fi
      if item_processed then
                 count:=buffer.put
                 vel:=eval(...parameters...) fi
      od
   od
end Main
end Prod[i]
```

Here "vel" is the actual producer speed and "count" is the actual number of items already present on the conveyor belt. This counter is updated as a return parameter of the remote procedure call "buffer.put". Procedures not expanded in the text have the following functionalities:

eval: evaluates actual processing speed as a function of COUNT and VELMAX;

program.p: remote procedure "p" of process "program" which dispatches new programming parameters, if any, to the i-th producer;

alarm: controls 'local' alarms, and requests to halt the assembly line from operator or from the consumer process;

stop: halts the machinery;

execute: controls a machinery cycle on a single item; its execution time is assumed to be sufficiently short to guarantee that no new alarm shall be generated in the meantime.

```
Consumer sequence
global a vel count word :=0
       velmax WORD :=...
Main procedure
entry
  do
    ...parameters...:=program.c
    do
      if alarm then vel:=0  stop  exit
               else a count := buffer.get
                    vel:=eval(...parameters...)
                    execute   fi
    od
  od
end Main
end Consumer
```

In this sequence, procedures with names similar or identical to those above illustrated have the analogous functionality but different implementation. Note that "eval" is a function of both "a" and "Count" (the former discriminates between picking up the last item, and the case where no more items are available).

```
Buffer sequence
global count ok word :=0

Put controlled procedure returns (c word)
entry
  count:=count + 1
  c:=count   ok:=1
end Put

Get controlled procedure returns (a c word)
entry
  if count>0 then count:=count-1
                  a:=1  ok:=1
             else a:=0   fi
  c:=count
end Get


Main procedure
entry
  do
    if ok=1 then ok:=0
                 vel:=eval(...parameters...)
                 execute
            else retest !wait until ok! fi
  od
end Buffer
```

The conveyor belt is active only if some item is present (count > 0); and execute is the control algorithm for its motor (recall the fixed speed assumption). In case the conveyor belt is driven by the consumer motor, the MAIN procedure is not necessary (i.e. only the logical function of the buffer is required).

A more sophisticated implementation of the buffer process may contain not just the simple 'number of items' counter, but also their relative position on the conveyor belt in some vector. This information must therefore drive the relative speed evaluations (Eval procedures) of producers or consumer; the hardware system might also contain several sensors for monitoring item positions. Note also that the reverse approach for relative speed adjusting is possible, namely, it is easy to specify similar processes in the case of fixed speed producers/ consumers and variable speed conveyor belt.

Diagnostics and/or statistical processes can be written to complete this MML program in a similar way.

4. Hardware and software integration

To complete our case study, we shall now briefly discuss the two following points:

1. how to select a configuration among those possible in our modular architecture,

and

2. how to distribute processes to processors in such a configuration.

To this purpose, the first choice is taken after considering the following points which often arise in industrial process controls:

a. Coordination of various machineries, each of them being programmable by the operator independently from the others.

In this class we can list these applications:
- an assembly line as the above one, where some item is being manufactured;
- the coordination of two or more independently controlled robots acting in a common workspace [Bast87];
- an automatic distribution line in a store, with the aim of optimizing the storage;
- the collection of data on plant behaviour, to produce statistics on its operarations, for maintenance or accounting purposes.

b. Separate control of various parts of the same machinery, in order to minimize response times.

This does not necessarily imply tight coupling of processes, such as it happens in array processors, but medium granularity of parallelism is sufficient, where those functionalities which are logically less correlated are effectively done on separate computational units. Possible examples of this case are:
- separate controllers for the motors driving robot arm articulation [Kasa85];
- separate controllers for the various functions of a complex machinery, or for operator interface.

Consider also that a limited amount of information exchange does not necessarily require availability of common memory shared among all processors, especially if such an event is likely to happen only off-line, as in the case of the "program" process of the previous example.

A set of cooperating machineries may even exhibit both of the above said types of problems; this is the reason for the above choice for the architecture, where the hardware hierarchy directly reflects the two aspects (a) and (b):

a. any machinery is controlled by an autonomous control system, possibly a cluster of processors, which communicates to other controllers through relatively slow links, the operational speed being however bound by mechanical considerations;
b. the control system of each machinery may be composed by one or more processors close to one another, with a tighter connection to guarantee a fast communication, for instance by means of double-port memories.

For the problem (2), it has to be decided how to do process distribution to hardware resources, i.e. processors, memories, communication channels, in accordance with real-time requirements of the application. MMDS has been designed to favour separation between software design and allocation of processes, since it also provides a tool for simulating on a single microcomputer the expected behaviour of the application software on the microprocessor networs [Gian85]. Thus, various configurations and software allocation strategies can be experimented with little efforts.

In the case study, we may decide to dedicate a cluster to each producer machinery; a microprocessor can be devoted to operator interfacing and then connected with a serial link to all clusters or processors; the conveyor belt and the consumer machinery can be both controlled on the same cluster (by one, two, or more processors according to response time). Also the clusters devoted to producer machineries may be composed by one or more processors, according to the complexity of their control algorithms.

In product evolution, assembly lines with a different number of 'producers' and 'consumers' (each one identical to one of those above considered) are likely to occurr. Variations in the number of producers are simply achieved by varying in accordance the value of the index in the family of processes 'Producer' (and possibly in the operator interface); while addition of consumers can be obtained by converting the code of the 'Consumer' sequence into that of a family of processes. Corresponding additions shall be made to hardware configurations.

5. Conclusion

The paper has illustrated how industrial controls can be developed in flexible and efficient way by use of multimicroprocessors programmed in a concurrent language. The proposed methodology, which has been experienced in a case study, is composed by three steps, as described in the previous sections:
- choice of a modular architecture, configurable as a set of loosely coupled clusters, each one containing one or more tightly coupled processors; processing units are 16-bit micros.
- choice of the MML concurrent language, rather than a non-concurrent one, togheter with low-level kernel functionalities. The case study has shown that the choice of a concurrent language allows a clear separation between processes; and that it encapsulates a safe mechanism for process coordination and synchronization, thus avoiding mutual exclusion problems, interprocessor synchronization and timing signal exchange (these two functions are transparently performed by the MML kernel);
- final integration of the concurrent software on some distributed system configuration.

This latter phase is especially eased by the development system we employed, MMDS, which allows a logical separation between design and debugging of MML programs, on a simulator, from allocation and performance tuning, on the chosen hardware configuration.

References

[Anco86] M.Ancona et al., "MML: a programming system to develop applications for multimicroprocessors", in:'acific Computers Communications 85', K.H.Kim, K.Chon, C.V.Ramamoorthy (eds.), North Holland, 1986, 393-395.

[Anco87] M.Ancona et al., "Using Different Language Levels for Implementing Fault Tolerent Programs", Microprocessing and Microprogramming, North-Holland, Vol.20, 1986.

[Baia83] F.Baiardi et al., "Development of a debugger for a concurrent language", Proc. ACM Symp. on high level debugging, 1983, 98-106.

[Bast87] R.A.Basta, "Multiple Arm Coordination Using Concurrent Processing", Proc. IEEE Southeast Conference 1987, pp.328,331, 1987.

[Bate83] P.C.Bates, J.C.Wilden, "High-level debugging of distributed systems: the behavioral abstraction approach", J.Systems and Software, 1983, 255-264.

[Boar84] M.Boari et al, Multiple- Micrprocessor Programming Techniques: MML, a New Set of Tools, IEEE Computer (January 1984), pp. 47-59.

[Garc84] H.Garcia-Molina, F.Germano, W.H.Kohler, "Debugging a distributed computing system", IEEE Trans. on Software Engineering, Vol. SE-10, no.2, March 1984, 210-219.

[Gian85] V.Gianuzzi, "Time performance evaluation in multimicroprocessor emulation", IASTED Int. Conf. on Computer Aided Design and Applications, Paris, June 1985.

[Inte83] "iAPX286 Operating Systems Writer's Guide", Intel Corporation, 1983.

[Kasa85] H.Kasahara, S.Narita, "Parallel processing of robot-arm control computation on a multimicroprocessor system", IEEE Journal of Robotics and Automation, Vol RA-1, No.2, June 1985, 104-110.

[Saka87] K.Sakamura, "The TRON Project", IEEE Micro, April 1987, pp.8,14.

EASY HIGH LEVEL LANGUAGE BOOTSTRAPPING ON MULTIPROCESSOR SYSTEMS

M. Ancona(*,**), A. Clematis(*), V. Gianuzzi(**)

(*) Istituto per la Matematica Applicata C.N.R.
(**) Dipartimento di Matematica dell' Universita'
Via L.B. Alberti 4, 16132 GENOVA ITALY

Keywords: Embedded Systems, Kernel, Concurrent High Level Language, Modularity.

Abstract

A kernel implementation methodology is presented here, whose main goals are the following:
1) To supply an efficient high level language bootstrap tool on target machines without operating systems or without any kind of software development facilities;
2) To allow an easy transportability of the kernel from one target to another;
3) To simplify the design and to improve the reliability of kernels for multimicroprocessor systems.

We present also as an example a kernel implemented using MML (Multi Micro Programming Line [Boar84]), a concurrent programming language for multimicroprocessors. Such kernel acts as run time support for MML, including features such as: interrupt management, exception handling, timeslicing, preemptive and non preemptive process activation, remote call and synchronization. Using this kernel we obtained also a remarkable improvement in the language bootstrap time on a target under test.

1. Introduction

Nowadays microprocessor and multimicroprocessor based hardware systems are used in a wide range of applications such as industrial control, or as part of OEM's computer systems. The increasing complexity of applications using such systems leads to the use of high level languages to develop software, which is also useful in order to increase reliability and to reduce development time. The host target approach to develop software looks like a suitable one in this context.

Usually we may find a development system hosted on a mini or medium size computer composed at least by a compiler, a linker and a down loader to the target system. On the target, a kernel or run time system is necessary in order to execute application programs. The run time system is often designed following a server based approach with a kernel implemented in assembly language and server processes implemented in either high level languages or assembly language.

We note that many high level language support low level facilities for implementing operating system kernels (e.g. Modula-2, Edison and several Pascal extensions) but none of them, to the authors knowledge, resolve the retargetability problem with a target devoid of software development facilities. The retargetability requirement has been widely investigated from the host point of view, and different tecniques to obtain retargetability (e.g. for compilers and code generators [Glen78]) have been developed. On the contrary we have a lack of retargetability on the target due to the impossibility of writing the whole kernel in high level language.

Moreover it is important to point out that the kernel, in spite of its small size, reaches a high complexity level and the possibility of writing it in a high level language is a desirable one. Using a high level language we may deal with kernel complexity in a more reliable way and, at the same time, we tackle the problem of retargeting the run time system by using the host facilities.

We present a methodology to write a kernel using a high level language suitable also to simplify the language bootstrap on the target. This methodology is first presented in a general way, and then the MML programming language is introduced and used as a case study.

2. General method

The starting point of our work was the analysis of the features requested from the kernel by user processes, and of the typical implementation techniques used in assembly kernel implementation. A kernel must be able to receive requests from processes, dispatch them by giving them CPU control, and handle interrupts from devices and the clock. To keep track of the status of a suspended process and to recognize the service requested the kernel must use process descriptors, and it must be able to exchange information with the processes themselves.

The most suitable mechanism provided by microprocessor architectures to handle kernel services is the software trap, which implements the supervisor call issued by the high level language code generator. A second important aspect is the protection environment assigned to the two running levels of assembly code: master and slave. But the most evident advantage using assembly language, is the possibility of bootstrapping a high level language also on a target provided only with a simple downloading monitor. In this way, however, it is necessary to rewrite a specialized kernel whenever the architecture changes and to know in detail the language interface.

By a survey of well known concurrent programming languages [Appel85], it is possible to point out some useful features and other critical aspects of implementing a kernel for one of such languages, using the language itself. For example some of these useful features are:
- the possibility of defining a task or process type as a different object from procedure;
- realization of statical mapping of variables to allow I/O operations;
- definition of a binding between the locations at which the trap occurs and the portion of code written in high level language.

Some of these features are translated by the compiler in supervisor calls and then resolved by means of assembly routines. Furthermore, all these languages work under a well defined operating system, and the bootstrap problem has never been taken into great consideration.