A GENERALISED APPROACH TO MACHINE CONTROL

by

ROBERT HARRISON

A Doctoral Thesis
submitted in partial fulfilment of the
requirements for the award of

Doctor of Philosophy

of the Loughborough University of Technology
Department of Manufacturing Engineering

January 1991

DECLARATION

No part of the work described in this Thesis has been submitted in support of an application for any other degree or qualification of this or any other University or any other Institution of learning.

ACKNOWLEDGEMENTS

The author wishes to thank:

Professor R. H. Weston for his help, interest and encouragement.

Dr J. Pu, Dr G. G. Rogers, Dr P. R. Moore, Mr A. H. Booth, Mr A. Carrott and Mr A. S. Goh for their interest and friendship.

My parents for their help and support.

The SERC and Quin Systems Ltd for their financial and technical assistance.

ABSTRACT

The intrinsic complexity and variability of typical real time control problems makes a generalised approach to producing control systems difficult to specify. Due to a lack of standardisation, current machine controllers are usually extremely difficult to configure, support and integrate together in a generalised manner. These problems have severely hindered the development and subsequent application of advanced factory automation. The exploitation of advanced computer technology, particularly modern software methods can now enable a consistent machine control structure to be maintained for diverse applications of widely differing complexity. This thesis addresses the need for a major change in the design of machine control systems and proposes the use of a reference architecture which offers a consistent approach to the control of real time industrial operations.

A broad based look at existing control systems focuses on the functionality they currently offer in the control of various categories of industrial operations. A study of current manufacturing automation highlights the functional similarities between the control requirements of different industrial processes both in terms of their control structure and hierarchical communication requirements for factory integration. Given this commonality it is proposed that all industrial controllers should logically be based upon a common hardware independent architecture. A design methodology has been devised, termed Universal Machine Control (UMC) which enables individual machine controllers to be created (with functionality closely matched to their specific applications) whilst still maintaining common structural and communications features. This methodology aims to simplify the process of defining, programming and controlling systems built up from user defined mechanical hardware. A modular design framework or reference architecture for machine control has been derived which allows control systems to be modelled in a generalised manner.

A particular implementation of the control architecture conforming to this reference model and an associated definition environment have been created. The implementation is based on the selective use of modern computer methods and emerging standards for real-time control. A demonstration system has been produced targeted at the flexible assembly of printed circuit boards. The possible application areas for this control philosophy are however extremely diverse and it could have a significant impact on automation methods.

Contents

1
Introduction

Although still in its infancy industrial automation is becoming progressively more widespread with the need to reduce product costs through improved efficiency and better quality control. There is now a widely held belief that the key to advanced automation is the integration of systems throughout business from boardroom to shopfloor. The subject of this thesis is the real-time control of industrial machines which form the foundation of this industrial hierarchy.

Industrial machine and process control is extremely diverse both in form and function. The tasks performed can be simple or complex and may either partly or fully automate manufacturing operations. These diverse control requirements have led to a wide range of controller types often applied in an ad hoc manner with very little standardisation between them. As a result industrial real-time control systems are difficult to maintain, modify and integrate.

In addition to localised real-time control, machine control systems have a vital but as yet largely unrealised role to play in computer integrated manufacturing (CIM) by enabling information access and exchange with higher factory levels. Typically enhanced machine control systems are required with appropriate additional functionality to achieve this information interchange. Machine controllers can no longer be regarded and evaluated as discrete items of plant but as vital building elements of factory systems.

The Modular Systems Group at Loughborough has worked for some eight years on designing and producing modular distributed manipulators and investigating methods of applying them in flexible manufacturing. The concept behind this work is to enable programmable machines to be easily created to cope with any given application. The approach advocates the use of both distributed manipulation and processing elements which are an integral part of the required machine and in turn a given factory configuration. Actuator groups of appropriate type and performance are used where they are actually needed and support as much - but no more - functional complexity than the various application tasks actually demand. This form of mechatronics approach to automation is now becoming a widely researched field of study. Central to the research into configurable manipulators at Loughborough

has been the desire to create a highly configurable machine control system. By embodying a generic structure and extendible functionality such a modular control system would potentially be able to meet in a consistent manner, diverse machine and process control requirements.

This thesis addresses the need for a machine control strategy offering a major change in methodology from present methods. The aim is to overcome the lack of standardisation which currently blocks advanced machine automation, encourages inefficiency and discourages future investment. The intrinsic complexity and variability of typical real-time control problems makes a generalised approach difficult to specify. The exploitation of advanced research into information technology, particularly modern software methods now offers the potential to maintain a consistent machine control structure for diverse applications which intrinsically involve widely differing complexity. The evolution of an effective approach to generalised control has required:

(1) evaluating the nature of the control problem,

(2) assessing the capabilities of the enabling technology and

(3) selection of a new approach which is dominated by the problem rather than being handicapped by the technological origins which hinder the implementation of many current machine and process controllers.

This decision process forms the basis for the construction of the thesis:

Chapter two broadly establishes the role of real-time machine control in industry and evaluates what attributes such control systems should possess in order to interface effectively with a CIM environment.

Chapter three looks at the origin and capabilities of different forms of industrial controller which are in widespread use today. The historical changes in enabling technology are reviewed and their effect on the configuration and capabilities of current controllers is considered. The strengths and weaknesses of current industrial controller designs are assessed with reference to available customer and vendor perceptions of their future needs. From this assessment a set of requirements are derived for a more consistent approach to diverse control problems.

Chapter four reviews the state of the art in problem oriented approaches to both control system architectures and methods of application program description.

Chapter five endeavours to realise a generalised approach to diverse control problems by advocating a Universal Machine Control (UMC) methodology. An essential model for a Reference Architecture (RA) to support generalised machine control is proposed.

The enabling technology for the implementation of generalised machine control systems is assessed in chapter six. Conventional low level coding methods are contrasted with the use of an operating system as the environment for implementing real-time control. Modular computer hardware is assessed for the construction of configurable physical control system architectures.

From the creation of an essential model for a more generalised approach to machine control in chapter five and the evaluation of enabling technology in chapter six it emerges that the opportunity now exists for problem dominated approaches to the engineering of real-time process and machine control systems.

Chapter seven looks at the practical implementation of a proof of concept Reference Architecture for the control of modular distributed manipulators. The chapter illustrates how the functionality offered by a real-time operating system is used to implement process and data structures conforming to the UMC methodology.

The implementation (software design, structure and coding) of selected UMC software modules is detailed in chapter eight.

Chapter nine considers the capabilities of the current implementation of UMC, its perceived advantages, limitations, industrial potential and possible routes to its exploitation.

Future extensions to the UMC methodology are suggested in chapter ten to enable it to cater for the design and life cycle requirements of machine systems in a broader manner.

Conclusions and a resume of the author's contributions to knowledge are presented in chapter eleven.

2
The Role of Machine Control in Manufacturing

2.1 INTRODUCTION

This chapter seeks to broadly establish the role of real-time machine control in flexible manufacturing. A layered model representing the manufacturing hierarchy is utilised together with a broad classification of industrial manufacturing operations. An evaluation is made of what attributes control systems should possess in order to help manufacturers to compete effectively in today's highly volatile business environment.

2.2 AUTOMATION OF MANUFACTURING

In the post war period there has been a demand within Western markets for manufactured goods in high volumes. Recent years have however seen the decline of many traditional industries in Europe and North America in an ever more competitive world environment [1].

Factory automation was initially applied to highly repetitive tasks, continuous process operations being the primary example of this. Such cyclic operations are predictable and thus relatively easy to cope with [2]. Automation has developed progressively from its origins in the manufacture of primarily "non-shaped" products (*process automation*) to products having "shapes" (*mechanical automation*), generally requiring the handling of discrete parts in contrast to a continuous flow of materials [3]. Machines can now enable many products to be made in large volumes in a short time.

In the West the market for volume manufactured goods is now maturing and the desires of the consumer have changed considerably. There is now a perceived need for product diversification in manufacturing to meet increasing customer demand for choice [4]. This requirement for product diversification must be reflected in the engineering of manufacturing systems. The emphasis when assessing manufacturing facilities must shift from purely measuring production rate as a figure of merit to assessing responsiveness to change [5]. The lack of responsiveness of

modern factories is now perceived as a serious problem. World-wide total industrial productivity is now approaching or even exceeding demand in a number of industrial sectors [6]. Items which the customer does not require represent waste however efficiently they are produced.

New production systems must be able to respond quickly to changes in customer demand [7]. The emphasis is now changing from how to manufacture to what, when and how many to manufacture. The pressures on a modern production system are summarised in figure 2.1. The form of future manufacturing systems is an important consideration in enabling factories to be more responsive to customer demand. Computer integrated manufacturing (CIM) is seen as having the potential to revitalise many industries but its practical realisation remains limited [8]. The core concept of CIM is total optimisation in systems engineering. A CIM system must however be well matched and interfaced to its environment or it may in itself be a recipe for chaos [9].

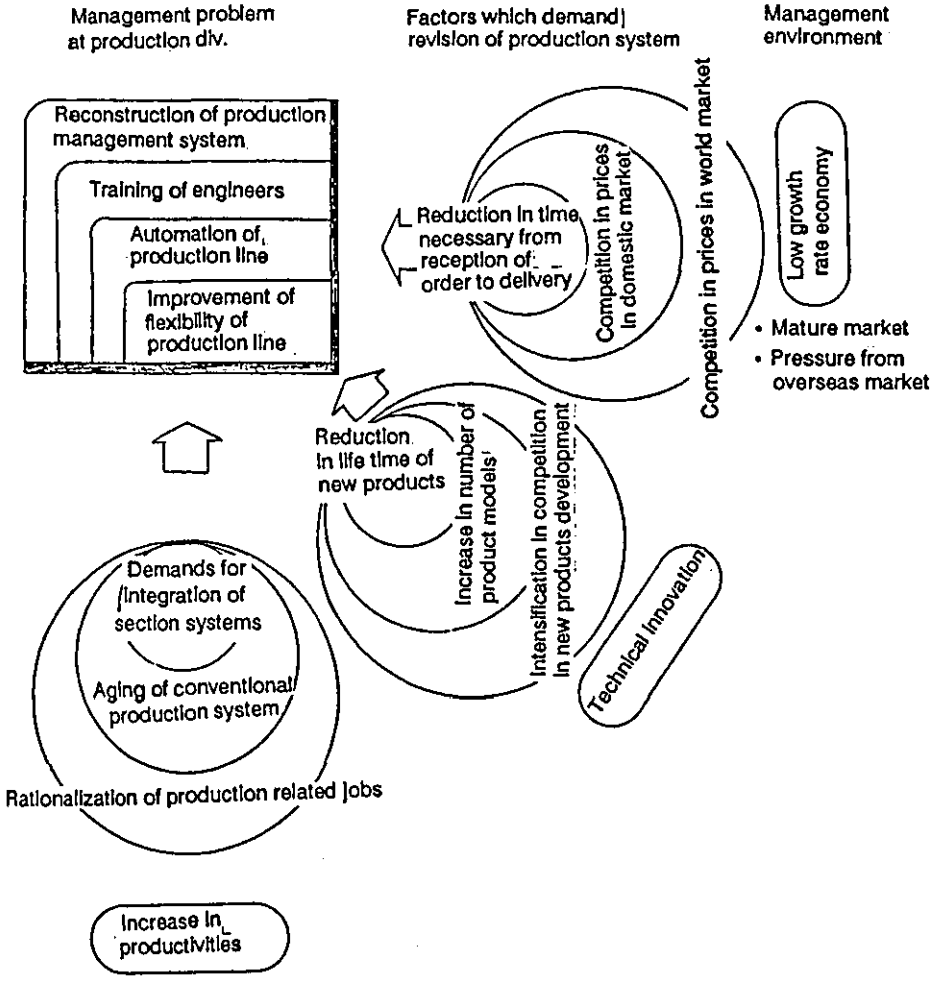


Figure 2.1. Pressures on a modern production system. Source: Suda.

It is in the context of a responsive manufacturing environment that the role and requirements for future real-time machine control systems must be evaluated. Real-time machine control tasks form the automated plant interface at the base level of the manufacturing hierarchy and their correct implementation is thus of vital importance.

2.3 THE MANUFACTURING ENVIRONMENT

Industries commonly require processes of different types to be integrated together within the same factory, on the same production line or indeed as parts of the same machine. This section considers the role of CIM systems in the context of a responsive factory structure and its implications on machine control.

2.3.1 Functional Hierarchy

Conceptual representations of the required levels in a manufacturing environment have been developed by several research groups [10].

A simplistic representation of the industrial functional hierarchy for all types of automation has been suggested by Bernard [11] and is shown in figure 2.2. Functions are subdivided into three groups: business functions at the higher levels, manufacturing at the lower levels and safety functions at the bottom level. This thesis addresses the requirements for real-time machine control which encompasses levels one to three of the CIM environment shown in figure 2.2. The NBS information hierarchy for discrete manufacturing plants [12] is shown in figure 2.4. The hierarchy proposed by the Mitsubishi Electrical Corp. for continuous process plants [13] is illustrated in figure 2.3. Both diagrams illustrate the progressive variations in the time and information domains with real-time devices frequently exchanging relatively small quantities of information. It is interesting to note the similarities in these architectures which encompass widely differing types of industrial automation.

2.3.2 The Need for Integration

As explained by Ekong [14] and many others [15, 16], CIM requires the ability to integrate vertically between factory levels from planning down to the control of real-time operations in a flexible manner. This type of interrelationship is depicted in figure 2.5. There is also an increasing need for horizontal integration to directly link one activity or operation with another, typically where synchronisation is

required [17]. See figure 2.6. The requirement for integration has profound effects on the functionality required at the real-time control levels in order to integrate with and adequately service the higher factory levels [18].

LEVEL	TYPES OF INDUSTRIAL AUTOMATION			
	CONTINUOUS	DISCONTINUOUS	BATCH	DISCRETE
	BUSINESS FUNCTIONS			
5 CORPORATE PLANNING	- Economic Forecasting - Market and Production Planning - Decision Support Systems			
4 TACTICAL PLANNING	- Process Modelling and Simulation - Economic Sensitivity Analysis - Accounting and Finance			
3 TECHNICAL SUPPORT	- Management Information Systems - Computer Aided Design - Plant Maintenance Systems			
2 PRODUCTION PLANNING	- Scheduling - Batch Management - Material Requirements Planning		- Purchasing - Inventory Control	
1 PRODUCTION CONTROL	- Order Entry and Tracking - Labour/Payroll		- Material Utilisation - Shipping and Billing	
	MANUFACTURING FUNCTIONS			
5 DETERMINE BEST OPERATING CONDITIONS	- Optimise Equipment Utilisation - Energy Management Systems - Computer Aided Manufacture			
4 QUALITY ASSURANCE	- Composition Measurement Systems - Statistical Quality Control - Automated Inspection Systems			
3 MOVE OPERATING CONDITIONS	- Supervisory Control	- Sequence Control		- Cell Control
2 HOLD DESIRED OPERATING CONDITIONS	- Adaptive Control - Multivariable Control - Feedforward Control		- Model Reference Control	- Workstation Control
1 MEASUREMENT AND ACTUATION	- Feedback Control	- Logic Control		- Machine Control
	SAFETY FUNCTIONS			
0 HUMAN AND PLANT PROTECTION	- Automatic Safety Systems			- Manual Shutdown

Figure 2.2. Simplistic representation of the industrial functional hierarchy for all types of automation. Source: Bernard.

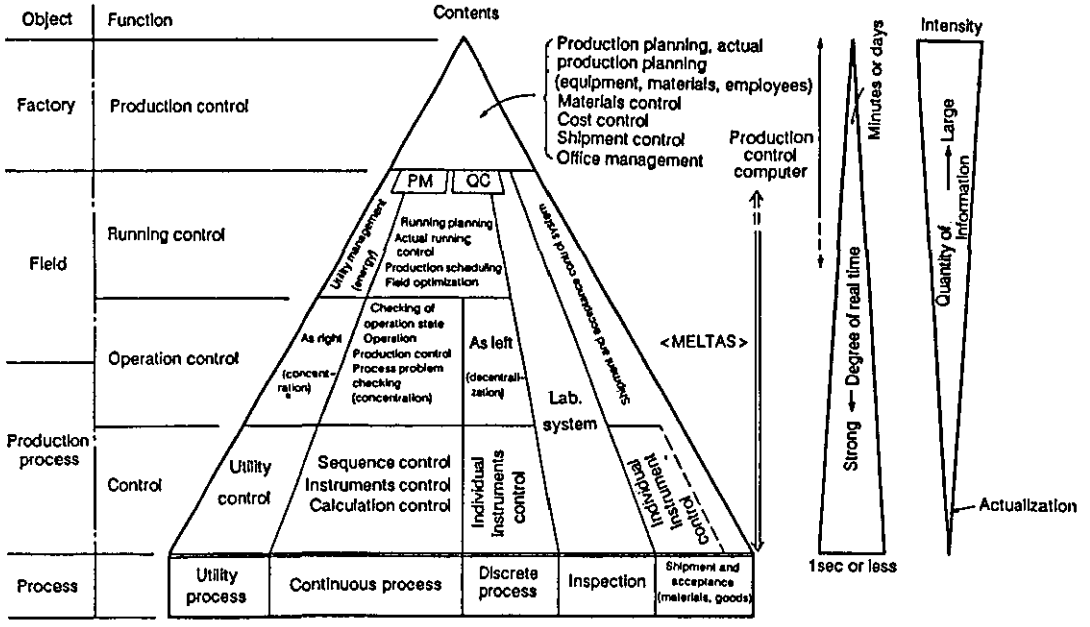


Figure 2.3. Hierarchy proposed by the Misubishi Electrical Corp. for a mixed operations plant. Source: Misubishi Denki Giho.

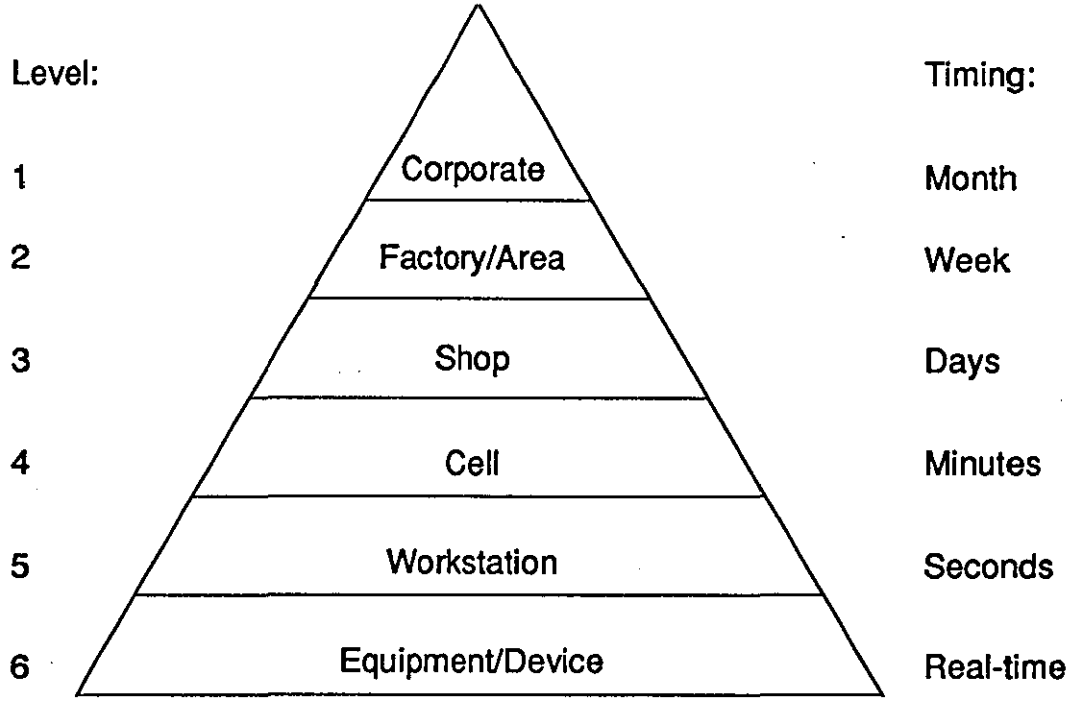


Figure 2.4. NBS architecture for discrete manufacturing plants.

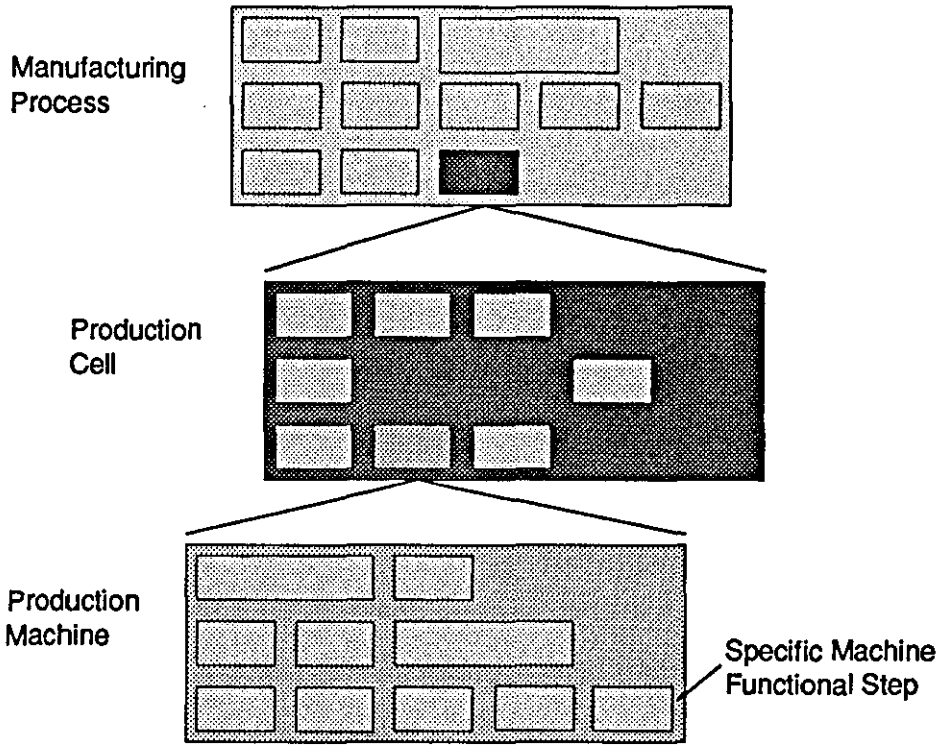


Figure 2.5 Integrate vertically between factory levels.

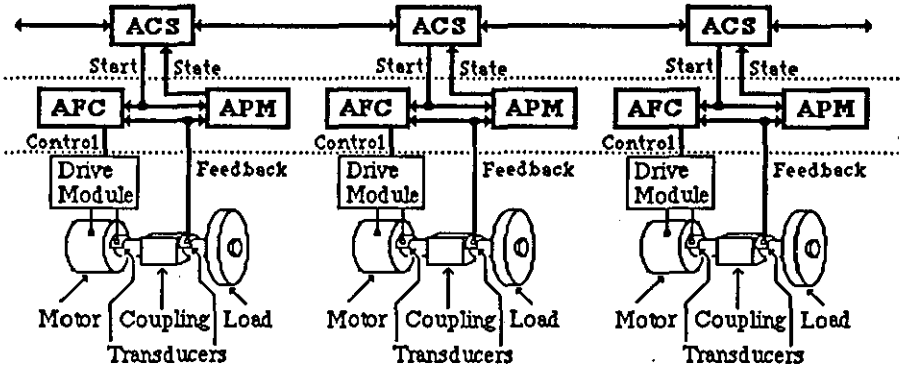


Figure 2.6. Horizontal integration to directly link one activity or operation to another. Source: Fenney.

2.4 CONTROL OF MANUFACTURING OPERATIONS

This section looks at the types of manufacturing operations where real-time machine control systems are typically applied. With the aid of some examples the functional requirements for the control of widely differing manufacturing operations will be considered.

2.4.1 Characterisation of Real-Time Machine Control Systems

Industrial machine control systems are extremely diverse both in form and function making a simple definition of real-time machine control difficult [19].

A real-time control system is characterised by being connected via physical devices to processes which are external to it. These external processes will operate in their own time scales and the controller is said to operate in real-time if the tasks carried out within the controller relate to the time scales of the external processes [20].

Appreciation of machine control varies widely between different engineering and scientific disciplines involved in manufacturing automation due commonly to differences in background and hence intrinsic vocabularies [21].

There are no fixed boundaries between the types of control appropriate for process or mechanical automation [22, 23, 24] although there is much confusion in both terminology and methods surrounding the assessment, selection and application of real-time control systems [25]. See chapter three. Figure 2.7 illustrates some of the commonly used sub-divisions across the manufacturing spectrum.

Market forces now dictate the creation of more flexible manufacturing systems, particularly for the mechanical automation of the lower volume discrete parts manufacturing and job shop environments [7]. The focus of this thesis is on the real-time control of mechanical systems for such discrete parts handling. The origin of the work is in modular manipulator control which has been a field of intense research in the Department of Manufacturing Engineering over the past ten years [26]. See section 7.3.5. As will emerge later however, the concepts discussed in chapter four and the methodology proposed in chapter five are more generally applicable to both mechanical and process automation.

2.4.2 Types of Manufacturing Operations

One established method of classifying industrial operations [27] is to segment the major industries into continuous, discontinuous, batch and discrete units of operations as shown in figure 2.8.

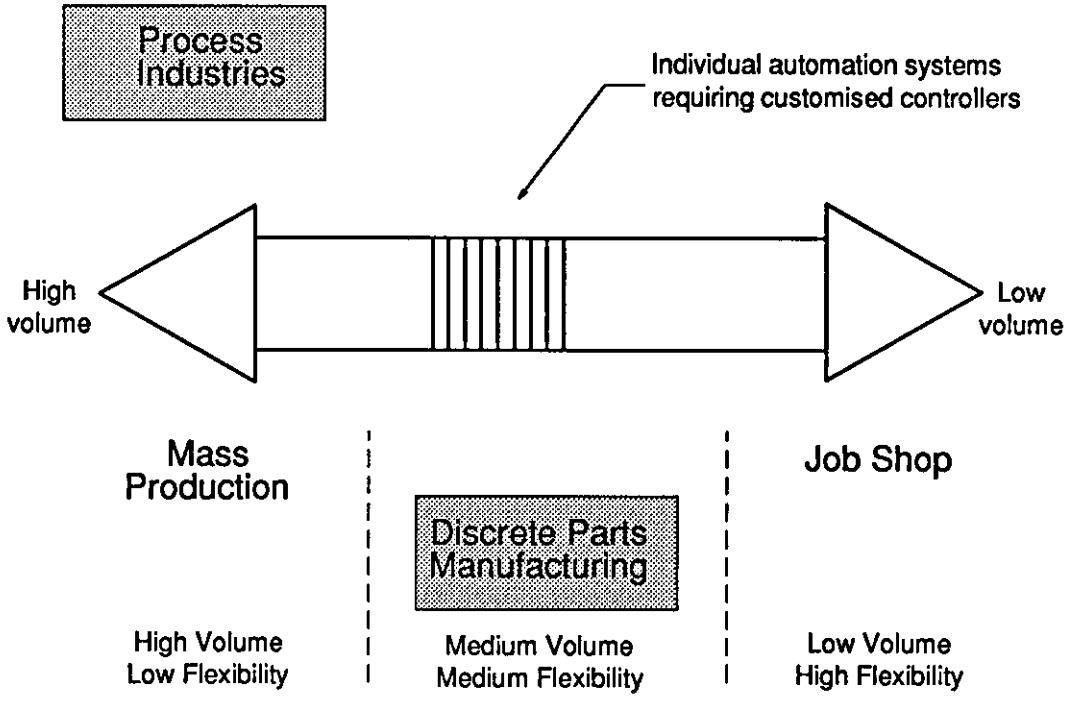


Figure 2.7. Commonly used sub-divisions across the manufacturing spectrum. Source: Greenwood.

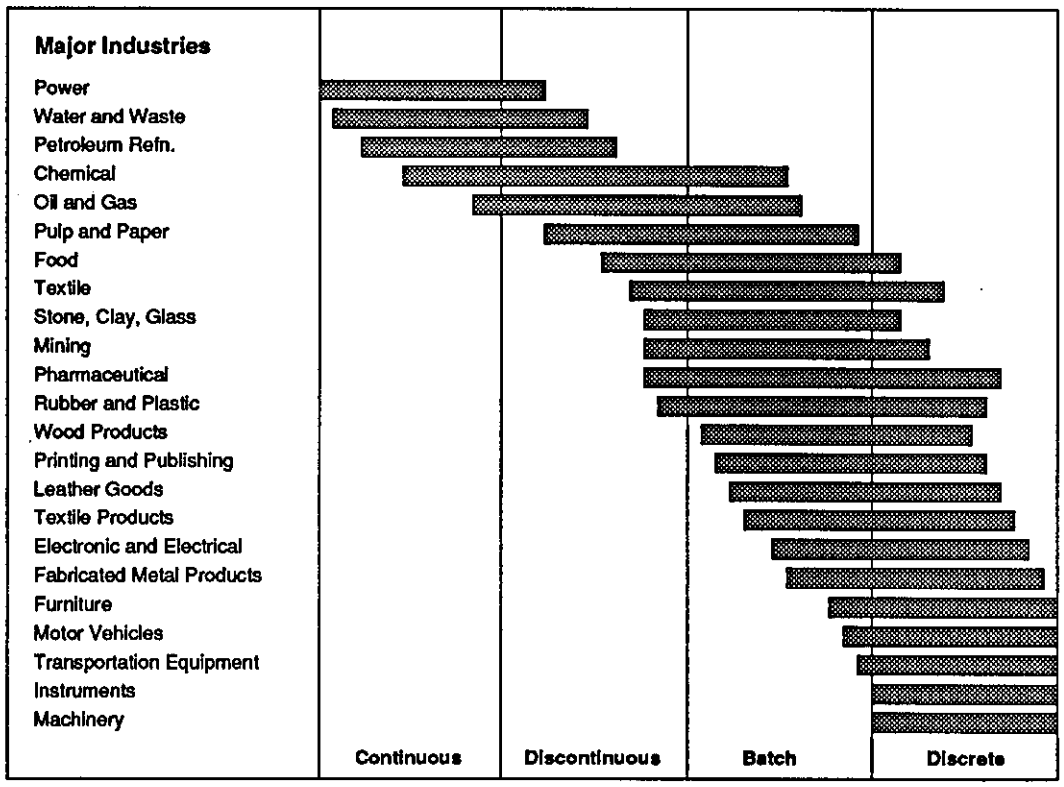


Figure 2.8. Segmentation of the major industries into continuous, discontinuous, batch and discrete units of operations. Source: Bernard.

Continuous operations are those where the raw materials, intermediates, and final products are processed in continuous fashion for long periods of time. These operations are typified by the petrochemical and water industries [20]. A typical example would be a large distillation operation as used in many petroleum and chemical separations.

Discontinuous operations, when running, are the same as continuous operations with the exception that they are frequently changed from one product to another [27]. Such changes may occur every few hours or days. Examples of such operations are paper making machines and steel rolling mills.

Batch operations are somewhat different in that the processing is done in a specified sequence over a given trajectory of operating conditions, such as temperature and pressure [28]. Batch is the oldest form of unit operation and is still used extensively in many industries such as food, glass and textiles. Many electronic and metal parts are also processed in batches.

Discrete processes are those which produce one product at a time, such as motor cars or cookers [29]. Such processes use an assembly line where the product moves through the various operations, or the product may be relatively stationary with different processing steps done at the same location. A vast array of products are manufactured by a series of discrete processes. These include computers, domestic appliances, furniture and toys.

Job shop manufacture is a term often associated with production of discrete items in low volumes. It is characterised by small lot production of varied products, typically with a high manual labour content [6].

This is a generalised classification of manufacturing operations and many industries will to some extent combine these categories of operations in their manufacturing cycle [30].

2.4.3 Automation of Diverse Manufacturing Operations

Automation has been widely applied to high volume products [31] in most types of manufacturing operations. The need for more responsive production of smaller volumes has promoted a strong interest in small batch production and more flexible automation [32].

At the two extremes of manufacturing operations are continuous processes and discrete parts handling. While the control requirements may at first sight appear very different in these sectors of industry much of the required controller functionality is the same [33]. This commonality is illustrated by considering the required functionality for machine controllers in a plant which embodies a board mix of manufacturing operations.

Figure 2.9 is a schematic for such a machine line which is typical of many in the food and glass industries [34]. The product is initially processed in continuous form, then divided into discrete items for the final stages of manufacture and packaging. This fictitious production line can be divided into typical machine sections. Four typical manufacturing operations have been chosen for the example machines illustrated in figure 2.9.

Table 2.1 summarises the functionality that might typically be required for each of the chosen manufacturing operations. It can be seen from the table that:

- Varied physical requirements are imposed by the form of each machine.
- A separate logical structure is imposed by the particular control requirements of each machine.
- Control functions are generally a combination of concurrently operating tasks which can be open and closed loop controlled.
- External communications requirements typically consist of services for both machine synchronisation and information exchange with higher factory levels.

These examples illustrate that although machine operations may be distinctly different there are generally many common functional requirements for the machine control systems. It is the scale of the machines, mix of required functions and response times which commonly vary [35]. There will also typically be differences in operational considerations such as the degree of fault tolerance or the maintainability required [36].

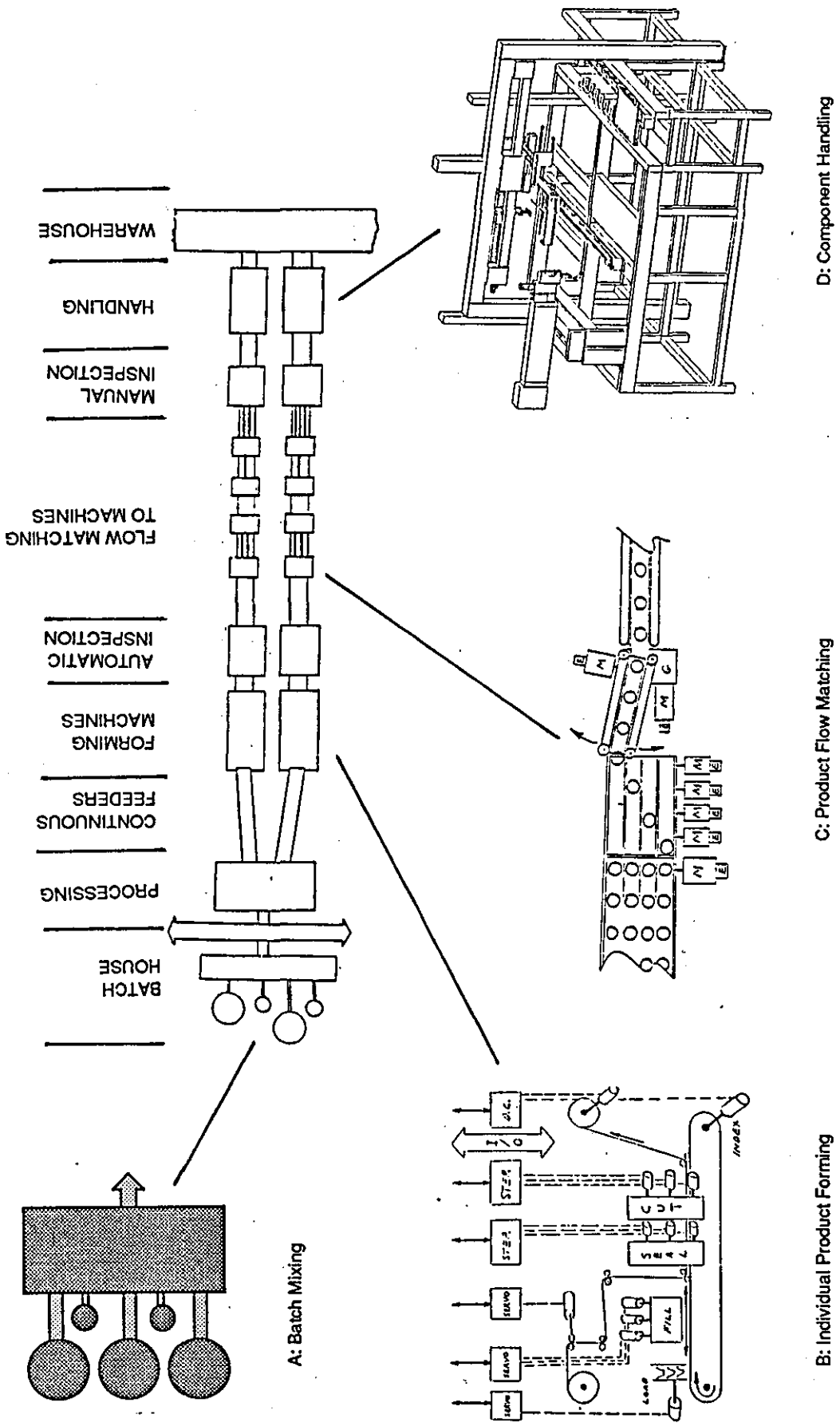


Figure 2.9. Schematic for machine line.

Operation	Continuous Batch Wet Process	Continuous Batch Dry Process	Discrete Disordered	Discrete Regular
Example	Food Mixing	Wrapping	Sorting	Packaging
Physical Distribution	10-100m	5-20m	1-10m	1-10m
Time Response	0.1-1sec	0.001-0.01sec	0.001-0.01sec	0.004-0.04sec
Control Type	Combination of Concurrent Closed Loop and Discrete Event Tasks			
Comments	Mix Optimisation	Programmable Gearbox	Interrupt Biased	Sequential
Communications	I/O Interfacing and Machine Integration			
Operator Interface	Centralised Operator Interface for Each Machine			

Table 2.1. Typical machine functionality.

2.5 MACHINE CONTROL IN A RESPONSIVE MANUFACTURING ENVIRONMENT

This section looks at the necessary attributes which machine control systems should embody in order to support an integrated manufacturing environment which will be responsive to change. A set of desired capabilities for flexibly automated machines suggested by Weston [37] are listed below:

(1) *perform more quickly* (i.e. shorter cycle-times to improve responsiveness)

(2) *facilitate fast product changes and interact with operatives in a generally efficient manner* (i.e. to enable small batch working and operations)

(3) *perform more accurately* (to improve quality, yields and down time)

(4) *facilitate pre-, in- and post-process inspection and/or test* (again to improve quality, yields and down time as well as avoiding the further processing of faulty products)

(5) *include error recovery procedures* (to provide similar benefits to (4))

(6) *include comprehensive diagnostic/maintenance facilities* (mainly to simplify support of the technology)

(7) *enable efficient planning and manufacturing control* (to interact with supervisory planning and control systems to yield short planning horizons and thus high levels of efficiency and responsiveness coupled with low inventory levels).

(8) *enable efficient logistic functions* (to interact with archiving and materials control systems as well as planning systems, to reduce inventory, improve quality and enable improved process control)

(9) *facilitate "integratability"*, (in the sense that where possible vertical and horizontal integration should be enabled as appropriate so that the inherent overheads of time and cost which result from separating out organisation, planning, control and manufacturing functions can be reduced).

In order to meet the requirements of any given application in a flexible manner it is necessary to optimise machine capabilities. A machine controller must therefore be capable of changing the number, type and interaction between the functions it offers [38]. This identifies the need to offer *"configurability"* in order to:

(1) install machine controllers in diverse manufacturing applications, and;

(2) modify/enhance their functionality in a given application as product changes and/or required operating practices dictate.

Due to the great diversity of manufacturing operations the functionality required for a specific machine controller is obviously very much application dependent as discussed in section 2.4. However although each machine may be of a unique configuration there will typically be many common real-time and communication functions required by most machines particularly within a common industrial sector [14]. The essential need is therefore the provision of the correct combination of real-time machine control related functionality to suit each particular machine in a flexible manner.

Factory integration requires the cooperation of many manufacturing entities which of course include machine control systems. The functionality of the overall manufacturing system depends on effective information exchange between these entities. Maximising the external visibility of machine control information is thus important in supporting a responsive manufacturing environment. The provision of this external visibility combined with a suitable CIM infrastructure is necessary to enable effective information exchange [39].

An effective infrastructure is required to support vertical and horizontal integration both above and below the machine controller level. Machine controllers

need to integrate both dumb and intelligent sensing and actuating devices (position sensors, closed loop drives, vision systems etc). A supporting infrastructure is necessary to achieve this [40]. Similarly integration of machine controllers themselves as entities within the wider CIM infrastructure requires the use of methods which are compatible with wider integration philosophies [41].

The implementation of flexible rather than hard automation offers the potential for companies to adopt efficient production methods whilst still remaining adaptable enough to cope with frequent product changes and enhancements. As suggested by Weston and others [42] the role of the machine control system in this type of responsive manufacturing environment is to provide both local real-time machine control capabilities, and the global machine related needs of integrated manufacturing in a flexible manner. The shift to more responsive approaches to automation brings with it the need to look closely at the wider issue of methodologies for the design of machines and machine control systems [43]. It requires the ability for machine control systems to support the reconfiguration of production equipment in a structured manner. Support and design environments are required to enable both run-time and longer term configuration changes in order to adapt efficiently to product modification and change. See table 2.2.

2.6 SUMMARY

As product life cycles continue to fall, todays manufacturing environment needs to become more responsive to change. An important aspect of improving manufacturing responsiveness is the provision of more flexible production machinery and associated machine control systems. Machine control systems must embody adequate real-time machine control and integration capabilities. This functionality needs to be provided in a flexible manner through the provision of adequate design, configuration and run time support environments. The next chapter will assess the manner and degree to which current control systems are capable of fulfilling these requirements.

	CONTROLLER ATTRIBUTE	REQUIRED CAPABILITY
LOCAL MACHINE/PROCESS RELATED REQUIREMENTS:	REAL-TIME CONTROL CAPABILITIES TO MAXIMISE MACHINE PERFORMANCE	PERFORM MORE QUICKLY
		PERFORM MORE ACCURATELY
	MAXIMISE CONTROL SYSTEM FUNCTIONAL VISIBILITY	INCLUDE COMPREHENSIVE DIAGNOSTIC/MAIN-TENANCE FACILITIES
	FUNCTIONAL FLEXIBILITY	FACILITATE PRE-, IN- AND POST-PROCESS INSPECTION AND/OR TEST
		INCLUDE ERROR RECOVERY PROCEDURES
	PROBLEM ORIENTED OPERATOR INTERFACE	FACILITATE FAST PRODUCT CHANGES AND INTERACT WITH OPERATIVES IN A GENERALLY EFFICIENT MANNER
GLOBAL MANUFACTURING RELATED REQUIREMENTS:	MAXIMISE INFORMATION VISIBILITY	ENABLE EFFICIENT PLANNING AND MANUFACTURING CONTROL
		ENABLE EFFICIENT LOGISTIC FUNCTIONS
DESIGN REQUIREMENTS:	EFFECTIVE ARCHITECTURE AND SYSTEM BUILDING TOOLS	FACILITATE "INTEGRATABILITY"
		FACILITATE "CONFIGURABILITY"

Table 2.2. Required Machine Control System Capabilities

Chapter 2: References

1. Kendrick J.W., (ed), "International Comparisons of Productivity and Causes of the Slow-Down", Ballinger, 1984.

2. Anon., "Computer Software for Process Control", Report of the Working Party on Real-time Computer Software, May-December 1968, Plant Engineering and Energy Division, BISRA.

3. Riley, J.R., "Building Flexibility Into Your Assembly Systems", Assembly Engineering, Hitchcock Publications, November 1983.

4. Baer, T., "CIM: Avoiding Future Shock", Managing Automation, Thomas Publishing Company, October 1986.

5. Williamson, I.P., "Integrated Manufacturing: Developing Your Strategy", Proc. 5th CIM Europe Conf., May 17-19, pp 335-346, IFS Publications, 1989.

6. Suda, H., "Future Factory System Formulated in Japan", Japanese J. Adv. Automation Tech. Vol.1 pp 67-76, Fuji Technology Press, Sept. 1989.

7. Carlisle, B.R., "The Changing Nature of Assembly Automation", 10th Int. Conf on Assembly Automation, Japan, 1989.

8. Halbrecht, H.Z., Nostrand, J.W. "Is Management to Blame for the Unfulfilled Dream of CIM?", Production Engineering, July 1987.

9. Knill, B., Weimer, G. "Systems Integrators:Helping to Organise a Vast and Complex Array", Production Engineering, April 1987.

10. Van Dyke Paranuk, H., White, J.F. "A Framework for Comparing Factory Reference Models", Proc. of the Spring 1987 Int. Purdue Workshop on Industrial Computer Systems, 1987.

11. Bernard, J.W., "MAP/TOP Promise or Practice", Proc. of 16th ESD/SMI INT Programmable Controllers Conf.,pp 1 to 18, April 1987.

12. Lui, D., "Factory Management Architecture and Information Infrastructure", Proc. on Factory Standards Model Conference, National Bureau of Standards, November 1985.

13. Hyodo, T., "The Situation in Integrated Automation-Control Systems for Industry", pp 2 to 4, Vol.63, No.5, Misubishi Denki Giho Publishing, Tokyo, Japan, 1989.

14. Ekong, E.S., "Controls: The Bridge to Systems Integration", Proc. of 16th ESD/SMI INT Programmable Controllers Conf.,pp 49 to 57, April 1987.

15. Simpson, J.A. et al., "The Automated Manufacturing Research Facility of the National Bureau of Standards. ", Journal of Manufacturing Systems, 1982, Vol.1:1, pp 17-32.

16. Mayer, C., "ICAM MCS Architecture and its Implementation", Proc. on Factory Standards Model Conference, National Bureau of Standards, November 1986.

17. Sides, K., "A Factory Floor CIM Applications Architecture", Proc. of 7th Annual Control Engineering Conf., 1988, pp17-42.

18. Rubin, S.E., "Sliding up the Bannister of Technology", Proc. of 7th Annual Control Engineering Conf.,1988, section VII, pp 6-13.

19. Groover, M.P., "Automation, Production Systems and Computer Integrated Manufacturing", Part VII, Control Systems, Prentice-Hall, 1987, pp 521-671.

20. Bennet, S., "Real-Time Computer Control", Prentice-Hall, 1988.

21. Ward, P.T. and Mellor, S.J., "Structured Developments for Real-Time Systems", Vol 1, Yourdon Press, 1985, pp 10-13.

22. Bridge, A., "PC or PLC - That is the Question", Industrial Computing, September 1986, p 39.

23. Dunbar, R.D., "The Use of Programmable Controllers for Batch Process Control", pp 193-212.

24. Readman, G., "Is it a Controller or an Industrial Computer?", Control and Instrumentation, January 1987, pp 39-40.

25. Anon., "Programmable Controller Market Assessment", Frost and Sullivan, 1986.

26. Weston,R.H. et al.,"Universal machine control system primatives for modular distributed manipulator systems", Int. J. Prod. Res., 1989, Vol. 278, No. 3, pp 395-410.

27. Bernard, J.W., "MAP/TOP Promise or Practice", Proc. of 16th ESD/SMI INT Programmable Controllers Conf., April 1987, p5.

28 Morris, H. M., "Batch Applications Proliferate: Control System Manufacturers Respond", Control Engineering, Vol. 33, July 1986, pp 54-57.

29 Considine, D., "Patterns of Industrial Production", Standard Handbook of Industrial Automation, Chapman and Hall, 1986, pp 5-6.

30. Anon.,"Integrated Automation-Control System for the Food Industry", Japanese J. Adv. Automation Tech. Vol.1 C-7 pp 91-93, Fuji Technology Press, Sept. 1989.

31 Northcott, J., "The Impact of Microelectronics", Section 3, Patterns of Diffusion, PSI Research Report 673, Pinter 1988, pp 43-55.

32. Anon., "Outline of 5th Report on Future Factory Systems", Japanese J. Adv. Automation Tech., Vol.1, Fuji Technology Press, September 1989, pp 68-84.

33. Shannon, J.H., "Complex PLC Control Tools: What they mean to the User", Proc. of 6th Annual Control Engineering Conf.,Rosemount, IL.,1987, pp 470-475.

34. Salihi A. and Weston R.H., "Distributed computer control and monitoring of glass container manufacturing plant", Glass Technology, Vol. 25, No. 1, February 1984, pp 38-43.

35 Ulsoy, A.G.and DeVries, W.R. "Microcomputer Applications in Manufacturing", John Wiley, 1989.

36. Kopetz, H., "Distributed Fault-Tolerant Real-Time Systems", IEEE MICRO, February 1989, pp 25-40.

37. Weston R.H., "ACME Automation Strategy Advisory Document RHW1//ACME/RA1-3//23/FEB/89", Manufacturing Engineering, Loughborough University, February 1989.

38 Weston R.H., "Generally Applicable Cell Controllers and Examples of Their Use", Commissioned chapter of book on "Cell Control" to be published late 1990 by Springer V., Ed. D Williams and P Rogers.

39. Weston, R.H. et al., "Integration Tools Based on OSI Networks", Autofact October 1989, Detroit, Michigan, SME Technical Paper MS89-708.

40 Scholten, C.A., "The Missing Link in CAM Systems", Paper CFT-B-35/87EN, 1987, Philips Centre for Manufacturing Technology, 5600 MD Eindhoven, The Netherlands.

41. Klevers, T. "The European approach to an 'Open system architecture' for CIM", Proc. 5th CIM Europe Conf., May 1989, pp 109-120.

42. Stringer, H., "Industrial Computers in machine control", Proc. 2nd Int. Conf. Machine Control Systems, May 1987, IFS, pp 139-164.

43 Rogers, G., "Modular Production Systems: A Motion Control Scheme for Actuators", PhD. Thesis, Loughborough University, 1990.

3
Current Practices in Machine Control

3.1 INTRODUCTION

Chapter two has considered what the role of machine control systems should be in flexible manufacturing. This chapter aims to evaluate how well current machine control systems fulfil this role. The origin and characteristics of computer based machine control systems are investigated and an assessment is made of the strengths and weaknesses of current designs. Where possible this assessment references user and vendor perceptions of both their current control systems and future needs.

3.2 THE EVOLUTION OF MACHINE CONTROL METHODS

3.2.1 Introduction

In an industrial sector which is very much driven by user demand [1], an appreciation of how control technology has evolved is important since traditional techniques continue to have an enormous influence on the form of modern control systems [2].

Prior to the middle of this century the standard approach to machine control was purely mechanical. This typically involved the use of linkages, gears and cams which were an integral part of any given machine [3]. Tremendous expertise has been attained in the field of mechanical control and it is now a very mature subject area [4]. There are many examples of programmable machines dating back more than one hundred and fifty years including textile looms and automatic piano players. As early as 1725 knitting machines were controlled with punch cards. These devices represent considerable technical achievement for their time [5]. The first automatic feedback controller used in industry is generally agreed to have been James Watt's fly-ball governor applied in 1769 to control the speed of a steam engine [6].

A great diversity of both discrete and continuous control systems have been implemented mechanically. Cam shafts, linkages and peg-in-hole mechanisms have been developed to implement sequential and combinational logic [7]. The speed of response of such mechanisms is obviously limited by dynamic constraints [8, 9].

With the development of electrical engineering early this century binary sensors, solenoids and relays became available as building elements to implement combinational logic circuits [10]. "Fast" combinational logic could be implemented with relays relatively easily and circuit rewiring was quick in comparison to mechanical redesign.

In the continuous process industries analogue control was commonly implemented using pneumatic single-loop controllers in conjunction with relay operated alarm circuits [11]. This type of technology predominated until about 1960.

The advent of semiconductors saw the use of transistors and later integrated circuits for binary and analogue control improving the speed of response, reliability and ease of circuit modification [12]. Analogue electronics enabled a rapid expansion in closed loop control, particularly in the process industries [13]. These advances allowed a great reduction or even complete elimination of moving parts in machine controllers. However since the logic for the actions to be performed was still defined in hardware, (albeit in electronic rather than mechanical form), machines remained relatively inflexible in operation.

3.2.2 The Emergence of Computer Control

A major change in control systems technology occurred with the development of the so-called "stored program" computer which enabled the control logic to exist "independently" of the control hardware. One of the first electronic computers was ENIAC (Electronic Numerical Integrator And Calculator), produced in 1946 at the University of Pennsylvania. It was composed of more than 19,000 valves and was programmed using wired plug boards [14]. At this time John Von Neumann suggested the idea of storing the program instructions in the computer as coded digits, in the same way that the data itself was stored [15]. The earliest stored-program computers were research machines developed in the UK and the USA during 1950 and 1951 [16].

The size, cost and environmental needs of these early computers limited their general applicability and the market for them was very definitely data process-

ing at company corporate level. The newness of the technology was reflected in the poor reliability of these physically large and expensive machines. Serious attempts at using digital computers for real-time control did not emerge until the late 1950s [17]. These computers typically had an addition time of one millisecond and a mean time between failures (MTBF) of about 50 to 100 hours. They were used for centralised supervision in process control applications where high installation and maintenance costs could be justified.

By the 1960s digital computers typically had an addition time of about 100 microseconds and a MTBF of around 1000 hours. Increased performance and reliability provided the opportunity to use computers not only in a supervisory mode but also to directly control processes. There were however still major problems with both controller cost and reliability in these applications [18].

During the decade from 1965 to 1975 smaller, faster, more reliable and lower cost computers termed minicomputers emerged and were widely used for complex control applications [19]. These minicomputers could typically perform additions in one microsecond and their MTBF was better than 20,000 hours [20].

3.2.3 Microcomputer Systems

It was with the evolution of the microcomputer in the 1970s that the digital computer became an attractive alternative for nearly all control tasks [21]. These devices were just as powerful and reliable as existing minicomputers at from one-tenth to one-hundredth the cost [22]. During the 1980s, microcomputers have become even more attractive for both simple and demanding automation tasks. With developments in microelectronics technology such as Very Large Scale Integration (VLSI) the current trend toward more powerful and more cost effective microcomputer hardware can be expected to continue [23]. See section 6.6.

3.3 COMPUTER BASED MACHINE CONTROL

3.3.1 Introduction

Computer technology has been made usable in different industries each with their own distinct environment, background of skills and levels of user training. Current controller types while sharing common implementation technology have distinctly different historical origins. These different starting points have signifi-

cantly affected how each type of controller is perceived [24]. The following sections take a more detailed look at the evolution of these major categories of computer based controller.

3.3.2 Programmable Logic Control

A programmable logic controller (PLC) is defined by the US National Electrical Manufacturers Association (NEMA) [25] as:

"A digitally operating electronic apparatus which uses a programmable memory for the internal storage of instructions for implementing specific functions such as logic, sequencing, timing, counting and arithmetic to control, through digital or analogue input/output modules, various types of machines or processes."

In essence the programmable controller consists of computer hardware which is programmed to simulate the operation of the individual logic and sequence elements that might be contained in a bank of relays, timers, counters and other hard-wired components [26].

In the late 1960s a small number of DEC PDP-8, IBM 1800 and Honeywell 316 computers were being used for control purposes [10]. In 1968 a specification was written for General Motors Hydromatic Division for a "programmable logic controller". Hydromatic had experience with computer control and were aware of its advantages. However the requirements for sophisticated software preparation and the problems associated with using mainframe computer hardware, which was not suited to industrial environments, made it clear to Hydromatic that a new generation of controls was required. Hydromatic's revolutionary concept was that of a solid-state control panel that could have its control functions changed without any wiring alterations [27].

The major reason that programmable controllers were acceptable when minicomputers were not, was in the area of software [10]. Minicomputer control required applications programs written in either assembly language or in real-time Fortran. Few end users of control equipment had programming staff sufficiently trained to handle the problem of debugging or reprogramming minicomputers. The software technique that really made the programmable logic controller succeed was the introduction of a relay ladder format that provided the machine builder with a means of program representation he could relate to. It also allowed end users, typi-

cally plant electricians, to change, update and edit the program once the machine was installed in his plant [28].

Early in its history the role of the PLC was clear; it was used primarily as a straight relay replacement [29]. The benefits were associated with economies in the wiring of the machines and probably most significantly in the area of machine debug. A completely new relay ladder logic program could be entered in a matter of hours whereas to rewire a relay panel would require days or even weeks [30]. Programmable logic controllers were tailored to meet requirements such as repetitive I/O polling, rapid logic solving and repetitive real-time predictability. None of these features were available on a general purpose computer in an industrially usable form. PLCs are thus highly specialised computers which are widely used and understood on the shopfloor but their functionality has traditionally been limited [31].

The capabilities of PLC have been progressively extended in recent years to include closed loop process control, motion control, maths functions, operator graphics, high level language programming and networking capabilities [32]. In some of today's more advanced control systems the original concept and application of the PLC is almost forgotten. The PLC aided by its industrial respectability has evolved to become the cornerstone of many industrial automation systems competing directly with industrial microcomputers [28]. See section 3.3.7.

3.3.3 Process Control

Computer process control is defined by Groover [33] as:

"The use of a stored program digital computer to control an industrial process usually of a continuous nature."

Closed loop pneumatic and electronic analogue controllers were first supplemented in the mid 1960s [34] with direct digital control (DDC) systems, implemented using large centralised computers [35]. Conventional analogue electronic controllers usually remained, mounted close to the sensors and actuators and were used for back up in the case of computer failure which frequently occurred. The fast DDC loops were written in assembler (process frequencies in seconds) while the slower supervisory loops (process frequencies in minutes) were typically written in Fortran [36]. The vendors of this early equipment included, Foxboro, GE and IBM [34].

In addition to closed loop control, process plants typically require a large amount of binary I/O in order to interlock various devices and prevent damage particularly during start up and shut down sequences. By the early 1970s PLCs were typically used independently of the process control system for these I/O functions and as direct replacements for relay panels [35].

The cost of DDC systems fell rapidly and by the mid 1970s they became within the financial reach of smaller process plants [37]. Many of these systems were however poorly implemented and did much to tarnish the image of DDC process control [35].

From 1975 onwards the microcomputer enabled massive reductions in the cost of control systems. It became cost effective to directly replace local analogue electronics and practical to integrate these devices in a reliable manner. Modular distributed controls systems (DCSs) offered local intelligence and a level of process redundance in the case of failure [38]. The efficiency of process control was progressively improved through the use of feed forward and model based control techniques [39].

In state of the art systems local controllers now typically handle all processing tasks with the exception of global optimisation, statistical process/quality control and information storage [40]. The multiple microprocessor architectures now employed offer redundancy with automatic switch-over to back-up systems in the event of failure [41].

The latest process control philosophy is that of hybrid continuous/batch/sequence controls [42]. Many vendors have with mixed success, attempted to include some form of logic programming capability in their continuous control systems in order to replace the use of PLCs and relay ladder logic which has been found cumbersome by many users [35].

3.3.4 Numerical Control

Numerical control (NC) is a form of programmable automation in which the processing equipment is controlled by means of a language composed of numbers, letters and other symbols [43]. The development of numerical control owes much to the U.S. Air Force and aerospace industry during the 1950s. Although early NC controllers used valves or later semiconductors in their hardwired logic circuits and

not computer control, they defined numerical control to mean machining based on coded numerical information [44].

The development of software to automatically generate this numerical information and thus enable high level applications programming was vital to the success of NC [45]. Work on NC programming systems at MIT resulted in the creation of APT which was first used industrially in the early 1960s on large mainframe computers [46]. Efforts were made at an early stage to create a machine independent programming system with postprocessing to suit machine specific NC codes as necessary [47]. There has since been a high degree of standardisation on at least the concepts of APT which has gained world-wide acceptance [48]. NC machine tools are thus typically supported by a centralised part programming system. Advanced direct numerical control (DNC) installations of the late 1960s and early 1970s integrated hardwired NC machine controllers with their part programming systems using large centralised computers [45].

With advances in computer technology (see section 3.2.2), hard-wired logic circuits were progressively replaced, and the computer numerically controlled (CNC) machine tool was created [49]. CNC systems have now been developed specifically for the machine tool industry. They typically accommodate only limited configurability and are supplied to customers with their associated type of machine tool [50].

The internal design of CNC controllers has seen progressive rationalisation with the introduction of multi-processor architectures and the adoption of electronics industry standard bus systems [51]. CNC controllers typically incorporate local software and graphic display facilities that enable the operator to review and modify NC programs. Advanced controllers usually of Japanese origin now incorporate features such as adaptive control and three-dimensional contouring capabilities [52]. State of the art distributed numerical control (DNC) systems typically involve the use of a central computer for part program storage and real-time transmission to various attached CNC controls [53].

The term NC is now universally applied to the flexible automation of general purpose machine tools. The applications of numerical control have expanded from purely metal cutting to include applications such as PCB drilling, assembly, drafting and inspection [54]. The common operating principle of NC in all these applications is control of the relative position of a tool or processing element with respect to the object (e.g. work piece) being processed [33].

3.3.5 Robot Control

The evolution of the robot is a very emotive subject which has been well documented [55, 56]. There is however no international agreement as to what constitutes a robot [57, 58]. The ISO (International Standards Organisation) definition of an industrial robot begins:

"An automatic servo-controlled reprogrammable multifunction manipulator having multiple axes, capable of handling materials, parts, tools or specialised devices through variable programmed operations for the performance of a variety of tasks."

In Japan a pick and place device (a simple arm whose motions are defined by mechanical end-stops and typically sequenced from a PLC) is termed a robot, whereas in the West it is considered to be a special case of fixed automation [59]. To complicate the issue further there are now numerous classes of robotic device including various modular machines and programmable manipulator systems [60, 61]. See also section 7.3.5. In an effort to overcome this confusion the Japanese have coined the global term Mechatronics to refer to the combination of mechanics and electronics that is present in all these devices [62].

The mechanical requirements for robots have never been particularly demanding [63] and the restraining technical factors in robot development have been primarily in real-time control and the successful integration of associated sensors and actuators [64]. Viewed in terms of its general configuration a robot is essentially a sophisticated machine tool and thus its control system possesses similar basic features and has been able to utilise machine tool technology for much of its development [65]. The robot however is applied as a general purpose manipulator and therefore requires integration into its intended application [66] whereas the CNC machine tool is generally able to operate as an essentially self contained special purpose machine of well defined functionality [67].

Many of the early developments in industrial robots and their associated control systems originated from the work of George Devol between 1954 and 1958. This work was exploited commercially in the US by Unimation, a company formed in 1962 by J Engelberger [68]. The controllers of these early industrial robots were hardwired to perform a specific set of tasks.

One of the first flexible control systems originated at MIT in 1968 and was based on a DEC PDP-6 computer [69]. Robotic control architectures have since

taken a number of approaches to meeting the inter-related requirements of configurability, computational power and programming. Early systems [70, 71] typically relied on a large minicomputer to provide both programming and control functions, but the systems themselves were typically implemented for a particular robot with little attention to modularity. Provision for non-manipulator interfaces and for manufacturing integration has usually received scant attention [72]. Later systems [73, 74] have tended to employ a larger number of processors utilising much of the control technology which had already been developed for the machine tool industry [64]. Recently there has been interest in the use of specialised computational hardware for manipulator control [75].

Robot programming methods have evolved which are distinctly different from those used for NC part programming. Facilities for motion teaching are almost always provided and more advanced systems incorporate off line textual programming facilities [76].

In the past there has been little standardisation of robot programming systems [77] although a number of groups are now starting to address this issue [78, 79]. To improve application visualisation and productivity, state of the art systems are starting to utilise graphical interface, modelling and simulation tools [80, 81]. See also sections 4.5 and 4.5.

3.3.6 Motion Controllers

Programmable motion controllers are a new breed of position control device, usually with limited I/O and programming capability. They answer the need for custom motion control in user defined machine configurations [82].

Motion controllers have their technological origins in CNC and robotic control systems. The need for motion control in diverse manufacturing applications has led to the development of general purpose controllers for both open and closed loop motion control. The capabilities of these devices have evolved from simple point to point operation of single axes to multi-axis coordinated operation [83] and they are now available in numerous forms [84]. The general trend has been to offer more flexibility in the manner in which motions can be described and programmed [85, 86].

The application of motion controllers is extremely diverse. Overall system performance is typically limited by the capabilities of the drive systems used [87].

The notable development of brushless servo motor drives has progressively advanced the application boundaries for programmable motion control using closed loop electric drives [88].

Motion controllers are typically used as devices within larger machine control systems. The integration of such systems is typically achieved using industrial microcomputers [89] or more crudely with PLCs [90].

3.3.7 General Purpose Industrial Computers

Having traced the emergence of computer control in section 3.2, this section reviews the current state of general purpose industrial computer systems.

Most standard business or personal computers are neither sufficiently robust nor offer the required electromagnetic screening for the shopfloor [91]. The manufacturer's solution to these problems has been the development of more ruggedly packaged general purpose computers. These fall broadly into two hardware categories, industrialised business or personal computers, and purpose built industrial computers [92]. The former are today predominantly IBM compatible MSDOS based microcomputers [93].

Business and personal computers have a wealth of excellent software packages available for them and have become widely accepted [94]. In an industrial context they are well suited to program preparation, data logging, user display and supervisory control. Software packages are now available for them to fulfil many types of control applications at least to some degree and at modest cost [93, 95]. A wide variety of expansion hardware is also available to extend their capabilities in an industrialised and modular form [96]. Restrictions on the use of purely MSDOS based computers relate chiefly to severe limitations in the operating system for real-time control applications [97, 98].

Purpose built industrial microcomputers are typically based around standardised parallel bus hardware together with a suitable real-time operating system. This technology is reviewed in chapter six. Industrial microcomputers are now being extensively used for the flexible control and integration of both dumb and intelligent devices [99]. Microcomputer system applications are extremely diverse and range from low level sensor and actuator interfacing to cell control activities [100]. Section 3.4.4 considers the integration of such control devices.

3.4 ASSESSMENT OF CURRENT CONTROL METHODS

3.4.1 Introduction

Regardless of the type of industry, common changes can be observed in the implementation of control systems as enabling technology has improved.

Mechanical control systems were typically an integral part of the machine they regulated. With the development of electrical and electronic circuits the controller became a physically separate item but remained close to the sensing and actuating elements associated with it.

The advent of the first reliable computers saw a progressive move towards more centralised control of machines and processes in high volume or high cost production sectors of industry which could justify very high levels of investment in automation. Computer technology was initially very expensive, non ruggedised and difficult to program. Centralised controllers were the most cost effective solutions with a minimum of dedicated control electronics at individual machines or devices.

As the cost of the technology progressively dropped and its industrial packaging and usability improved it became practical to place more intelligence close to the process sensors and actuators on individual machines. Such decentralised control, based on low cost microprocessor technology has now penetrated virtually every industrial sector [101].

3.4.2 Control System Usage

The industrial usage of microelectronics-based real-time controllers is shown in figure 3.1, from a recent Policy Studies Institute (PSI) report funded by the DTI [102]. Between 1981 and 1987 the differences in usage between industries have progressively narrowed as the technology has spread to more plants in the traditionally low-tech sectors.

Table 3.1 lists the range of applications identified as being controlled by microelectronics in the 1985 PSI survey [101]. PSI reported that between 1981 and 1983 the number of different processes controlled by microprocessors had doubled and was half as great again by 1985. In 1989 they concluded that the technology is being used in most of the main production processes of importance in industry today.

mashing	refrigeration	boring
mixing	pasteurisation	milling
blending	cooking	grinding
balancing	firing	turning
weighing	melting	roughing
counting	boiling	routing
collating	moulding	punching
calibrating	drawing	stamping
size sorting	rolling	pressing
colour sorting	unrolling	photopressing
colour matching	forming	drafting
grading	forging	printing
sizing	die casting	plate making
sequencing	extrusion	paper finishing
time control	compressing	compositing
pattern control	shaping	painting
flow control	straightening	spraying
speed control	bending	inking
braking control	defiling	enamelling
tension control	folding	dyeing
boiler control	flanging	dipping
electricity control	coiling	colour control
spark emission	winding	coating
control	twisting	plating
moisture control	spinning	insertion
pressure control	stranding	assembly
cleaning	scouring	testing
washing	carding	quality control
irrigation	seaming	inspection
drying	weaving	fault detection
developing	tentering	batching
fermentation	knitting	handling
filtration	stitching	pumping
refining	embroidery	lifting
electrolysis	baling	loading
leak detection	binding	unloading
impurities	glueing	filling
detection	bonding	bottling
mineral content	welding	capping
control	riveting	canning
corrosion	soldering	packing
treatment	sawing	wrapping
effluent disposal	guillotining	bagging
air conditioning	metal cutting	strapping
deionising	fabric cutting	pricing
temperature	polystyrene cutting	labelling
control	drilling	coding

Table 3.1. Type of control application.

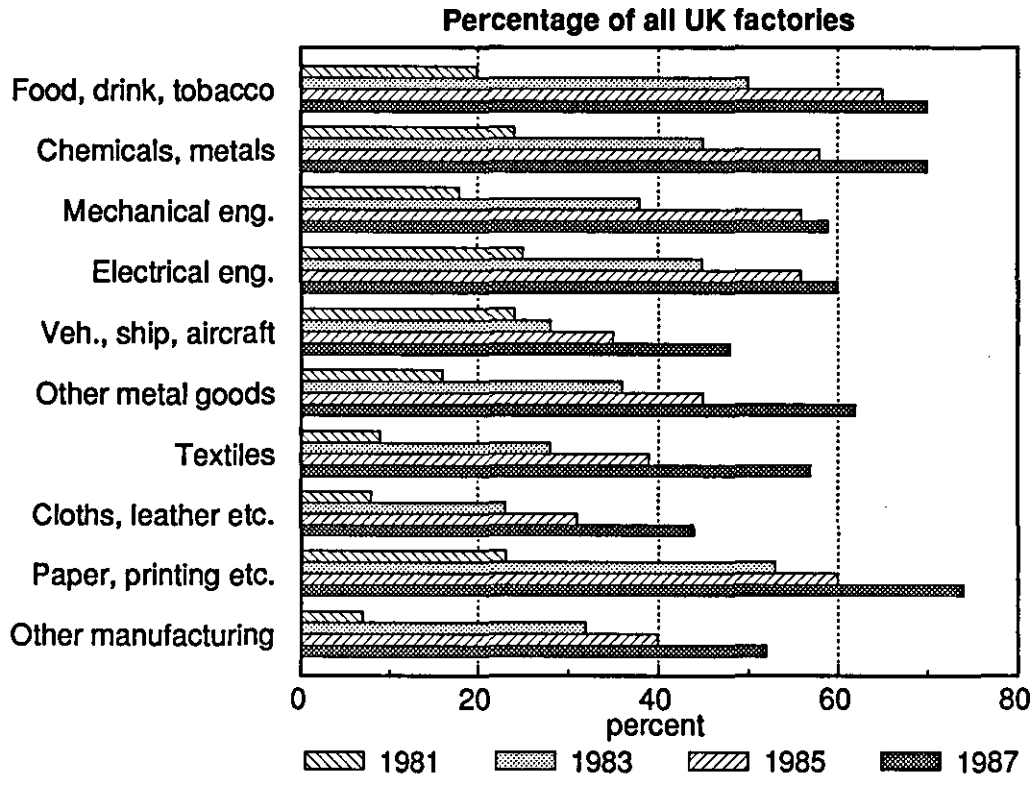


Figure 3.1. The use of microelectronics in manufacturing processes by industry.

The most common forms of microelectronics in production processes are used for the control of individual machines (particularly in the metal working industries) or individual pieces of process plant (mainly in continuous process industries such as food, drink, chemicals and metals). Although the kinds of plant are varied, the basic control principles involved are often very similar [101]. As reported by PSI the percentage of plants using microelectronics for the control of individual machines has risen steadily from 50 per cent in 1983 to 71 per cent in 1987. Microelectronics have also been rapidly exploited for testing and quality control.

For many industries centralised computer control was neither appropriate nor economic and has never been applied. In typically shopfloor machine control installations separate relay panels were used and are now being progressively replaced by individual PLCs. With the development of low cost PLCs particularly from Japan it is now more cost effective to use a PLC rather than a set of relays down to less than ten I/O lines [103].

At the other extreme in terms of processing power, CNC and robot control systems again use localised microcomputer systems typically with multi-processor

architectures [104]. PLCs and process controllers are adopting similar architectures in demanding applications [105].

Figure 3.2 compares the usage of several major categories of machine controllers [102]. PLCs are the most widely used kind of control equipment. Nearly 30 per cent of all factories use them and the total number used has doubled in the last four years. They are employed mostly for relatively straight forward control or monitoring of a single process or piece of equipment [101]. The Frost and Sullivan report on PLCs [106] predicted a rise in the UK market value from $530.2 million in 1986 to $1,416.3 million by 1991 and this despite large anticipated reductions in the unit costs of these devices with continued improvements in hardware technology. The report points out that the current applications represent only half the potential uses for PLCs even in todays market place. There have also been large increases in the use of machine and process controllers used in more complex applications than can be handled by a PLC [101]. Frost and Sullivan also stated that for many applications the choice between process controllers, industrial computers and PLCs is no longer obvious.

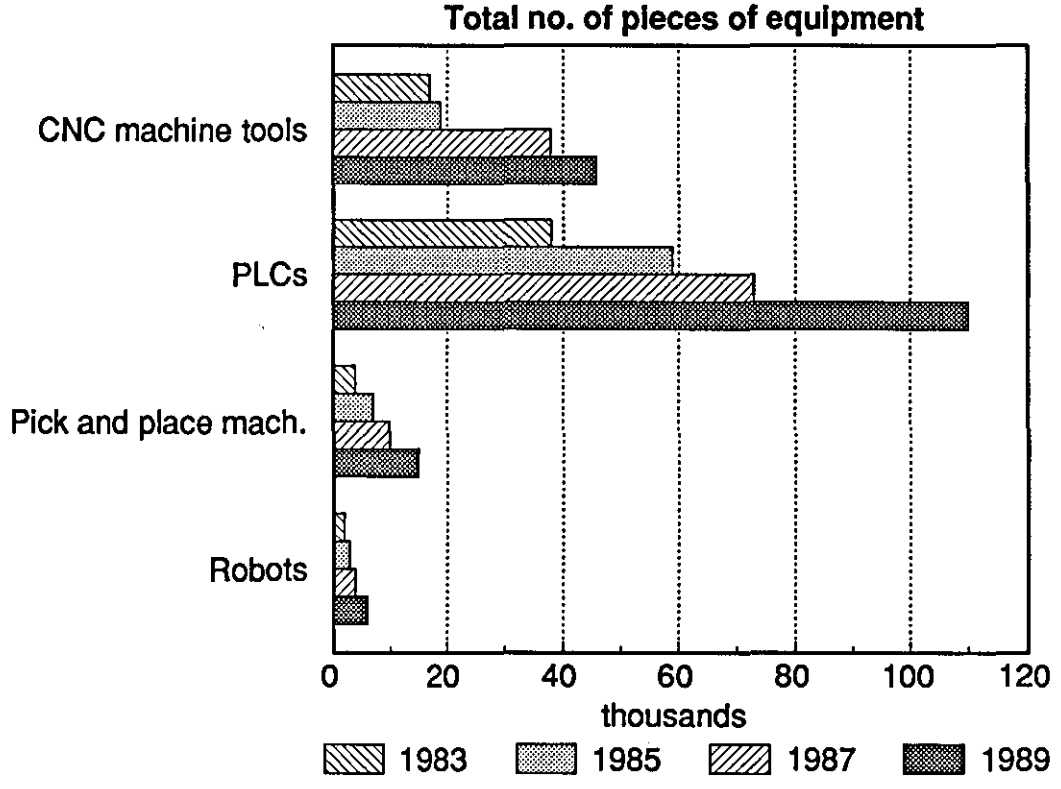


Figure 3.2. Usage of CNC, PLC, robots and pick and place machines.

In Britain about one factory in five currently uses CNC machine tools and associated controls. Far fewer factories, only between 2 and 3 per cent of them, use robots, so while the robot embodies what are perhaps the most complex control systems, their practical utilisation remains very small [102].

The extent of use of microelectronics in production processes is difficult to accurately determine [107]. The PSI survey calculated (by consideration of the total employment in UK manufacturing) that the usage of microelectronics for process applications rose from 43 per cent in 1981 to 82 per cent in 1987. On average about a third of these production processes were controlled by some form of computer controller. See table 3.2. The report concluded that some 25 per cent of processes were controlled using microelectronics in factories employing 20 or more people. While difficult to quantify the survey states that the microcomputer is still considerably under-exploited by existing users and that most companies do not recognise the scope that currently exists for its further application [107].

	1981	1983	1985	1987
Process applications				
% of users (a)	43	65	76	82
% of their processes (b)	22	27	32	31
% of total processes (a*b)	10	18	24	25

Table 3.2. Overall extent of use.

The integration of groups of machines or several stages of processes offers greater potential advantages when used in appropriate situations, but requires much greater specialist expertise and organisational change to introduce successfully and is therefore at present much less common [101].

3.4.3 User Perceptions

Most users derive very important benefits from the application of microelectronics based controls. Of those surveyed by PSI in 1989 [108], three-quarters reported the ability to manufacture more consistent, better quality products through improved control of their production processes. More than half the user of computer based machine controllers recognised benefits from greater speed of output, lower production costs and more efficient use of labour, equipment and materials.

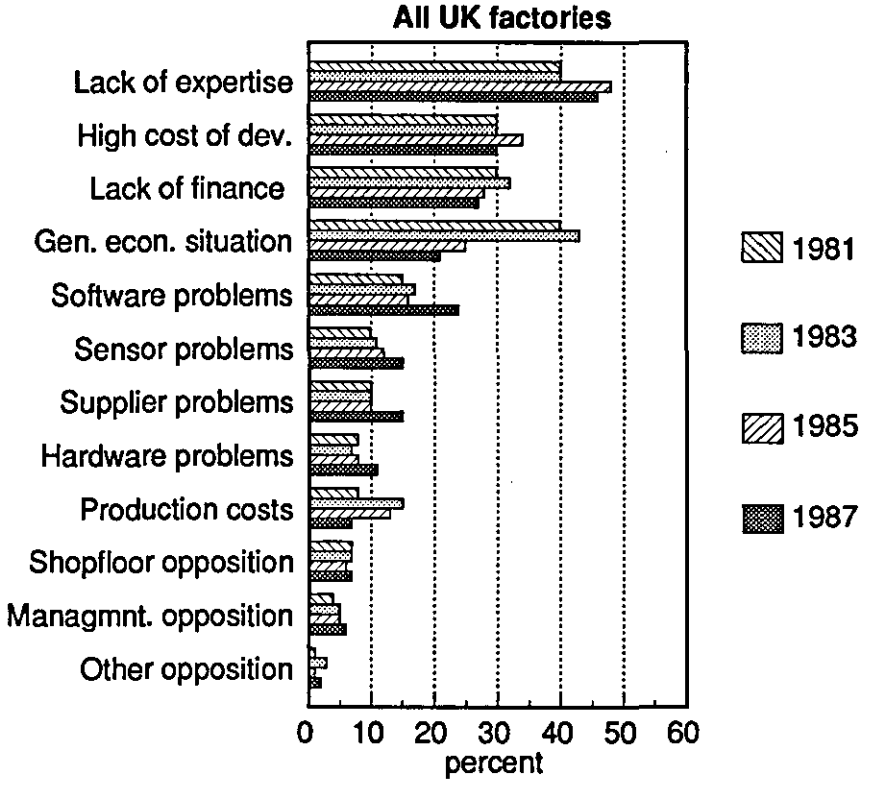


Figure 3.3. Main difficulties of microelectronics users.

The most common difficulty, experienced by nearly half the users surveyed by PSI over the last seven years, has been a lack of people with the specialist microelectronics expertise required. Figure 3.3 lists the main difficulties of microelectronics users as found by PSI [108]. One in five users reported problems with existing control software. They were reported particularly from plants with high levels of control system usage and with more advanced kinds of application. PSI concluded that software problems were assuming growing importance and were not merely the "teething troubles" experienced by new users. Hardware problems were found much less predominant and were usually related to equipment price or supply rather than performance.

3.4.4 Integration Needs

In both continuous process and discrete parts manufacture there is now a perceived need for plant wide integration to increase manufacturing responsiveness. See chapter two. Since microprocessor based control systems are now commonly distributed and placed with their associated I/O close to the controlled plant they provide the potential for tremendous real-time information visibility if properly integrated [109]. However in the drive towards the implementation of CIM, the role of the machine control system is often overlooked, ignored, or simply too difficult for companies to implement effectively [102].

As illustrated in figure 3.4, about half the factories in Britain are using microelectronics in stand-alone applications - to control one or more individual devices, machines or processes. Far fewer, only about 19 per cent, have implemented any form of integrated control of a group of devices, machines or processes.

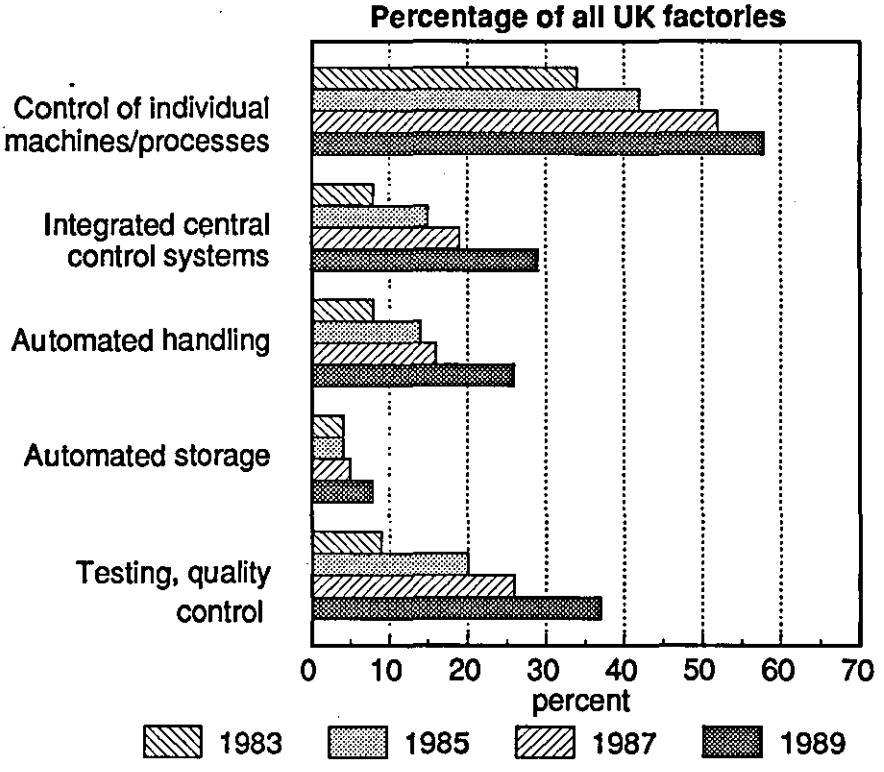


Figure 3.4. Application of microprocessor based controllers.

The integration of a diverse range of application dependent devices is an important practical requirement for machine control. In a typical mechatronics environment multiple devices, many with their own intelligent controllers, must be made

to function as a single integrated whole [110]. The physical and logical structures for integration have been the subject of extensive research and are typically hierarchical [111]. The concept behind such integration is however relatively simple; at machine level a set of hardware and software functional entities will be required to create an individual control system. When viewed externally this system can in turn be considered as a single composite element at the next level of integration and so on [112]. See figure 3.5.

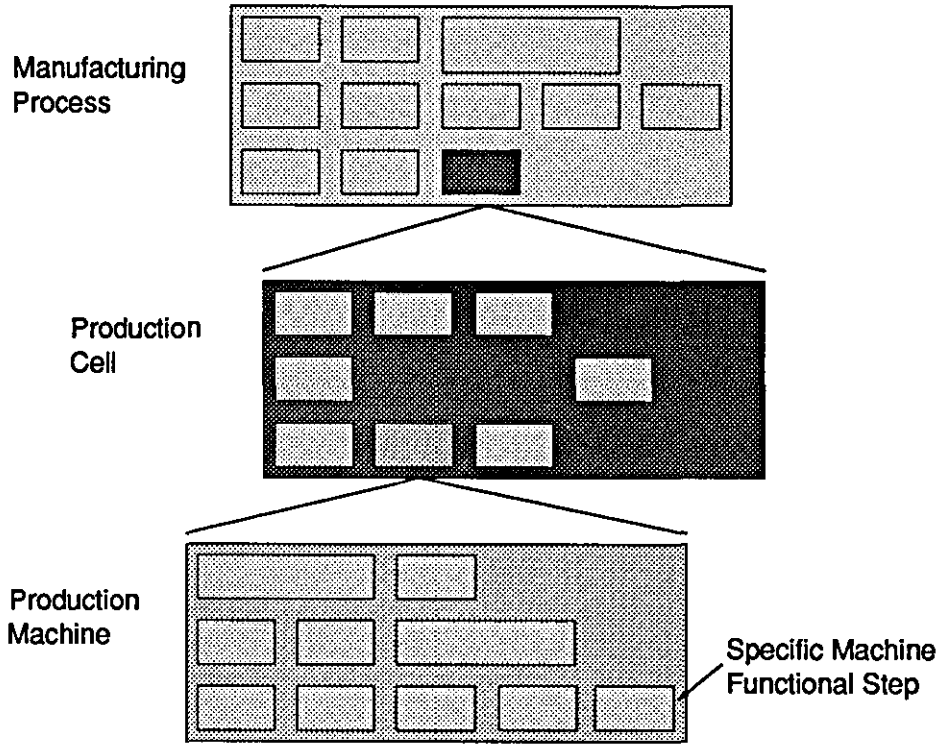


Figure 3.5. Integration hierarchy. Note this figure is identical to figure 2.5 and is reproduced for convenience.

Despite the potential offered by the computer the practical integration of control devices still presents major problems [113]. Manufacturers typically base the operation of their controllers on proprietary software and hardware that offers the required functionality when the devices (e.g. robots, PLCs, conveyors, packaging machines, vision systems) are used stand-alone but is often inadequate when the devices need to be integrated together to form a practical system [66, 65].

The typical current solution to such problems is to directly interconnect digital I/O between the various device controllers [114]. In this type of set-up a PLC is often used to provide the primary sequencing of more complex custom devices

(such as motion controllers, robots or machine tools) using its configurable I/O system. Such inelegant and ad hoc solutions may work, but are difficult to design, commission and maintain. Of even greater importance they are almost impossible to reconfigure to meet varying production requirements.

In order to improve on such solutions many system builders advocate integration using PCs, dedicated machine controllers and PLCs in a triangular hierarchy [115, 116, 117]. See figure 3.6. The PLCs and machine controllers are utilised for time critical control while the PCs perform monitoring, data collection and data management functions. Such systems are typical of practical but unstandardised approaches to integration which will become increasingly difficult to maintain and reconfigure in the future [118].

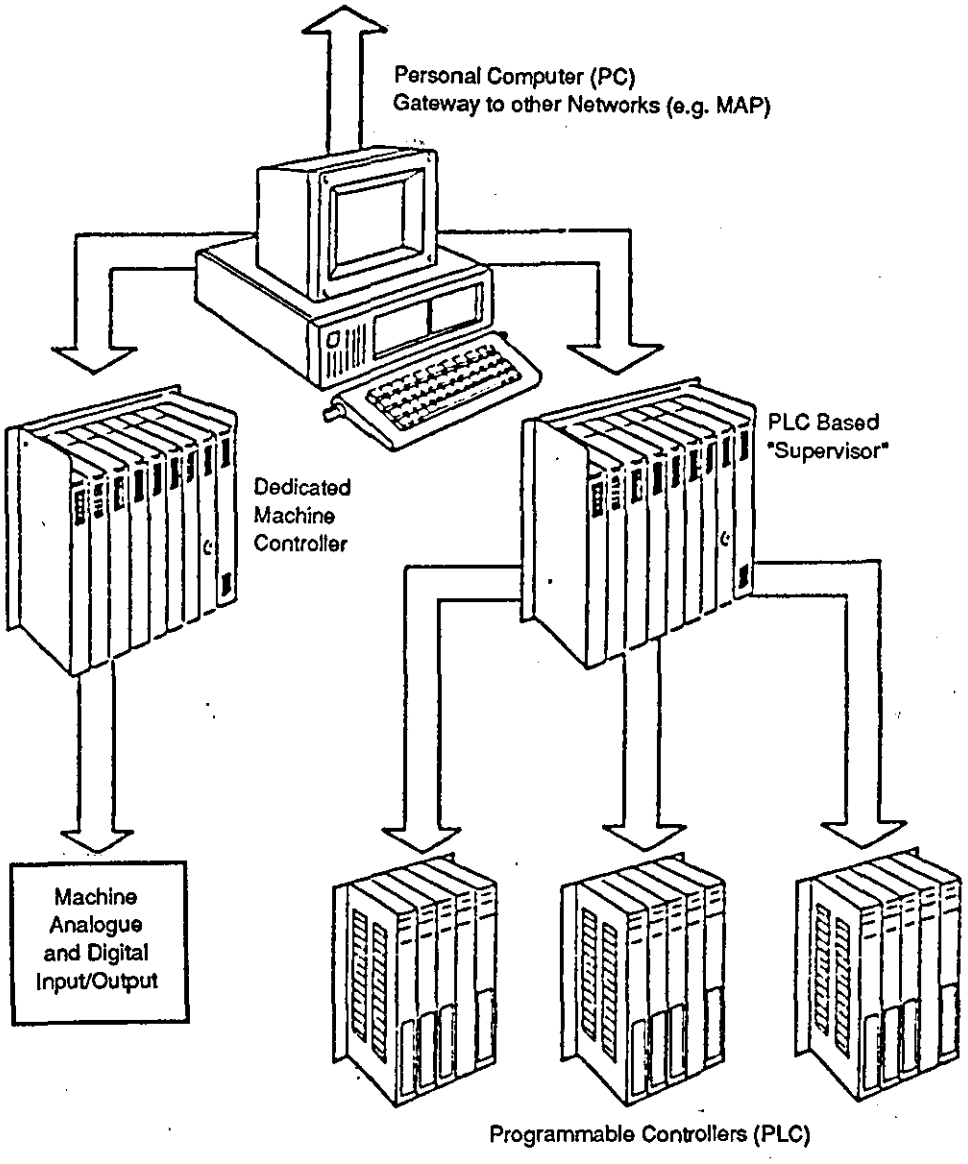


Figure 3.6. PCs, dedicated machine controllers and PLCs integrated in a triangular hierarchy. Source: Pinto.

To address the issue of integration in a more ordered way, numerous vendors, users, and university research groups have talked about and worked on various cell control concepts [112, 119]. See also sections 4.2 and 4.3. Practical cell control systems typically involve the use of an industrial microcomputer of appropriate power which:

- Interfaces to a higher level network if present.
- Provides a cell-level man-machine interface if required.
- Provides reprogrammable integration of the cell's individual device controllers to provide maximum operational flexibility.

While relatively simple in concept, see figure 3.7, the universal cell controller has proved to be rather elusive due to a number of complications. Interfacing to all the device controllers in an adequate manner usually proves very difficult since many projects involve an element of retrofit to older controllers or unsupported devices [114]. Thus where integration has been attempted the system configuration is often effectively fixed since the costs and problems associated with expansion or modification are generally immense [112].

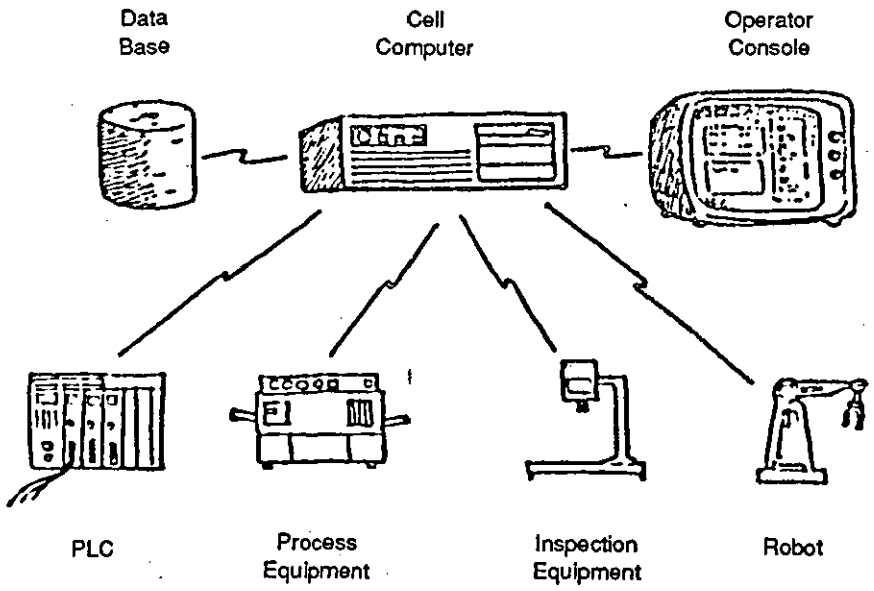


Figure 3.7. Workcell schematic. Source: Bukow.

There are current examples of well integrated systems particularly in CNC and large scale process control applications where requirements are relatively fixed and well defined [120]. Even in these systems, equipment from different manufacturers is usually not compatible and there is virtually no support for non-standard devices making system reconfiguration difficult.

The safest option for most companies currently is to leave the integration of machine control systems to specialist system builders who have experience in solving interfacing problems [34]. For some years to come at least, such system builders will be almost indispensable for overcoming the problems of merely linking incompatible devices together [121] with little consideration being given to the wider issues involved in achieving effective system integration [118].

Manufacturers have failed to realise that in attempting to tie together and control computerised processes, they need to establish a control hierarchy with a supporting infrastructure even at the lowest levels i.e. associated with real-time control. As a result current manufacturing facilities are typically composed of a multitude of disjointed systems and applications [114]. Due to these shortcomings vendors of traditional controllers are now realising the importance of supporting integration. "Significantly larger PLCs will be used in the future. These devices will tend to aggregate groups of integrated systems into a single group and they will be significantly larger in communication and integration capabilities. Extra functionality must be built into the controller to enable such capabilities to be supported. The functionality in terms of machine control may be similar but the manner of implementation will be fundamentally different." D. Morley, Modicom AEG [122].

A consistent approach to integration is vital if CIM is ever to become widely utilised. The information visibility currently provided within machine controllers is generally very poor [123] and this makes the effective integration of these real-time systems particularly difficult.

3.4.5 Controller Configurability

Control requirements are obviously applications related and ideally the functionality of any controller should be individually matched to the needs of each specific process or machine. Using a suitable set of hardware and software entities (from the wide range which are currently available) a machine control system can now be readily created to perform a single, specific manufacturing application [95].

However most current controllers are of a fixed configuration (set-up to fulfil within predefined boundaries a dedicated set of functions) which cannot be easily modified or extended. Machines and control systems are therefore often badly matched or applied to inappropriate applications. US robot manufacturers for example, have found it nearly impossible to reach a break-even level of profitability [65]. Part of their problem has been producing robots and in particular associated control systems of fixed configuration that have been far more sophisticated than necessary for the relatively simple jobs that most users have wanted them to perform [124].

In some cases families of existing control devices can be flexibly configured and this feature is now being perceived as an important requirement by both system vendors and users [125]. PLCs are the best current example of a configurable controller [122]. They are however now constrained by a structure and programming philosophy which, while well understood by their traditional customers, is totally inappropriate to many areas of application.

Alternative approaches based typically around industrial board-based microcomputer systems are viewed with suspicion and even fear by many customers who do not have the skill required to support such devices in house. The use of real-time industrial computers, operating systems and system programming languages can now enable the development of flexible and efficient control systems. These approaches require a high degree of programming skill to implement but provide a very powerful, flexible solution to control problems and are commonly used by custom machine builders [86]. Such systems can initially be tailored exactly to the intended application but once installed represent a custom solution to all but the most technically skilled industrial users.

The PLC is the principal reconfigurable controller used in industry today. The PLC has evolved to satisfy the customers needs for a usable shopfloor controller and any replacement for it must offer similar usability. The industrial acceptance, simplicity, versatility and other time proven virtues of the PLC need to be retained. However there is no longer a good reason to design control systems in isolation. The machine controller of the future must be an integral part of CIM. To achieve this the functionality of controllers must be increased [126] but even more importantly this functionality must be visible to the programmer [127].

3.4.6 Control Functionality

When considering a specific machine application, an appropriate level of functional performance and operational flexibility is required from the control system. The actual requirement for both the machine and its associated controls are obviously dependent on the range and type of products to be made and anticipated changes in operating conditions. The available budget, method of implementation, available controller performance and other physical constraints typically limit the level of operational flexibility which can be achieved [128].

For the majority of current control applications adequate real-time control functionality is available from existing control systems and in many cases there are many alternative methods of implementation [107].

The limitations placed on most state of the art control systems come chiefly from:

- physical restrictions imposed by available sensor or actuator technology (e.g., drive technology limits the boundaries of programmable motion control performance) [129] or
- difficulties associated with making advanced control technology usable [127].

The capabilities of control systems have expanded progressively with improvements in enabling technologies. As discussed in section 3.2, the successive application of mechanical, electrical, electronic and latterly computer systems have all had a profound effect on manufacturing control. The advent of the digital computer has enabled the creation of controllers that are many orders of magnitude more powerful than their earlier counterparts [95, 130, 126].

Maturing technology has enabled current controllers (PLCs, DSCs, CNCs, robot controllers etc.) to solve most routine control problems in their own traditional fields and expansion into other market sectors has often been achieved by adding additional control functionality. The boundaries between controller types have thus been blurred by advanced units which now have some characteristics of microcomputers, PLCs and process controllers [2]. These advances have occurred in an ad hoc manner with little standardisation. Common examples are PLCs offering process control capability [128] or conversely process controller offering sequential logic capabilities [35]. The relationship between DSCs and PLCs is illustrated in figure

3.8. Such developments illustrate the inherent potential of microcomputers to control almost any task given suitable peripherals and software but they are currently implemented in a largely unstructured and unstandardised manner.

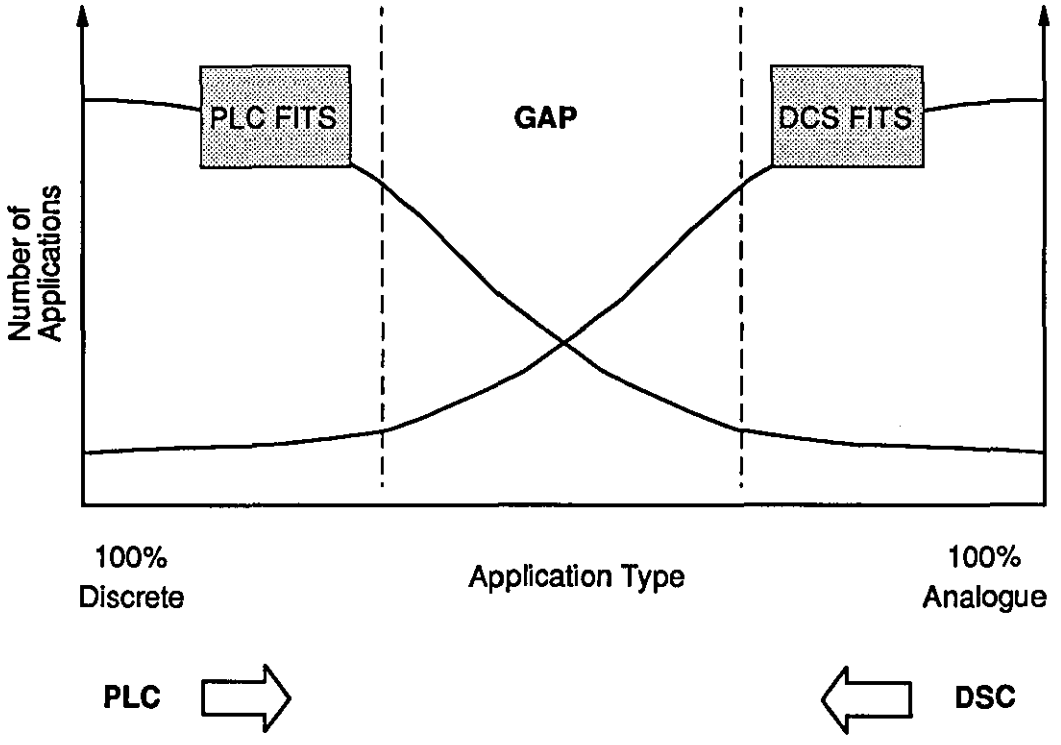


Figure 3.8. The relationship between DSCs and PLCs.

3.4.7 Compatibility and Standards

The adoption of general standards may not at first sight seem important particularly when considering the control of an isolated machine. Standards however become crucial for machines to communicate with one another and to be maintained, expanded and integrated in an effective manner.

Equipment compatibility and maintainability are major fears to users considering the purchase of new control equipment. According to a recent survey in the US magazine Automation [122] 82 per cent of users considered compatibility with existing equipment to be the most important need; more important than price, availability and even ability to upgrade. Control software is of almost endless variety and while many systems now offer very similar real-time functionality (as discussed in section 3.4.6) few are compatible, even for the same class of controller [2].

Due to these fears over equipment incompatibility and maintainability controller manufacturers particularly in the PLC market often become the preferred supplier of control equipment to certain companies. Their products become an in house company standard which is understood, known to be reliable and which can be accepted and supported on the shopfloor. This sort of policy has however often greatly restricted user choice and has resulted in the use of certain controllers in totally inappropriate applications. For example the use of PLCs employing relay ladder logic to control the motion of servo motors [1].

Despite the possible advantages there has been very little standardisation of machine controllers. Many users are not aware of the importance of standardised controller structure and implementation methods to facilitate system configuration and integration [131]. The International Electro-technical Commission (IEC) already has limited PLC standards [132] that specify programming and debugging tools, programming languages and PLC selection criteria, although conformance with these standards is currently very poor. There have also been some initial attempts at using standard hardware busses in PLCs [133, 2]. Manufacturers have virtually all adopted their own individual hardware and software systems each restricted to a limited range of applications.

Standards are required that can be accepted and adopted by industry in equipment of widely varying cost and performance [65]. Vendors are now realising that standards and open architectures are going to play a large role in future developments. "Open architectures are perceived as being a benefit. I think you'll see several vendors move [towards standards and open architectures] either subtly or very significantly." (Ken Jannotta, PLC Marketing Manager, GE Fanuc Automation) [122].

3.4.8 Problem Representation

The application of "technology" to real-time applications obviously predates the availability of electronic digital computers. Many open-loop and closed-loop control systems were (and still are) implemented with electrical or electronic binary and analogue devices. However the development of modern computer technology has enabled the solving of larger and more complex problems. As enabling technology has improved, techniques from the business systems computing sphere and methods developed for binary or analogue process/machine control engineering have been adapted in an ad hoc fashion to the engineering of modern digital electronic real-time control systems.

In the business sphere computer technology has been applied to simplify applications [134]. The computer technology utilised in these business applications has been implemented in a problem oriented way. This type of user friendliness is generally lacking in the configuration and use of industrial control system software [135]. In a limited manner PCs now offer cost effective and flexible solutions to such tasks as PLC programming, plant monitoring, mimic displays, data analysis and limited supervisory control [93, 94].

Powerful computer hardware is available today at low cost. Indeed in the computer industry the development of greater processing power has by far outstripped the development of software to exploit it effectively [136]. This software deficiency is seen in machine controllers where for example a new breed of "super" programmable logic controllers have been developed with very powerful processing units but a muddled software environment. In the business and real-time systems programming fields standard languages, operating systems and more recently CASE (computer aided systems engineering) tools have emerged [137]. These development tools are generally not suitable for direct embodiment in industrial real-time control systems but are now being exploited through the use of industrial microcomputers. They allow a more consistent approach to control problems, given the availability of software engineers to use them. See sections 4.4 and 4.5. The skills shortage is an underlying problem and it is only by using advanced software to simplify the representation of control problems that inevitably limited human resources can be efficiently utilised.

3.5 CONCLUSIONS

A major effect of the use of the computer for industrial control is that, with the exception of the provision of appropriate processing power and the configuration of I/O and other peripherals the hardware is essentially application independent. It is the interface devices and software which define a particular controllers functionality [2]. The flexibility of computer control is illustrated by its penetration into almost every sector of the machine controller market. The computer offers the key to an as yet largely unrealised opportunity for standardisation since the controller has now become independent of the machine it commands [135].

Current methods however are fundamentally flawed since due principally to their diverse origins they are unable to accommodate todays needs for integration, configuration and flexibility in a coherent manner [114]. These features are all potentially available by correctly utilising todays enabling technology. See chapter six.

The requirements of CIM suggest that a problem dominated approach is urgently required for the creation of a new breed of real-time machine controllers with a more natural method for control implementation.

Having studied the application of current control technology a number of characteristics can be highlighted:

- For a large number of applications the function aspects of real-time machine control are well understood at machine level. Problem representation is however often restricted due to technical limitations which need no longer exist if current enabling technology can be correctly utilised [1].

- Study of the evolving practices in business and real-time systems development reveals the following clear trends [138, 139]:

  (1) The current ad hoc development practices used for the implementation of machine controllers should be superseded.

  (2) The scope of development practices should be widened from focusing purely on implementation to encompass problem formulation and definition.

- Producing a range of machine controllers which can be totally integrated into a maintainable system requires the use of a coordinated approach [140]. There is the need for open systems architectures to achieve compatibility between devices to enable a more consistent approach to integration [141]. Much emphasis has been placed on the importance of the development of standard hardware communications interfaces [142, 143]. See section 6.6. This is however only one small but essential facet of the requirements for the emergence of open standards in control. If an open architecture approach to machine control is to be effective it must not restrict the use of new enabling technology for example improved processing hardware or software, the development of new user interface or better programming methods. Provision for compatibility with existing hardware is needed together with a coherent structure for expansion [72].

- An ability to accommodate wide variations in system size with reasonable efficiency is seen as essential. It is important not to overburden a simple system with complex hardware and software which is

largely redundant but to offer logical integration and expansion paths [140].

- The microprocessor is a very powerful industrial tool however the availability of engineers with a proper insight into its correct implementation is currently limited and this problem is likely to worsen [144]. A consistent approach to control offers the potential to simplify problem presentation and thus utilise skilled personnel in a more efficient manner.

Chapter 3: References

1 Blue, S, "Why hang on to ladder logic?", Industrial Technology, Hanover Publications, UK, February 1989, pp 34-35.

2 Hauser, N. "PLC or VME Bus: An open question?", Microsystem Design, March 1990, pp 8-9.

3 Molian, S. "The design of cam mechanisms and linkages", Constable, London, 1968.

4 Jones, F. D. (ed.) "Ingenious Mechanisms for Designers and Inventors", 3 volumes, Industrial Press, 1948-1951.

5 Anon, "History and Nomenclature of Machine Tools", Bristol Polytechnic, Dept. of Eng., Technical Note, March 1980.

6 Ferguson, E. S. "Kinematics of Mechanisms from the Time of Watt", U.S. National Mus.(Smithsonian Inst.), Bulletin 228, paper 27, 1962.

7 Fry, M. "Designing Computing Mechanisms", Part III, Cam Mechanisms, Mach. Des. 17, October 1945.

8 Gutman, A. S. "Cam Dynamics", Machine Design, March 1951, pp 149-153.

9 Rees-Jones J.(ed), "Cams and Cam Mechanisms", Mechanisms '74 Conference, 1974, Liverpool Polytechnic, I. Mech. E. Publications, 1978.

10 Treer K.R., "Assembly Automation", Chapter 11 - Controls, SME, 1979, pp 422-444.

11 Senyard, C. P., "A Brief History of Process Control Technology", Seventh Annual Control Eng. Conf., May 1988, Section XXV, pp 1-6.

12 Schmitt, N.M. and Farwell R.F., "Understanding Electronic Control of Automation Systems", Texas Instruments, Inc., Dallas, Texas, 1983.

13 Singh, M. G. et al, "Applied Industrial Control", Chapter 7, Pergamon Press, Oxford, 1980.

14 Bonner, D.T., "Traditional Information Technology: PLCs vs PCs", Proc, 2nd IFS Int. Conf. Machine Control Systems, pp 129-138, May 1987.

15 Goldstine, H.H., "The Computer: from Pascal to Von Neumann", Princeton University Press, 1972.

16 Osborne, A., 1978, "Introduction to Microprocessors", McGraw-Hill, Berkeley.

17 Astrom, K.J. and Wittenmark, B., 1984, "Computer Controlled Systems: Theory and Design", Prentice-Hall, Englewood Cliffs, N.J.

18 Savas E.S., "Computer Control of Industrial Processes", McGraw-Hill Book Company, New York, 1965.

19 Anon, "Computer Software for Process Control", BISRA Working Party, BSC, London, 1969.

20 Rembold, U., Armbruster, K. and Ulazmann, W., "Interface Technology for Computer-Controlled Manufacturing Processes, Marcel Dekker Inc., New York, 1983.

21 Best, P. J., Repledy P. G., Escuder, M. A., Powner, E. T., "Introduction to Microprocessor Systems", Parts I, II and III, Int. J. Electr. Eng. Educ., Vol. 14, No. 1 pp 73-80, No. 2 pp 173-186, No. 3 pp 269-279, Jan., April and July 1977.

22 Anon., "Special Issue on Microprocessor Technology and Applications" Proc. of IEEE, Vol. 64, No. 6, June 1976.

23 Ciminiera, L. "Advanced Microprocessor Architectures", Addison-Wesley, 1987.

24 Considine, D. (ed.), "Standard Handbook of Industrial Automation", Chapter 3, Control Systems, Chapman and Hall, 1986.

25 Anon., "NEMA Product Statistical Bulletin for Systems Group (4-IS-2)", National Electrical Manufacturers Association, Washington, D.C., February, 1985.

26 Groover, M.P., "Automation, Production Systems and Computer Integrated Manufacturing", Chapter 21, Sequence Control and Programmable Controllers, Prentice-Hall, 1987.

27 Kissel, T. E. "Understanding and Using Programmable Controllers", Chapter 2: History, pp 12-19, Prentice-Hall, 1986.

28 O'Sullivan, W.T., " Programmable Logic Controllers from Relay Logic to Process Control", Sixth Annual Control Eng. Conf., Rosemount, Il, May 19-21, 1987.

29 MacGregor, M., "The Evolution of the Controller", Industrial Computing, February 1987, pp 21-26.

30 Schmitt, N. M., "Understanding Electronic Control of Automation Systems", Texas Instruments, Inc., Dallas, Texas, 1983, Chapters 4 and 8.

31 Quatse, J. T., "Programmable Controllers of the Future", Control Engineering, Vol. 33, January 1986, pp 59-62.

32 Prevett, S., "Combining Machine Control with Process Intelligence", Drives and Controls, March 1988, pp 17-20.

33 Groover, M.P., "Automation, Production Systems and Computer Integrated Manufacturing", Chapter 22, Computer Process Control, Prentice-Hall, 1987, pp 673-705.

34 Wain, L., "Progress in Process", Industrial Computing, October 1986, pp 14-18.

35 Corley, P., "A Brief History of Process Control Technology", Seventh Annual Control Eng. Conf., May 1988, Section XXV, pp 1-6.

36 Anon, "A Brief Guide to Process Control Computing", Computer Software in Process Control, BISRA Working Party Report, BSC, London, 1969.

37 Bennett, S. and Linkens, D.A.(eds.), "Computer Control of Industrial Processes", Peter Peregrinus, 1982.

38 Zimmerman, C.K., "Evaluating Distributed Control Systems", Control Engineering, Vol. 31, November 1984, pp 109-112.

39 Prett, D.M. and Carlos, E.G., "Fundamental Process Control", Butterworths, 1988.

40 Zimmerman, C.K., "State-Of-The-Art Control Systems", Standard Handbook of Industrial Automation, Edited by D. Considine, Chapman and Hall, 1986, pp 139-151.

41 Legge N.L. "A Comparative analysis of Distributed Control Systems from an End User Perspective", Seventh Annual Control Eng. Conf., May 1988, Section III, pp 23-31.

42 Anon. "PC3000 Technical Summary", No. HA022230, Issue: 0.4, Eurotherm Systems, Worthing, Sussex, 1989.

43 Koren, Y. and Ben-Uri, J., 1978, "Numerical Control of Machine Tools", Khanna, Delhi.

44 Anon., "Modern Machine Shop 1987 NC/CIM Guidebook", Cincinnati, Garnner, Ohio, 1987.

45 Chang C. and Melkanoff, M. A. "NC Machine Programming and Software Design", Prentice-Hall, USA, 1989.

46 Rosenberg, J.,"A History of Numerical Control", 1949-1973, DoD DAHC-15-72-C-0308, 1973.

47 Drayton, D. E. et al., "Automatically Programmed Tools," Numerical Control Programming Languages, Proc. 1st Int. IFIP/IFAC PROLAMAT Conf., 1969, Amsterdam: North-Holland, 1970.

48 ANSI Standard ANSI X3.37-1980:"Programming Language APT", New York: American National Standards Institute, 1980.

49 Koren, Y. and Bollinger, J., 1978, "Design Parameters for Sampled-Data-Drives for CNC Machine Tools", IEEE Transactions on Industrial Applications, Vol. 1A, No. 3, pp. 255-264.

50 Anon., "Allen-Bradley OSAI Numerical Control Series 8601 T: A Compact Control for 2- and 3-Axis Lathes", Technical Specification 45002152S, OSAI A-B Ltd., Poole, Dorset.

51 Debourse, E. et al, "A Modular Machine Tool CNC Based on G-64 Bus", G-64 Only, Vol. 2, No. 4, Winter 1986, pp 25-30.

52 Anon. "Mazak CAM-CNC System: Mazatrol CAM M-2", Yamazaki Machinery Works, Oguchi-Cho, Japan, 1989.

53 Koren, Y., "Computer Control of Manufacturing Systems", McGraw-Hill, New York, 1985.

54 Pressman, R. and Williams, J. "Numerical Control and Computer Aided Manufacturing", Wiley, New York, 1977.

55 B. Rooks, "The Cocktail Party That Gave Birth to the Robot", Decade of Robotics, IFS Publications, Bedford, England, 1983.

56 Ayres, R. U. and Miller, S. M., "Industrial Robots on the Line", Technology Review, USA, 34-47 May/June 1982.

57 Parent, M. and Laurgeau, C. "Logic and Programming", Robot Technology Series, Vol. 5, Kogan Page Ltd., 1986, pp 10.

58 Inagaki, S., "What is the standardisation for industrial robots?", The Industrial Robot, Vol. 7, January 1980.

59 Scott, P. B., "The Robotics Revolution", Basil Blackwell, Oxford, 1986, Chapter 1, Fundamental Robotics, pp 9-26.

60 Schmitz, D. et al., "The CMU Reconfigurable Modular Manipulator System", Technical Report CMU-RI-TR-88-7, Robotics Institute, Carnegie Mellon University, Pittsburgh, USA, May 1988.

61 Weston R.H. et al, "Modular Machines and a New Approach to Machine Control", Computer Integrated Mechanical Engineering, ASME, US, 1982.

62 Anon. "Machine Design in a Mechatronics Age", Journal of Robotics and Mechatronics, Fuji Technology Press, Tokyo, Japan, Vol. 1, No. 2, August 1989.

63 Lhote, F. et al, "Robot Components and Systems", Kogan Page, 1986.

64 Warnecke H.J. and Schraft R.D., "Industrial Robots: Application Experience", Section 2.3 Control Systems, IFS Publications, 1982.

65 Shapiro, S.F., "Robotic Systems Learn Through Experience", Computer Design, pp 58-68, November 1988.

66 Stauffer, R. N., "R T Special Report: Designing the Robotic Workcell", Robotics Today, June 1987.

67 Leathan-Jones, B., "Introduction to Computer Numerical Control", Pitman, London, 1986.

68 Engelberger, R. C., "Robotics in Practice", AMACON, American Management Association 1980.

69 Scott, P. B., "The Robotics Revolution", Basil Blackwell, Oxford, 1986, Chapter 2, The Evolution of Robotics, pp 30-31.

70 Paul, R., "Modelling, Trajectory Calculation, and Servoing of a Computer Controlled Arm", Ph.D. Dissertation, Report STAN-CS-72-311, Stanford University, November 1972.

71 Ruoff, C., "PACS: An advanced multitasking robot system", The Industrial Robot, June 1980, pp 87-98.

72 Ekong, E.S., "Controls: The Bridge to Systems Integration", 16th ESD/SMI Int. Programmable Controllers Conf., Detroit, Michigan, April 1987.

73 Ozguner, F. and Kao, M. L., "A Reconfigurable Multiprocessor-Based Controller for the Control of Mechanical Manipulators", Proceedings of the 1985 IEE Int. Conf. on Robotics and Automation, St. Louis, March 1985, pp 815-821.

74 Carter, W.C., "Modular Multiprocessor Design Meets Complex Demands of Robot Control", Control Engineering, March 1983, pp 73-77.

75 Ahmed, S. and Besant, C.B., "Motion Control of Industrial Robots with Closed-Loop Trajectories., " Proceedings of the 1984 IEEE Int. Conf. on Robotics, pp 305-309, March 1985.

76 Blume, C. and Wilfried, J., "Programming Languages for Industrial Robots", Springer-Verlag, Berlin, 1986.

77 Anon, "Languages and Methods for Programming Industrial Robots", IRIA Int. Seminar, Rocquencourt, June, 1979.

78 Volz, R.A., "Report of the Robot Programming Language Working Group: NATO Workshop on Robot Programming Languages", IEEE Journal of Robotics and Automation, Vol. 4, No. 1, February 1988.

79 Souza, C. O., "Aspects to achieve standardised programming interfaces for industrial robots", Proc. of 13th Int. Symp. on Industrial Robots, Chicago, April 1983.

80 Dombre, E. et al "A CAD system for programming and simulating robot actions. Digital Systems for Industrial Automation", Vol. 2, Dane Russak, New York, 1984, pp 201-226.

81 Sata, T. et al, "Robot simulation system as a task programming tool", Proc. of 11th Int. Sym on Industrial Robotics, Tokyo, October, 1981, pp 593-602.

82 Robinson, D.T., "Industrial Digital Motion Controller Requirements", Power Conversion and Intelligent Motion, US, Vol. 14, No. 8, August 1988, pp 29 - 36.

83 Anon., "Programmable Motion Control Systems", BEI Electronics, INC., General Control Systems Division, 564 Alpha Drive Pittsburgh, PA 15238.

84 Schultz, W., "Servo Motion Controller Boards - A Market Survey", Parts I and II, Power Conversion and Intelligent Motion, US, 1988, Vol. 14, No. 8, August, pp 70-73 and No. 9, September, pp 50-54.

85 Anon. "Software gearbox controls mechanical linkages without physical contact", Drives Motors and Controls, November 1988.

86 Webb, M., "Building Sophisticated Motion Control Systems with a High-Level Computer Language", Seminar - What's New in Electric Motion Control, Drives and Controls Conf., Coventry, 10 March 1988, pp F1 - F6.

87 Fenney, L. et al, "Modular Machine Systems", IMechE Conference on High Speed Machinery, London, November 1988, pp 19-28.

88 Kohn, E.F., "A.C.- D.C. Drive Systems - Past, Present and Future Trends", Drives, Motors and Controls Conference Proceedings, 1983, pp 31-32.

89 Robinson D.T., "Digital Servo Control using the IBM PC", Seminar - What's New in Electric Motion Control, Drives and Controls Conf., Coventry, 10 March 1988, pp G1 - G7.

90 Hughes, I.E., "Motion Control Basics for Programmable Controllers", Sixth Annual Control Eng. Conf., Rosemount, IL, May 19-21, 1987, pp 309-315.

91 Stringer, H., "Industrial computers in machine control", Proc. 2nd Int. Conf. Machine Control Systems, IFS, May 1987, pp 139-164.

92 Prtak, L., "The 'work-a-day' PC", Microsystem Design, May 1989, pp 10-11.

93 Pinto, J.J., "Use the Personal Computer to Solve Industrial Control System Problems", Sixth Annual Control Eng. Conf., Rosemount, IL, May 19-21, 1987, pp 391-400.

94 Lawrence, A., "Shopfloor PC Power", Industrial Computing, February 1988, pp 43-45.

95 Bridge, A., "Will a PC do the Job?", Industrial Computing, April 1987, pp 22-25.

96 Anon., "Industrial Computer Source Book 1989", Industrial Computer Source, San Diego, California, USA, ISBN 0-9290069-05-6.

97 Wick G.L., "How to Implement Real-Time Unix for the PC Bus", Seventh Annual Control Eng. Conf., May 1988, Section IX, pp 12-29.

98 Caldwell F.E., "Personalized Computer Solutions", Sixth Annual Control Eng. Conf., Rosemount, IL, May 19-21, 1987, pp 359-363.

99 McMath, R.T. and Tramel, J.R., "Microcomputer-Based Factory Information and Control Systems", Seventh Annual Control Eng. Conf., May 1988, Section I, pp 14-20.

100 Nemeth, E., "Control Flexibility for Today's Manufacturing Environment", Sixth Annual Control Eng. Conf., Rosemount, IL, May 19-21, 1987, pp 334-343.

101 Northcott, J., "The Impact of Microelectronics", Section 4, Forms of Use, PSI Research Report 673, Pinter 1988, pp 56-65.

102 Northcott, J., "The Impact of Microelectronics", Summary, PSI Research Report 673, Pinter 1988, pp 1-24.

103 Shenton, J., "PC or Not PC? That's the Question", Control and Instrumentation, January 1986, pp 47.

104 Anon., "A-B OSAI Numerical Control Series 8600: For Robots, Production Lines and Special Machines", Technical Specification 45002117B, OSAI A-B Ltd., Poole, Dorset.

105 Prevett, S., "Combining Machine Control with Process Intelligence", Drives and Controls, March 1988, pp 17-20.

106 Anon., "Programmable Controllers", Industrial Computing, September, 1986.

107 Northcott, J., "The Impact of Microelectronics", Section 2, Extent of Use, PSI Research Report 673, Pinter 1988, pp 29-42.

108 Northcott, J., "The Impact of Microelectronics", Section 5, Benifits and Difficulties, PSI Research Report 673, Pinter 1988, pp 66-77.

109 Boake, A., "Tying the office to the factory floor", Seventh Annual Control Eng. Conf., May 1988, Section XV, pp 1-9.

110 Melynk S.A., "Shop Floor Control Systems", Dow Jones-Irwin, 1985.

111 Van Dyke Paranuk, H. and White, H. F., "A Factory Synthesis of Reference Models", ITI TR 87-29, Industrial Technology Institute, PO Box 1485, Ann Arbor, MI, USA.

112 Weston R.H., "Generally Applicable Cell Controllers and Examples of Their Use", Commissioned chapter of book on "Cell Control" to be published late 1990 by Springer V., Ed. D Williams and P Rogers.

113 Greenwood, N.R., "Industrial Computers for Cell Control", 16th ESD/SMI Int. Conf. Programmable Controllers, Detroit, Michigan, 1987, pp 75-84.

114 Allmendinger, G., "Cell Control Evolution", Sixth Annual Control Eng. Conf., Rosemount, IL, May 19-21, 1987, pp 80-83.

115 Yingst. J.C., "A Distributed System Architecture for a Personal Computer to Programmable Controller Interface in Process Control Operations, Sixth Annual Control Eng. Conf., Rosemount, IL, May 19-21, 1987, pp 617-621.

116 Alvers, J., "Personal Computers and Programmable Controllers Coexistence on the Factory Floor", Seventh Annual Control Eng. Conf., May 1988, Section XV, pp 23-34.

117 Kendricks, L.E., "An Approach to Using Personal Computers and Programmable Controllers for Flexible Cost-Effective Control Systems", Sixth Annual Control Eng. Conf., Rosemount, IL, May 19-21, 1987, pp 622-626.

118 Weston R.H., "Integration Tools Based on OSI Networks", SME Technical Paper MS89-708, 1989.

119 Anon, "The CIMPHONY Pilot Project Technical Papers", Philips, CFT-CAM Centre, 5600 MD Eindhoven, 1987.

120 Barsamian, J.A. and Patel, D.J., "Benefits of Integrated MIS with Advanced Process Control", Sixth Annual Control Eng. Conf., Rosemount, IL, May 19-21, 1987, pp 144-152.

121 Mack, D., "Serial Communications- or Is Anybody Out There?", Drives and Controls, November 1988.

122 Jasany, L.C., "PLCs Into the 1990s", Automation, USA, April 1989, pp 20-24.

123 Scholten, C.A. "The Missing Link in CAM Systems", Paper CFT-B-35/87EN, 1987, Philips Centre for Manufacturing Technology, 5600 MD Eindhoven, The Netherlands.

124 Challis, H., "Where is Robotics Going?", Automation,UK, May 1989, pp 23-29.

125 Forsyth, W., "A Flexible Automation Controller", Proc, 2nd IFS Int. Conf. Machine Control Systems, pp 175-186, May 1987.

126 Quatse, J.T., "Programmable Controllers of the Future", Control Engineering Magazine, January 1986.

127 Griffiths, C., "Cable Controllers Spread Their Net", Control and Instrumentation, January 1986, pp 39-40.

128 Rubin, S.E., "Sliding Up the Bannister of Technology: A Look a Programmable Controllers, Personal Computers and Process Control", Seventh Annual Control Eng. Conf., May 1988, Section III, pp 6-14.

129 Weston R.H. et al., "Derivation of a Generalised Approach to Designing Control systems for High Speed Machines", SERC Research Grant Application, February 1990.

130 Bonner D.T., "Traditional Information Technology", PLCs vs PCs", Proc., 2nd Int. Conf. Machine Control Systems, pp 129-138, May 1987.

131 Ruckman, P.R., "When I can talk to the Outside World, what will I say?", Sixth Annual Control Eng. Conf., Rosemount, IL, May 19-21, 1987, pp 114-120.

132 Anon, "IEC Draft Standard for Programmable Controllers", International Electro-technical Commission, Tech. Committee 65, January 1987.

133 Anon., "New Concepts for Industrial Programmable Control Systems on the G64 Bus", G-64 Only, Vol. 3, No. 4, Winter 1987, pp 15-19.

134 Ward, T.W. and Mellor, S.J., "Structured Development for Real-Time systems", Vol. 1, Chapters 1 and 2, Yourdon Press, Prentice-Hall, 1985.

135 Legge, N.L., "A Comparative Analysis of Distributed Control Systems from an End Users Perspective", Seventh Annual Control Eng. Conf., May 1988 Section III, pp 23-31.

136 Fischer W., "Real-time computing demands standards", Computer Design, October 1988, pp 77-80.

137 Anon., "Software Tools", Professional Engineering, Mech. Eng. Publications, September 1989.

138 Paranas, D.L., "The Use of Precise Specifications in the Development of Software", Information Processing 77. Proceedings of the Int. Federation of Information Processing Congresses, New York, 1977.

139 Simons, G. L., "Introducing Software Engineering", NCC Publications, 1987.

140 Glad, S.A., "Software Engineering Guide for Real-Time Control Systems Development", Sixth Annual Control Eng. Conf., Rosemount, IL, May 19-21, 1987, pp 188-199.

141 Sauter J.A., "The Role of Standards in the Development of Workstation Controllers", Sixth Annual Control Eng. Conf., Rosemount, IL, May 19-21, 1987, pp 98-106.

142 Titus, J. "Industrial Buses", Engineering Design News, February 1987, pp 114-126.

143 Epton, J., "Breaking Down Communications Barriers", Control and Instrumentation, January 1986, pp 49-51.

144 Northcott, J., "The Impact of Microelectronics", Section 6, Skills and Training, PSI Research Report 673, Pinter 1988, pp 78-84.

4
Problem Oriented Approaches to Machine Control

4.1 INTRODUCTION

Two trends are now having a major influence on control system design. Firstly as illustrated in chapter three hardware costs have dropped dramatically, function for function, while software implementation costs, being labour intensive continue to rise. Secondly as discussed in chapter two the demands for higher manufacturing quality, greater productivity and responsiveness have in turn demanded more sophisticated automation. Much more is now demanded from control systems and there is a need for more efficient engineering methods which can help to coordinate the efforts of the machine or process designer and the control system engineer.

This chapter looks at problem oriented approaches to machine control system design and implementation within the factory environment. A problem oriented approach uses methods related to the nature of the problem and is not compromised by technological limitations. Obviously the state of the enabling technology available must fully meet the requirements of the problem in question to allow such an approach to be adopted [1].

Models can be created to represent a machine control system as part of its factory environment. Similarly the internal behaviour of a controller can be modelled. Such models can provide order and structure to the design and implementation of systems. The concepts behind system modelling are discussed and an evaluation is made of existing factory reference models, generalised architectures for machine control and their implementation.

Within the disciplines of computer science and software engineering, methodologies have evolved to aid the generalised modelling and structured development of real-time embedded systems [44]. These methodologies embody the concept of supporting the stages of a control systems life cycle not only through design and implementation but also to enable effective system maintenance and enhancement. Recently activity in the emerging field of Computer Aided Systems Engi-

neering (CASE) has seen the creation of software tools which can automate the application of these methods. There would appear to be significant potential for the application of such approaches to the development of machine control systems.

Programming languages for industrial machine control systems are evolving along multiple paths driven by regional influences, controlled machine architecture, end use, industry use and software technology. Methods for the efficient representation of machine control problems are considered. If the method of problem representation reflects the problem being tackled both in terms of system structure and programming method then significant advantages might be gained in developing, maintaining and integrating machine control systems.

4.2 CONTROL SYSTEM MODELLING

4.2.1 Introduction

Additional to the physical structure of a given control system are also the less easily visible information and application structures, which should exist within the system software and require simultaneous planning during systems development. For any non trivial system the design and management of these software structures is a very complex task and in order to make these activities easier to cope with supporting methods and tools have been proposed by a number of research groups.

Models can be created to help with problem visualisation and to enable the implementation of control systems to defined structures or architectures. The scope and purposes of modelling techniques is extremely wide ranging.

4.2.2 Reference Models for Manufacturing Control

In recent years industrial and academic research groups have proposed "reference models" for manufacturing which encompass machine control. The scope of these reference models varies widely but in essence they provide "conceptual frameworks" for the manufacturing environment.

Numerous manufacturing reference models have appeared recently, in support of efforts for standardisation or as aids to specific factory management or control system implementations. They aim to provide an ordered description of essen-

tial system requirements which can in turn guide designers of both hardware and software tools to support their efficient implementation.

The following paragraphs provide brief summaries of known manufacturing reference models and associated system building tools:

CAMI — Computer Aided Manufacturing International model is a aimed at facilitating integration in discrete parts manufacturing [2].

DEC1&2 — The Digital Equipment Corporation has produced models aimed at control, manufacturing and marketing [3].

CIMOSA — ESPRIT project 688 "CIM-OSA" aims to provide an Open Systems Architecture which covers all the information needs of all functions in a manufacturing enterprise. CIM-OSA provides a reference framework made up of two parts. A Reference Architecture which contains generic building blocks and a Particular Architecture that contains particularised building blocks for a specific enterprise [4].

FOXBORO — As a supplier of industrial process control system, Foxboro has produced a detailed analysis of manufacturing functions [5].

HUGHES — A management and information architecture developed at Hughes for internal operations [6].

ICAM — United States Air Force Integrated Computer Manufacturing Program provides a conceptual framework for integrated manufacturing in the aerospace industry. A set of implementation projects aim to make these ideas a reality by the mid-1990s [7].

ISO — The International Standards Organisation aim to create a multidimensional open-ended reference model to aid long-range planning for standardisation through the identification of interfaces between and characteristics of system automation elements [8].

NBS[1] — The US National Bureau of Standards has developed a hierarchical control architecture in support of its Automated Manufacturing Research Facility [9]. See section 4.3.2.

ITI — The ITI work promotes the idea of *connectionist* models for factory systems. These are naturally distributed and parallel thus offering a natural fit with the distributed computational resources available with modern control hardware and software techniques. The connectionist model encourages the development of low level primitives and provides a promising approach to the creation of a tool kit for generic control [10, 11].

[1]Now renamed NIST (National Institute for Standards Technology).

NAYLOR	Mathematical models for manufacturing software [12].
PLAIC	The Purdue Laboratory for Applied Industrial Control has produced reference models for the process industries [13].
GRAI	Doumeingts at the RAI Institute of Grenoble has identified physical, information and decision structures in a model for manufacturing systems [14].
FFS	The Japanese Future Factory System initiative takes a *holonic* approach to manufacturing systems. A holon is an entity which has a double character to permit autonomy of a unit and order of a whole. The basic idea is developed from making an analogy between the holonic hierarchy of a living body and the structure of a factory enterprise or processing machine [15].
AUTOMAIL	Automail is an "integration shell" which comprises of a distributed system application language and a set of configuration management, concurrent task management and debugging tools. It does not seek to impose any particular application structure but supports the concepts of separate communication, application and information architectures [16].
MKS	Manufacturing Knowledge System is a computational framework for CIM which embodies an object-oriented methodology for applying knowledge systems in an advanced manufacturing environment. The test bed for this research is a semiconductor fabrication line at Stanford's Centre for Integrated Systems (CIS) [17].

4.2.3 Reference Model Construction

Parunak [18] discusses the essentially intuitive concepts behind the construction of reference models. The process is one of developing a set of generic requirements and then abstracting a generalised model from these. The model formed can then be used for reference in building specific systems. Figure 4.1 shows how this concept can lead to the development of a series of reference models at different levels of specificity, through a series of *abstractions* followed by a series of *syntheses*. Section 5.5 describes the formation of the UMC reference model.

4.2.4 Control Hierarchy

The most common structure, and the one represented in most factory reference models [18], describes the control of a system through a series of hierarchical layers. Table 4.1 illustrates the trends that emerge in such hierarchies.

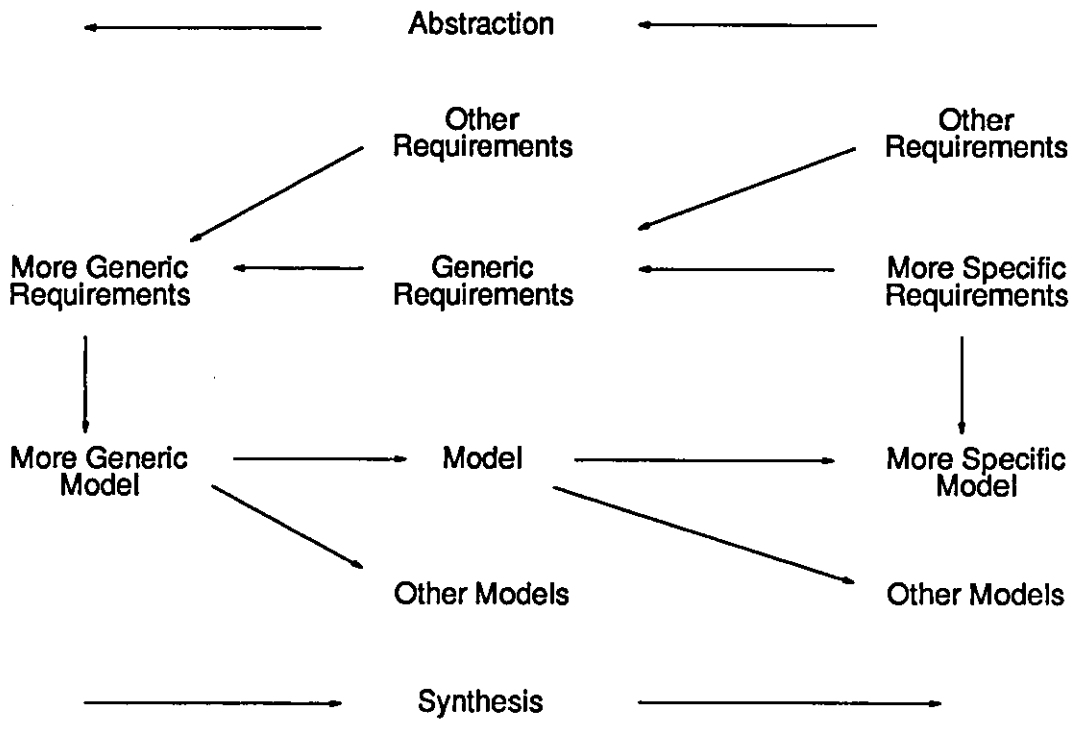


Figure 4.1. The concepts of model abstraction and synthesis. Source: Parunak.

	Relative Position in Hierarchy	
	Top	Bottom
Control Scope	Wide	Narrow
Time Constant	Long	Short
View of Data	Summarised	Detailed
Predominant Activity	Planning	Execution

Table 4.1. Trends in Hierarchical Models.

Various control hierarchies have been suggested by different research groups ranging between the extremes of completely *open hierarchies* and completely *closed hierarchies.*

4.2.5 Open Hierarchy

An open hierarchy can have any number of levels, depending on the needs of the specific application. The ITI control hierarchy for example allows an arbitrary number of levels [11]. Thus the number of levels need not be specified in the reference model. Instead, the emphasis is on generic behaviour which every level must provide. Such a hierarchy of generic entities may be termed a fractal description [19]. Where practical such methods permit the development of a common software framework for every level and can thus greatly reduce the programming effort needed to implement a control hierarchy. See figure 4.2. Section 5.5.2.2 describes the concept of a completely open hierarchy for the UMC task structure.

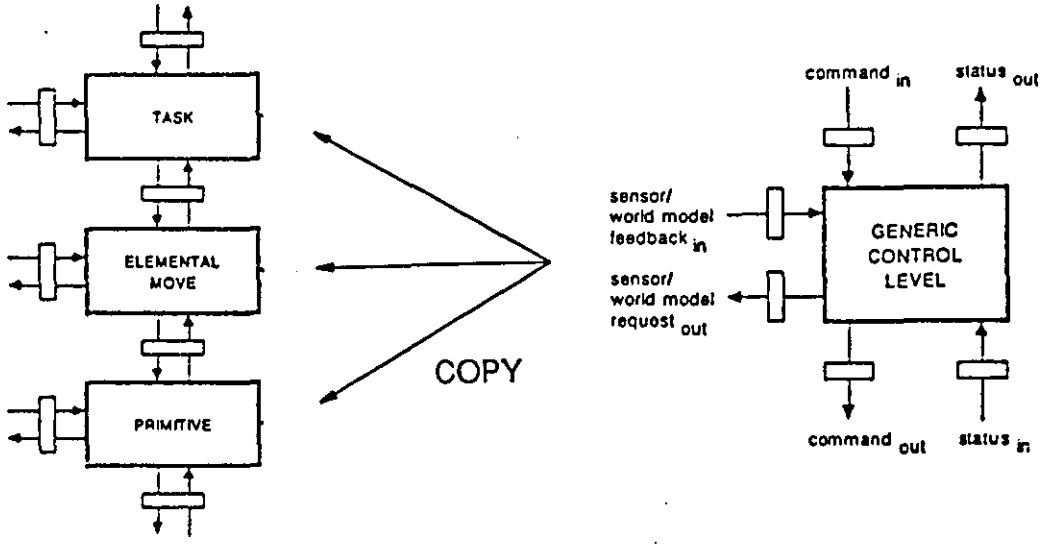


Figure 4.2. A generic control hierarchy. Source: NBS.

Open hierarchies typically identify functions that are common to all or most levels. The NBS model has three components at each level. Section 4.3.2 describes the implementation of the NBS model by a number of research groups.

The following list summarises the generic functions which have been proposed by various published open hierarchies [18]. Each level is responsible to;

- decompose commands that it receives into sub-tasks for its subordinates;

- schedule the execution of sub-tasks by its subordinates;
- dispatch sub-tasks to subordinates;
- predict feedback from subordinates;
- analyse feedback from subordinates;
- report conditions to its supervisor; and
- complete its assigned task.

4.2.6 Closed Hierarchy

A closed hierarchy has a fixed number of levels, with specific functions bound to each level. For example the physical structure or organisational requirements of a control system may required that specific functions are associated with particular levels.

Figure 4.3 lists the levels that appear in a number of factory reference models and indicates which models have which levels. The double-headed arrows indicate a single level in one model that spans two or more levels in another model.

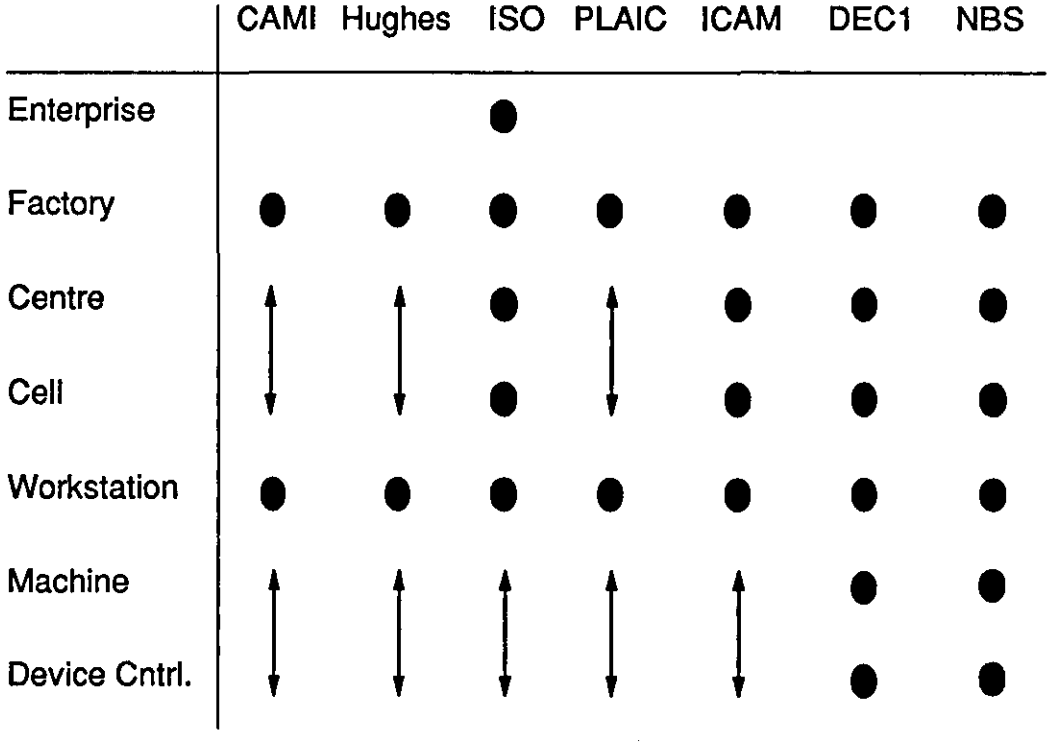


Figure 4.3. Factory model levels. Source: Parunak.

Although there is general consensus, not all of these models use the same terminology for the various levels. The following definitions are used in figure 4.3:

Enterprise	Geographically separate production facilities.
Factory	All operations at one geographical location.
Centre	An area within a factory that is managed as a unit.
Cell	Two or more workstations integrated to perform a group of related processing steps.
Workstation	A closely-coupled collection of equipment that performs a single distinct step in the manufacturing process.
Machine	A single piece of equipment, with a single external control interface.
Device Control	A component of a machine with its own control interface.

It is interesting that in the models considered only DEC1 and NBS include separate machine and device control levels. Most have concentrated their efforts at the higher levels and simply view machines externally without specifying their internal application or information structures.

4.2.7 Hierarchy Evaluation

The difference between open and closed hierarchies is not one of right or wrong. Often the most generic model of a system can be formed using an open hierarchy. As more specific functions are added to specific layers a hierarchy becomes closed. According to Parunak, open hierarchies are typically pure control structures, while closed hierarchies are usually a hybrid of control and organisational requirements [18].

4.3 ARCHITECTURES FOR REAL-TIME CONTROL

4.3.1 Introduction

Having considered the concept of factory reference models this section considers the more specific modelling and implementation of structured machine control systems. These are typically open control hierarchies with application dependent structures which may be distributed.

4.3.2 Hierarchical Control Architectures

One of the best known hierarchical architectures for real-time control was suggested by Albus et al at the Centre for Manufacturing Engineering, NBS Washington [9]. Their approach involves an open hierarchy of modules in defined layers. NBS have looked at the theory and practise of hierarchical control over the last decade. Their work covers factory models previously discussed in section 4.2.2 and includes the implementation of real-time machine controllers conforming to these models, based on finite state techniques [20, 21].

In the NBS architecture high level goals are decomposed through a succession of sub-goals with progressively simpler commands at the lower levels. The bottom level interfaces with the sensors and actuators. Each control level is a separate process with a limited scope of responsibility. Each level performs the generic control function of sampling its inputs and generating appropriate outputs. See figure 4.4. The input is characterised by three types of data, a command from the next higher level, process data from the same level and status feedback from the next lower level. The outputs are of three types, a command to the next lower level, a request for data at the same level and a status feedback to the next higher level.

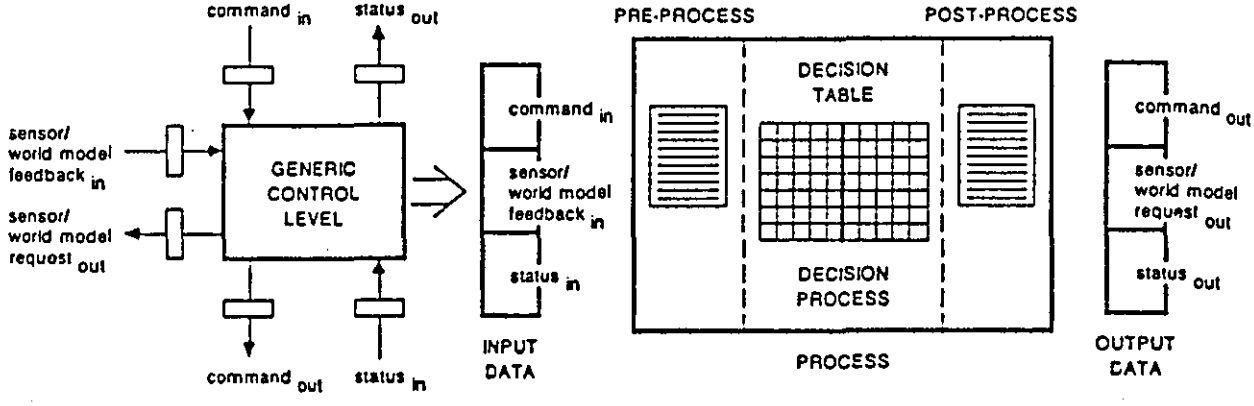


Figure 4.4. Generic control level in NBS computational hierarchy. Source: Albus.

The basic command and control structure is a tree in which each module has a single supervisor and one or more subordinate modules. As illustrated in fig-

ure 4.5 the computational "chain" of command from the bottom to the top of the organisational tree can be further segmented into three hierarchies:

(1) goal, or task decomposition (H)

(2) feedback processing (G)

(3) world model (M)

At all levels in the system the H, G and M modules are executed on a repetitive clock based cycle in order to ensure that the response of the system is deterministic. This model has been implemented at NBS [21] and adopted by Philips for their CIMPHONY factory integration project [22]. Hormann suggests a critical point in the implementation of real-time control systems using this architecture is the communication overhead within or between levels of the hierarchy [23]. Also since finite state tables contain no sequence information it is difficult to implement sequential tasks using the NBS control system [21].

Lent [24] has developed the concept of dataflow driven control system aimed at the machine tool market. This has similarities to the NBS implementation philosophy. The control program is split into processes, each of which operates like a finite state machine. The execution of a specific process is triggered unconditionally as soon as any of its input data changes in value. The operation of such a system has a strong analogy to hardwired logic circuitry. It offers an efficient implementation method for machine controllers of relatively fixed configuration.

4.3.3 Distributed Control Architectures

To overcome the disadvantages of a purely hierarchical system, Paul [25] has proposed a control system consisting of a distributed network of sensing, action and reasoning agents working in their own domains with a coordinator which integrates local knowledge. Shin [26] has considered the communication requirements for five classes of industrial process in the context of multi-robot systems. These range from independent to tightly coupled processes. Modular control systems for multi-robot or multi-actuator systems have been considered by several research groups [27, 28, 29] and such systems are now seeing initial commercial exposure [30, 31]. The concept of splitting problem representation by division into separate processing tasks is one well accepted and used commercially.

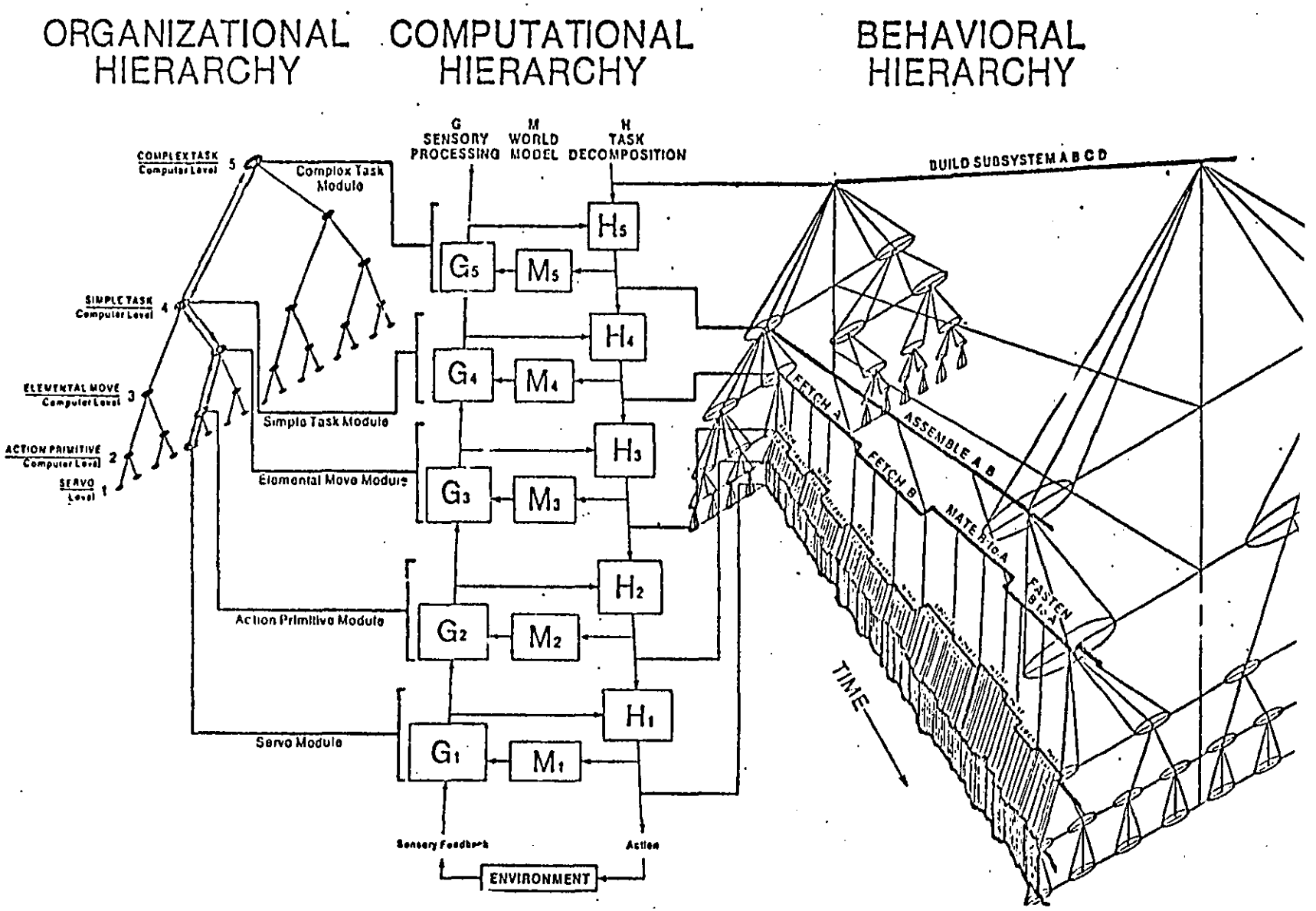


Figure 4.5. Three aspects of the NBS hierarchy. Source: Albus.

Taylor has been engaged in the real-time requirements for control systems at IBM [32]. His system consists of an interactive programming tool connected through shared memory to a multi-processor real-time system that performs time-critical operations. A major feature of this system architecture is the ability to configure the system to control a wide variety of sensor-based programmable automation equipment. The behaviour of the real-time system is determined through data flow graphs which are initialised and manipulated by application programs. The building blocks for these data-flow graphs are real-time application subroutines known as *verbs* which include device I/O, control laws and trajectory planners. See figure 4.6. Once defined verbs can be used or combined to form *compound verbs* and may be used to describe serial motion expressed as a graph. Taylor finds that such programs map more naturally onto graphs than onto conventional textural programs [33]. See also section 4.5.5 which discusses function chart programming.

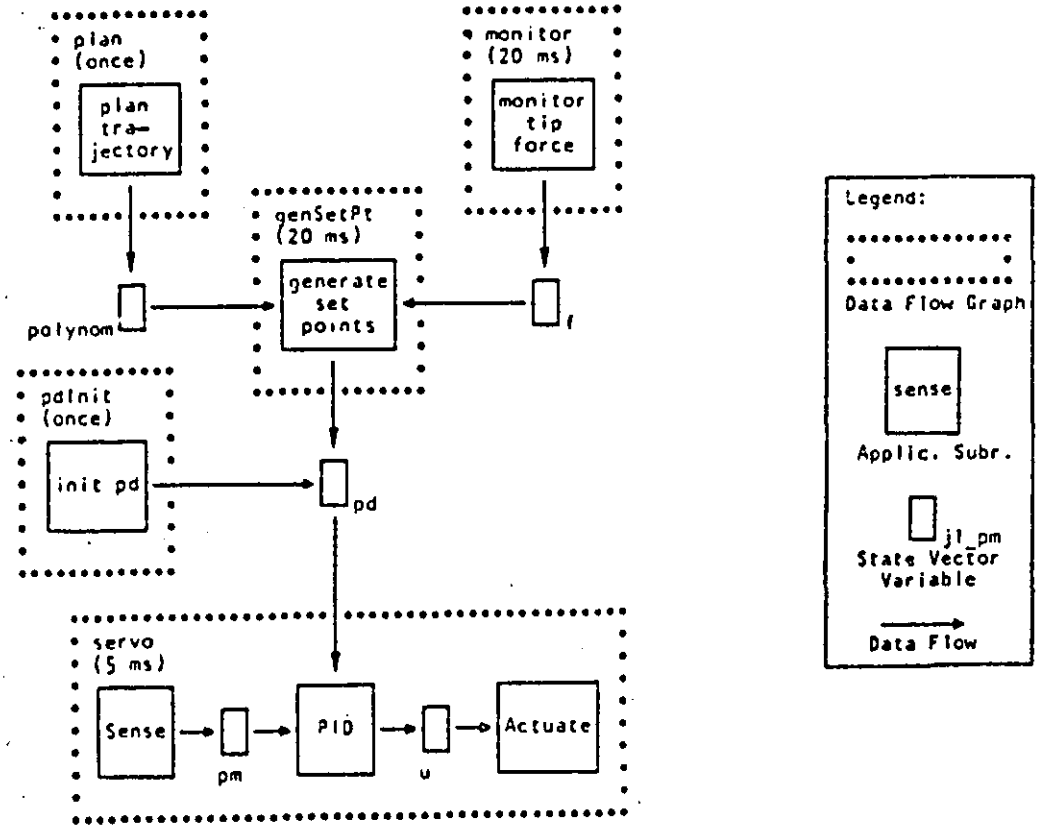


Figure 4.6. Example of Taylor's data flow graphs. Source: Taylor.

A quite different approach is taken by Bharu [34], in his research using a neural network structure for the implementation of a robot control system which currently consists of 19 processors. The approach called Control using Action Oriented Schemas (CAOS) aims to obtain intelligent control by simulating the functions of the biological nervous system. The main limitation associated with systems based on neural networks is their relatively simple structure compared to the biological nervous system which currently limits their capabilities to simple tasks [22].

Advanced work has been done on distributed fault tolerant industrial process control. PEARL [35] is a highly fault tolerant distributed system but Shin [26] reports that the results of this research may be difficult to apply where tight synchronisation of devices is required in for example many motion control applications. Mars [36] is a distributed fault-tolerant system offering deterministic performance. The Mars project started in 1980 at the Technische Universitat Berlin. The first prototype appeared in 1984 and demonstrated the fundamental concepts of Mars. The second academic prototype developed at the Technische Universitat Wien in Vienna has been functional since 1988. The main feature of Mars is deterministic performance under a specified peak load [37]. Its main industrial applications include rolling mills in which the controller system imposes hard deadlines and the system is capable of supporting tight motion synchronisation. In the UK, Holding [38] has similar research interests in the design of fault tolerant distributed systems and the control of high speed machine drives.

The implementation of distributed computing environments for real-time machine control is heavily related to the provision of a suitable operating system. Meglos, an AT&T research environment for robotics and general machine control [39], MARS mentioned above, and TRON [40] a new Japanese operating system designed to enable the integration of real-time devices appear particularly significant in this respect. See also section 6.9.5.

4.4 STRUCTURED SYSTEM DEVELOPMENT

4.4.1 Introduction

This section focuses on generalised approaches to the development of control system software. From their origin in computer science many and varied methods and associated software tools are currently used for the design and analysis of business and real-time systems software [42]. Such methods are widely used for

large embedded control systems although as yet rarely applied to the implementation of industrial machine control systems. A major drawback to the wider utilisation of these methods is that they are often confusing to untrained users.

In the field of software engineering, development methodologies have evolved initially for structured programming and more recently for systems definition, design, implementation and management in order to support the various phases in a control systems life cycle. The automation of such structured development methodologies in the form of CASE tools is currently the focus of extensive research activity [41]. See section 4.4.4.

4.4.2 System Life Cycle

The phases in a system life cycle are those it must pass through to achieve its goal. The structured development of any control system typically follows a series of sequential phases covering its definition, refinement, installation, run time activity, maintenance and modification [42]. Interestingly ICAM, ISO and CIMOSA embody the concept of the system life cycle into their factory reference models [18].

From a systems engineering perspective the role for a life cycle is to provide structure and order to the process of developing a system through time. Most approaches adopt a series of *stages* through which the system moves, and a set of *tools* that facilitate this movement. The term tool is used very widely in this context to include the overall concept and general architecture for the system to be implemented, documentation systems, project management systems, reference models and recognised standards. The tools are any entities which assist the project through its realisation stages [43]. The number of life cycle stages and the terms used to describe them vary but a generalised set based on the suggestions of [44] and [18] are:

- Requirements Expression
- Requirements Analysis
- Design
- Implementation
- Validation or Test
- Installation

- Operation and Maintenance

- Enhancement and Management of Change

Traditional system development proceeds in a step-wise fashion from problem to analysis, then design, implementation etc. Based on the experience of many real-time systems developers the main problem with conventional approaches to software development is that they are too rigid [45]. In development there is complex interaction and overlap between the various stages. As in other fields of engineering the specification is inextricably intertwined with the nature of the final implementation [46].

McMenamin and Palmer [47] have developed a modelling strategy which clearly distinguishes the *essence* of a system from its *implementation* as follows:

- The *essential model* describes what a system must do and what data it must store regardless of the technology used to implement the system.

- The *implementation model* describes a system as it is actually realised by a particular technology.

Based on the work of McMenamin and Palmer, Ward-Mellor [46] advocate clear separation of the essential and implementation models of a system during the design phase. See figure 4.7. The major benefit of this approach lies in its resolution of the common confusion between implementation dependence and level of detail. The spiral arrow in figure 4.7 suggests another feature of the Ward-Mellor essential/implementation model distinction, namely that the sequence of construction is independent of model content [46]. Thus as each successive level of essential details is defined, the corresponding details of the implementation may be filled in and so on.

The assumption that the specification can be fixed at an early stage in the development process is very rarely achieved even when the best practices are followed [16]. There is inevitably a limited appreciation of the final problem at the onset of a project and necessary enhancements are costly to achieve. Glad [42] provides typical figures for the distribution of effort in the software engineering of real-time control systems. See figure 4.8.

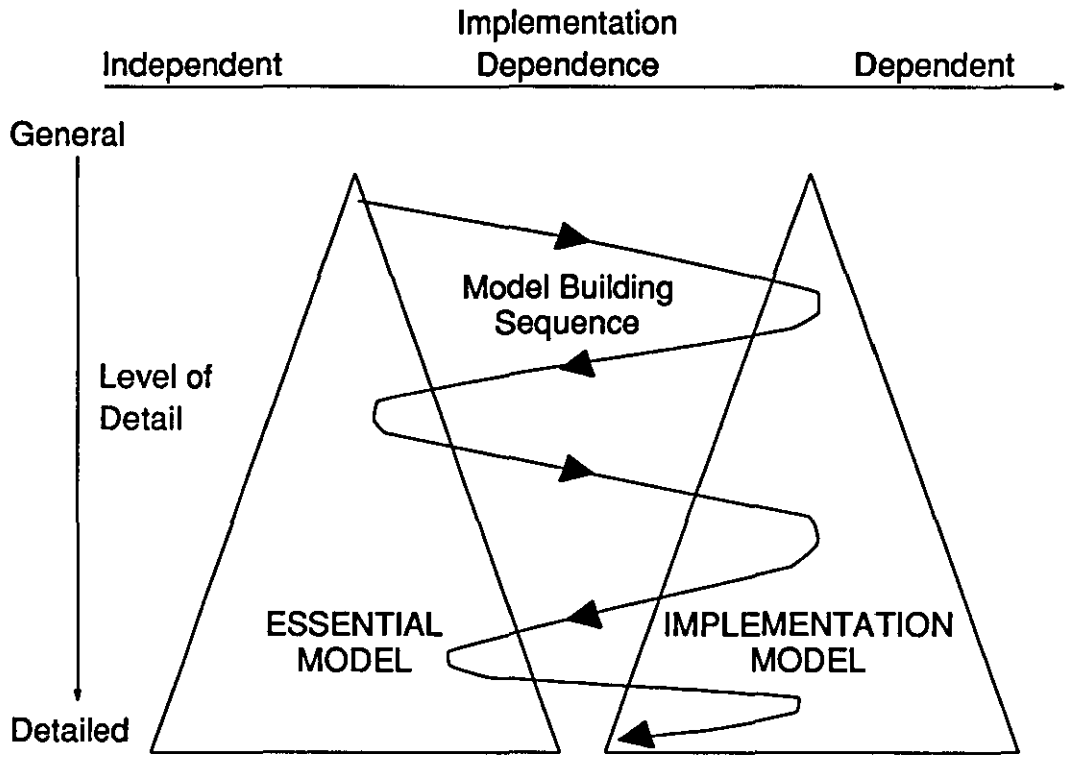


Figure 4.7. Relationship of essential and implementation models. Source: Ward-Mellor.

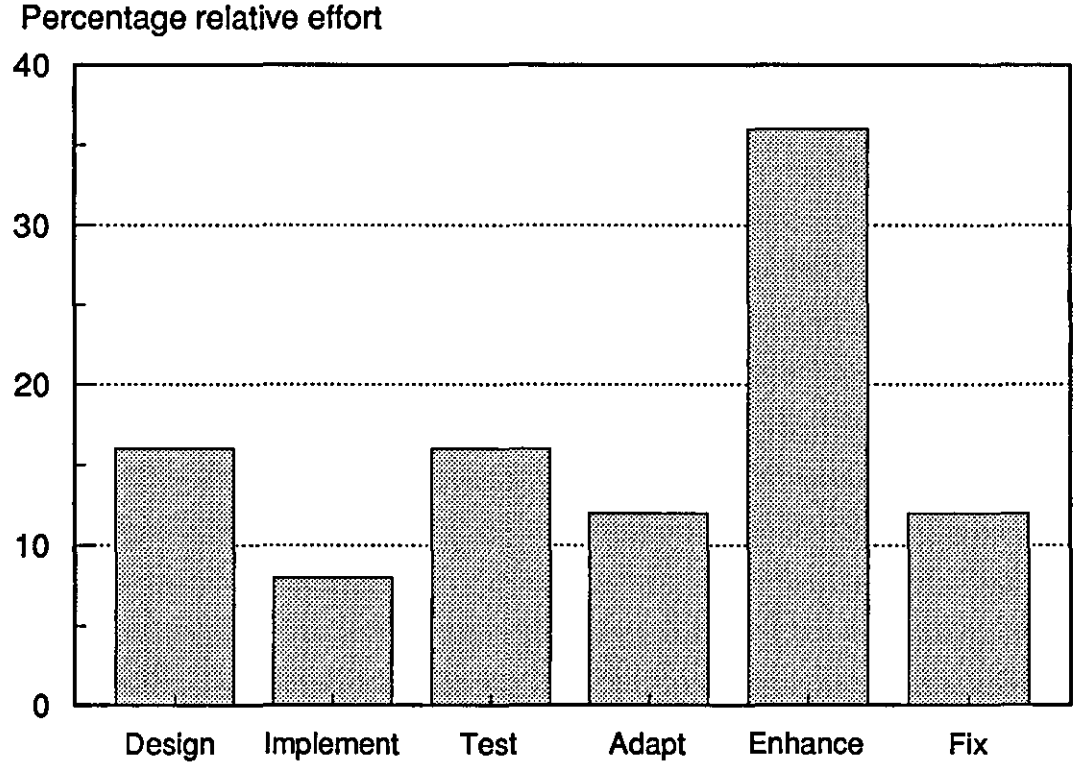


Figure 4.8. Distribution of effort in the software life cycle. Source: Glad.

4.4.3 Methods and Tools for System Design and Analysis

The methods used are numerous and varied but have been categories by Frost [44] into five major groups - CORE, Yourdon, Jackson, Mascot and HOOD. The situation is complicated by the large number of variations on these basic methods. For example there are two major variations on the Yourdon style of structured analysis for real-time systems [48] Ward-Mellor and Hartley, and many others which are only applicable to non-real-time design. These methodologies cover the requirements expression, requirements analysis and design stages of the system life cycle. See figure 4.9.

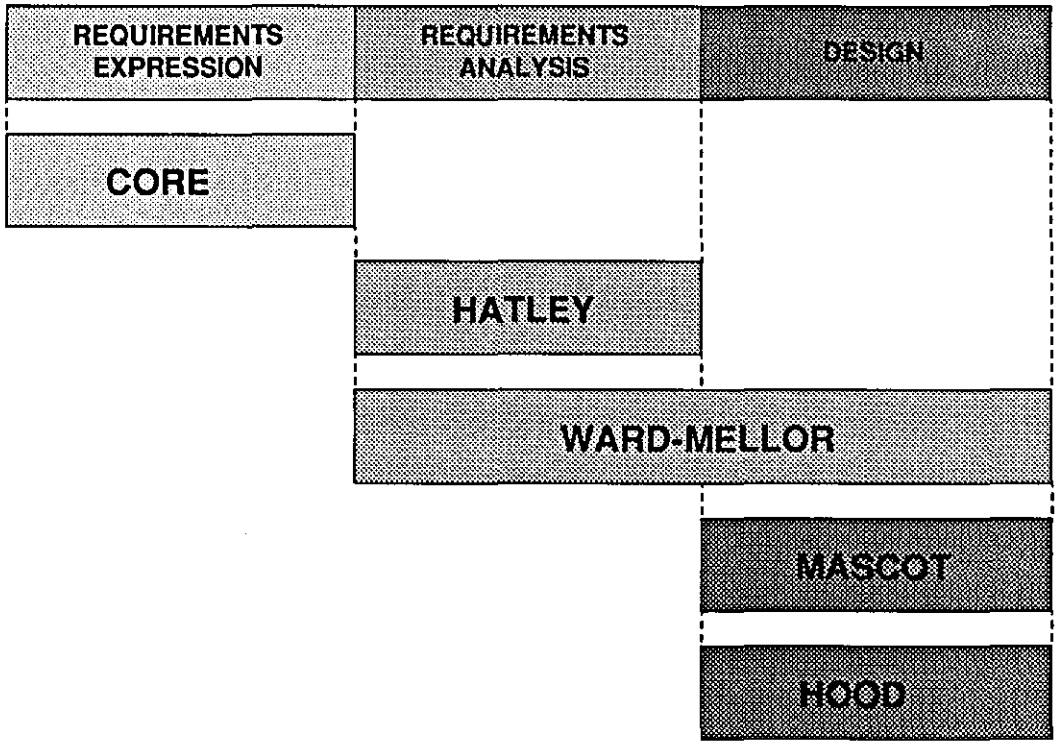


Figure 4.9. Method coverage.

Mascot (Modular Approach to Software Construction, Operation and Test) was originally developed in the seventies at the RSRE. There are two major definitions, Mascot 2 (1983) and Mascot 3 (1987). Each of them *may* be used with an associated real-time kernel onto which the software can be directly mapped [49, 50]. Mascot 2 is a simple design methodology which has limitations [44]. It is however relatively easy to understand and is well matched to modelling the communication mechanisms provided by most real-time operating systems.

HOOD is an Object Oriented Design (OOD) tool which was developed for the European Space Agency. OOD is a concept which is now attracting a lot of interest in the field of software engineering [44]. Many software developers are looking for an effective way of redesigning reusable software components. OOD is regarded as one of the most promising solutions and the concepts also appear particularly applicable to machine control.

4.4.4 Computer Aided Systems Engineering

Modelling methodologies provide guidance for building system models. However during a systems life cycle these models will require frequent modification and change. Models constructed by hand require enormous effort to construct and even more effort to change. Change is however inherent in a system's life cycle and must be made as easy as possible.

This need has led to the development of software tools that automate some of the model building methodologies used in business and real-time systems. In the Computer Aided Systems Engineering (CASE) philosophy the computer provides a development environment to enable the building of system models [51, 52].

The development environment needs to provide a range of tools appropriate to the different participants and stages in a system's life cycle. For machine control system development participants might include managers, manufacturing systems engineers, software and hardware engineers etc. To be considered as a consistent environment the tools must provide a common user interface, use a common database and provide configuration management capability.

CASE systems are now available from a number of vendors [52, 53]. HOOD [54] for example is a CASE environment developed for the European Space Agency providing object oriented design facilities targeted at Ada programming.

There are unfortunately no standards for CASE. Each system includes or omits whatever it wishes based on its own standards and each vendor would like its implementation accepted as *the* standard. In the US the Software Engineering Institute at Carnegie-Mellon and the Centre for Advanced Information Management (CAIM) at Auburn University have emerged to provide CASE guidance. These bodies are working closely with the International Standards Organisation (ISO) and United States of America Standards Institute (USASI) to establish standards for all aspects of information engineering [52].

4.5 APPLICATION PROGRAMMING METHODS

4.5.1 Introduction

Programming languages for industrial machine control systems are evolving along multiple paths driven by regional influences, controlled machine architectures, industrial usage and software technology.

Well documented work has been carried out by many working parties on defining programming methods and in particular language requirements for various categories of flexible machine [55, 56, 57, 58]. The capabilities of alternative programming languages have in turn been evaluated by various research groups [59, 60, 61, 62]. These languages are typically machine specific or based on extensions to an existing general purpose programming language [63].

The capabilities of existing machine programming systems obviously vary widely as do opinions about their effectiveness. Common criticisms however include difficulties in describing the application problem effectively and severe limitations when trying to control external devices not supported by high level functions in the particular language.

It seems likely that no single language will be capable of providing a practical answer to all industrial machine control problems. An alternative is a coherent environment capable of supporting multiple methods of problem representation.

4.5.2 The Nature of Machine Operations

Machine operations typically involve the execution of predefined sequences of actions, or in continuous process control, continuously maintaining predefined input/output relationships. Various approaches have been adopted to define these activities [58, 62, 64].

In most machines several distinct activities need to occur simultaneously and a machines operation can usually be naturally split into a number of application tasks [65]. Each task has its particular requirements and the most natural method of representing this activity to the programmer will therefore vary. See figure 4.10.

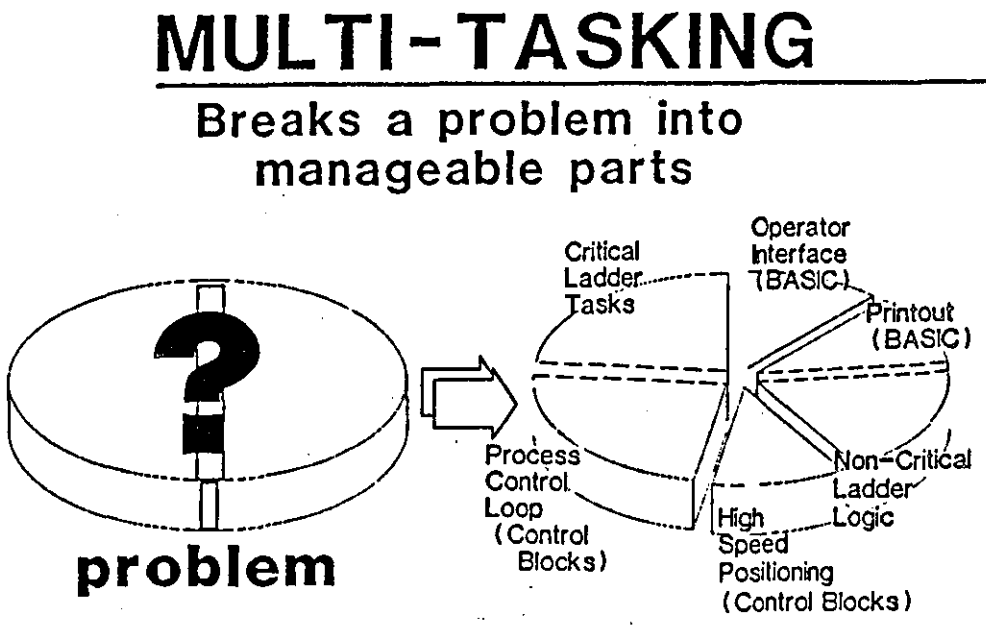


Figure 4.10. Modularising a control problem into concurrent tasks. Source: Notte.

Satisfying the requirements for machine control in a responsive factory environment involves the programming of many complex and varied concurrent tasks. For example the operation of a single machine may involve sequential and combinational binary logic, motion control, management information collection and process diagnostics. The nature of such a complex application is also likely to change as more is learned about the system and product requirements.

4.5.3 Programming Environment

Benedick [66] states that a good programming technique offers the capability of designing the control system from the top down and developing it from the bottom up. This is especially useful when designing and developing industrial control systems where many technical disciplines are commonly involved.

It has been recognised by Volz [57] that software engineers have come to realise that design environments and programming languages go hand in hand. A future control system design environment should include management tools to support multiple designs, source level debugging tools, a consistent and easy to learn interface, and freedom from machine specific details.

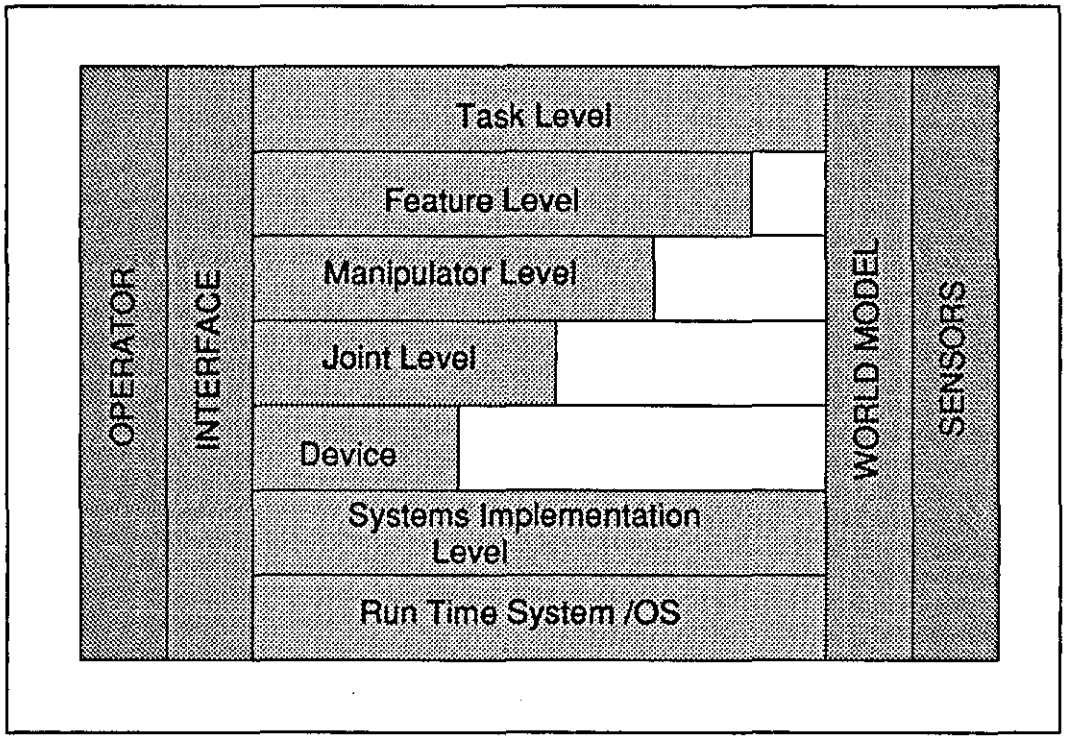


Figure 4.11. Environment for robot programming. Source: Volz.

It is considered extremely important by the IEEE robot programming language working group [57] that programming systems are based on some standard general purpose programming language with full support for software engineering. See section 4.4.4. The working party's concept of a "staircase" structured programming environment is depicted in figure 4.11. Different programming levels are provided in a similar manner to those suggested by Taylor at IBM [67]. Each level is built upon the totality of the levels below it, and the programs written at one level may utilise the commands from any lower level. This feature allows programming languages to be built upon and evolve from a standard system programming language and operating system. The operator can interface with and reprogram the system at all levels and a common world model is suggested for system wide information.

Several researchers including Taylor [67] at IBM and Haynes [21] at NBS have used combinations of graphical and textural programming methods for machine control. The International Electro-technical Committee (IEC) [58] has looked at textural and graphical languages and supports the concept of connectivity of programming languages. See figure 4.12. Basic textual and graphic programming lan-

guages may be used to define function blocks which may in turn define action blocks and translation conditions that are the elements of program execution.

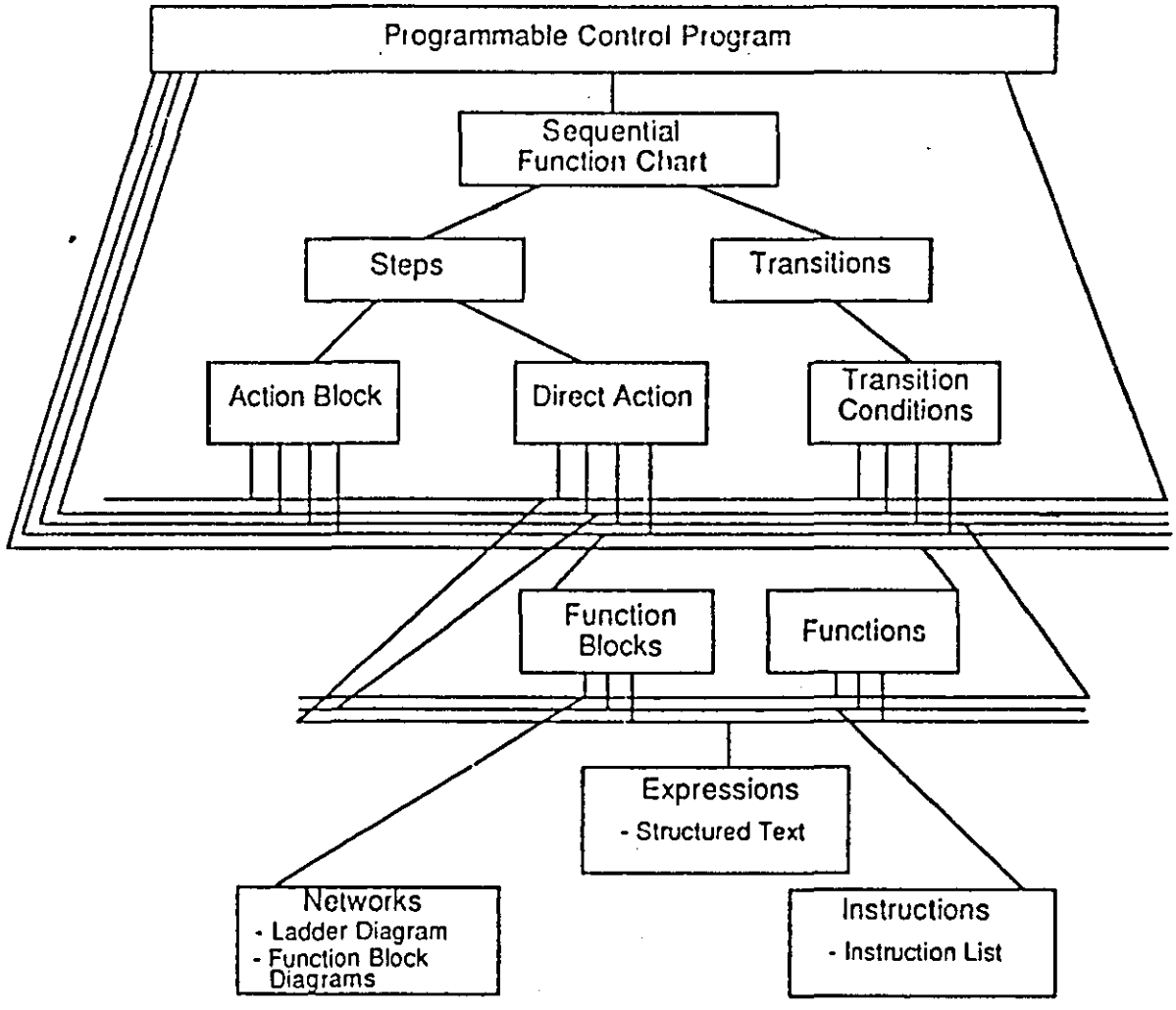


Figure 4.12. Hierarchy of IEC industrial control program elements.

One important aim is to mask the complexity of very sophisticated lower level languages from the higher level users. People with varying programming skills and backgrounds must be able to cooperate in the multi-disciplinary creation of a control system's structure and application programs.

4.5.4 Problem Description Languages

Problem Description Languages (PDLs) are widely used whenever non computer specialists need to work with computers. Some common examples of PDLs include spreadsheets, database languages and the programming languages found on programmable controllers. See table 4.2. What distinguishes problem de-

scription languages from general purpose programming languages is that the PDL already contains a model for a particular kind of problem [68].

Language	Application	Internal Model
Spreadsheet	Matrix Calculations	Office Filing System
Ladder Diagram	Discrete Control	Relay Panel
Finite State Logic	Any Event Driven System	Set of State Diagrams

Table 4.2. Examples of problem description languages.

PDLs are efficient because they try to minimise the demands placed on the programmer. Since a *particular model* is already built into the language the programmer only needs to supply the information about that particular instance of the general problem. In lower level procedural languages both the problem data and the structure of the problem must be supplied by the programmer [66].

Lower level languages are versatile and can be applied to many different kinds of problem. PDLs are problem type specific and there must be a close match between the PDL used (i.e. its internal model) and the problem to be solved if it is to be an efficient programming method. In contrast low level languages can be used for almost any purpose but programming efficiency is relatively poor.

A number of problem description oriented systems have emerged for machine control.

4.5.4.1 Relay Ladder Programming

All ladder diagram programming systems share a common internal model; the relay panel. If the programmers problem is to replace an electro-mechanical relay panel with an electronic controller with no moving parts and which can be easily "rewired" from a keyboard, then the internal model of a ladder program provides an excellent match provided that single bit binary state changes can adequately model the application. The difficulty often experienced with using ladder diagrams for more "advanced" programming applications is due to the poor match which often exists between the problem and the internal model of the relay ladder diagram language [68].

4.5.4.2 Finite State Diagrams

State descriptions share a common model of a set of state diagrams or tables. Using state logic the programmers job is to accurately define all possible states of a system. See table 4.3. Such methods have been extensively used at NBS [21].

Command	State	Feedback	Next State	Output	Report
—	C30	No New Command	C30	Wait	—
Fetch (A)	C30	New Command	C31	Reach to (A)	—
"	C31	Distance to A>T1	C31	Reach to (A)	—
"	C31	Distance to A≤T1	C32	Grasp (A)	—
"	C31	A Not Visable	C35	Search for (A)	—
"	C32	Grasp Pressure <T2 Grip Dist >T3	C32	Grasp (A)	—
"	C32	Grasp Pressure ≥ T2 Grip Dist > T3	C33	Move to (X)	—
"	C32	Grip Dist ≤ T3	C36	Back Up (Y)	Object Missing
"	C33	Distance to X>O	C33	Move to (X)	—
"	C33	Distance to X=O	C34	Release	—
"	C34	Grip Dist <T4	C34	Release	—
"	C34	Grip Dist ≥T4	C30	Wait	Report Fetch Done
"	C35	A Not Visable	C35	Search for (A)	—
"	C35	A In Sight	C31	Reach to (A)	—
"	C35	Search Fail	C30	Wait	Report Fetch Fail
"	C36	Back Up Not Done	C36	Back Up (Y)	—
"	C36	Back Up Done	C35	Search for (A)	—

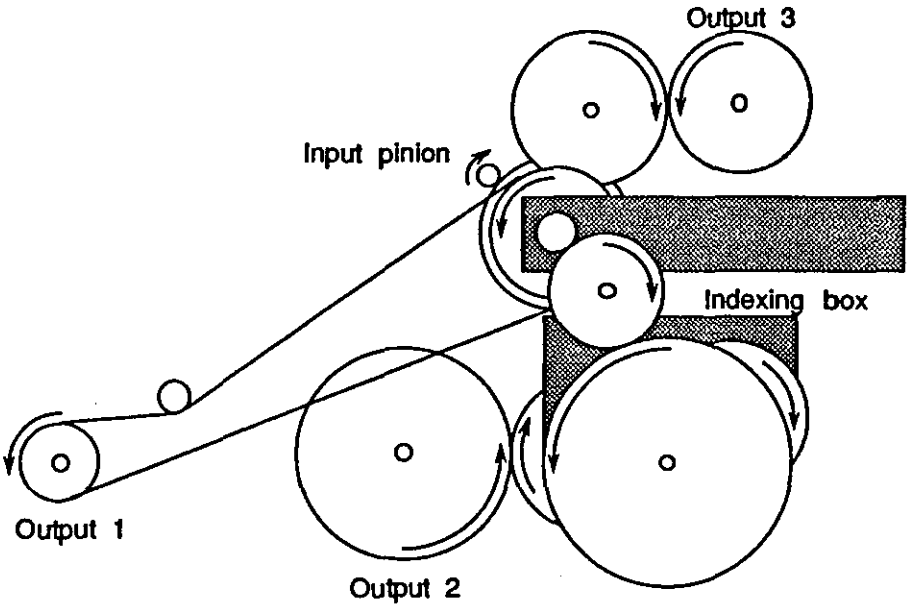
Table 4.3. Finite state table. Source: Albus.

4.5.4.3 Programmable Transmission Systems

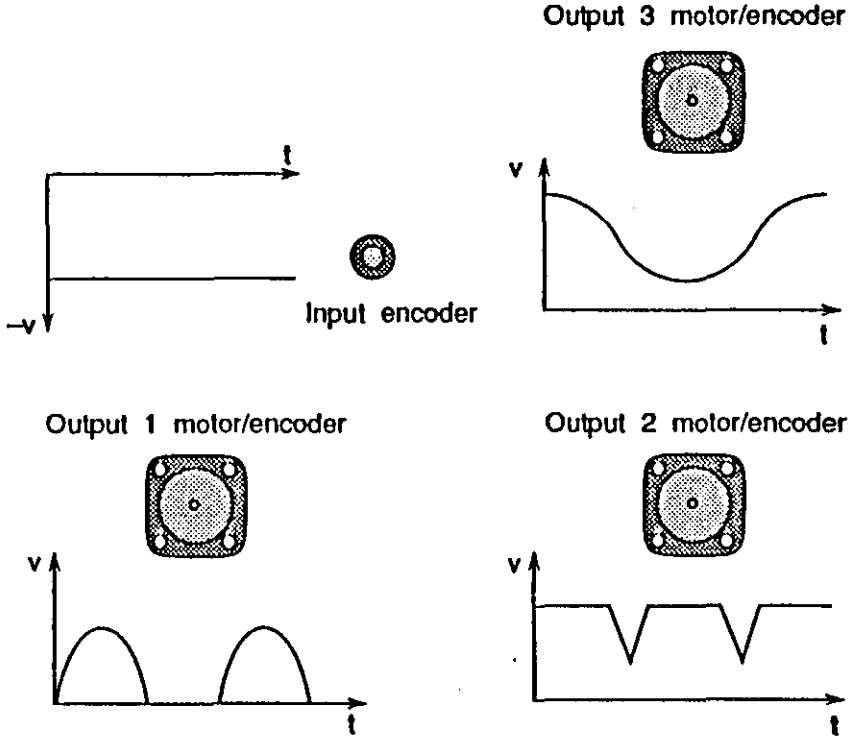
Software clutches, gearboxes and cams are electric motor driven replacements for conventional mechanical transmission systems. They can be regarded in much the same manner as relay ladder logic acts as a replacement for conventional hardwired logic. Potential advantages include improved operational reliability, flexibility, precision, and design simplicity [69].

A programmable transmission system defines a series of mappings to interrelate the drive motor positions in the system. See figure 4.13. The high level de-

scription of these position relationships may be in tabular or graphical form and is the subject of current research by the Modular Systems Group at Loughborough.



Complete Traditional Mechanical Transmission System



Programmable Transmission System with Servo Motors

Figure 4.13. Concept of a programmable transmission system. Source: Quin Systems.

4.5.5 Task Structure Description

Model based methods for defining the task structures for machine operations are gradually being introduced. Function charts are emerging as promising graphical environments which can support other methods of programming from within their structure.

Continuous Function Charts (CFCs) provide graphical representations of closed loop control problems. Sequential Function Charts (SFCs) similarly provide an appropriate model for sequential control [70]. See figures 4.14 and 4.15. Brandl [71] of Texas Instruments suggests that a future control system design environment should combine sequence control described graphically by SFCs, with continuous control described graphically by CFCs.

CFCs and SFCs provide a means of organising program segmentation and flow. Code segments must still be written in appropriate language(s). They provide a choice of languages which in turn may use an internal model if appropriate e.g. ladder logic, state tables, motion definition languages for programmable transmissions or other machine specific languages. Where control algorithms or structured data are required a general purpose textural language e.g. C, Pascal, Fortran may be used. Experts may therefore use the language in which they are most proficient provided that a well defined interface exists between these different blocks of code.

Initiated in France in the mid-1970s a joint industry and government effort undertook to define a new approach to logic states, steps and possible graphic representations of these ideas. Their work led to a French standard [72] for SFCs called Grafcet and there is hope for international adoption of this or a similar standard [70]. Other SFC programming languages are however emerging; for example a similar but different German national standard has been implemented by Siemens [73]. In the US no such programming language standard currently exists although Maxitron and Allen-Bradley are known to support function chart methods and Texas instruments have recently developed a similar programming environment called Applications Productivity Tool (APT) [74]. NEMA intends to adopt IEC standards for SFCs and CFCs which can be presented to ANSI for adoption as a national standard in the US.

In concept a function chart environment should allow designers to use an appropriate description language for each separate piece of code, check the coherence of data as the design progresses, and automatically generate a final run-time program from this description.

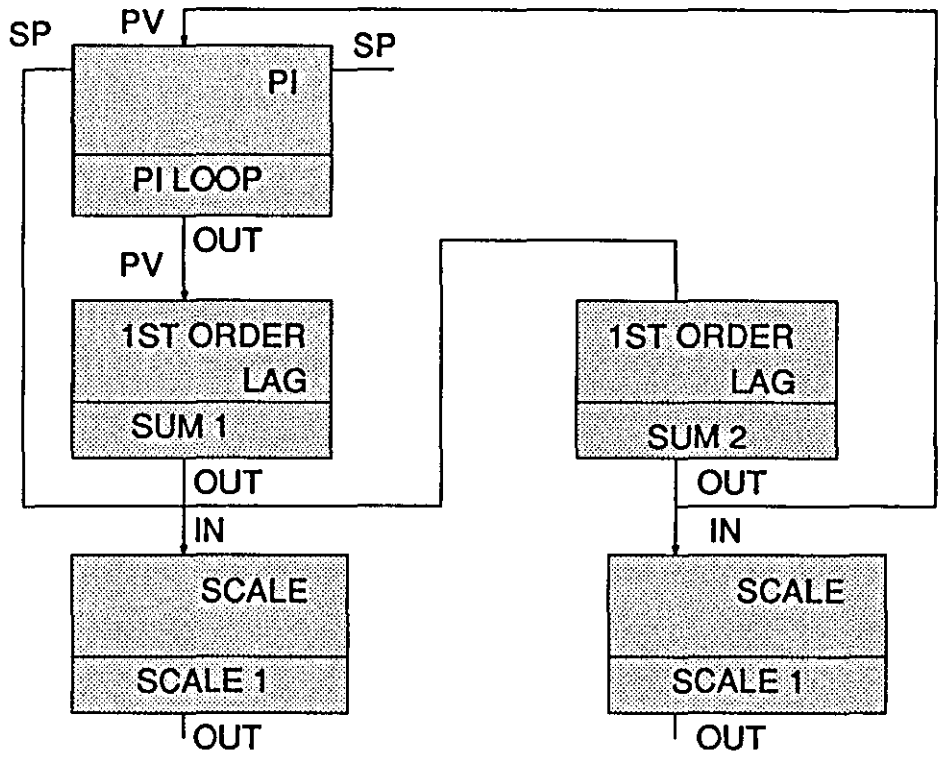


Figure 4.14. Continuous function chart. Source: Brandl.

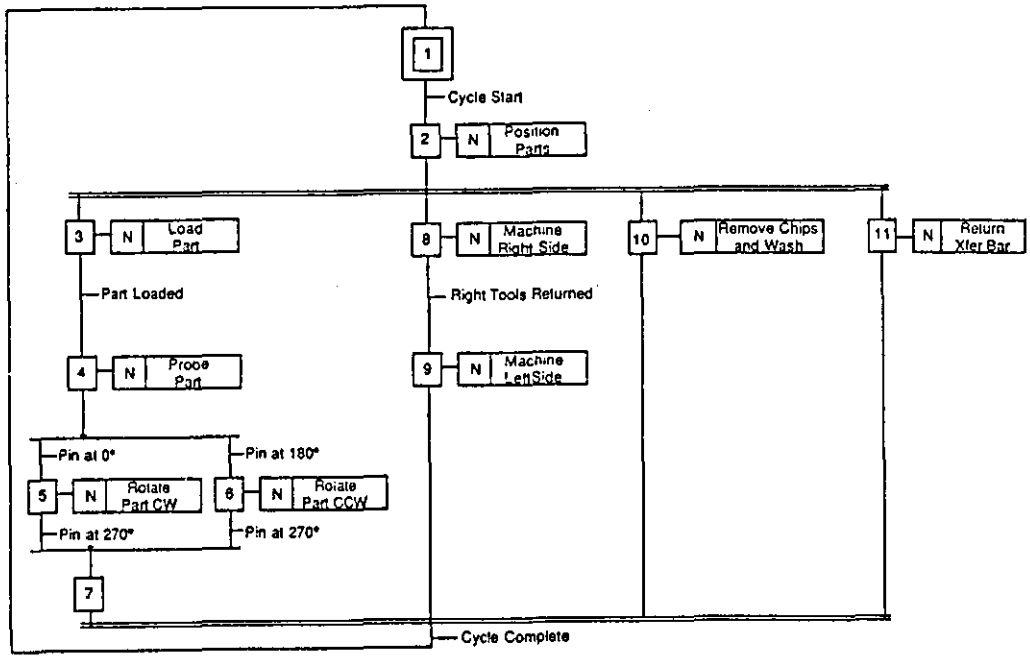


Figure 4.15. Sequential function chart. Source: Benedick.

4.5.6 Grafcet Charts

In order to explain the concept of function charts more fully this section describes the basic elements of Grafcet programming.

There are only three basic elements in Grafcet charts: Steps, Transitions and Flowlines. See figure 4.16.

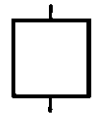
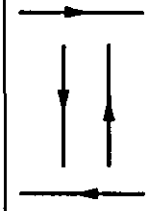
Name	Initial Step	Step	Trans-ition	Flow-line	Double Flow-line	Macro Step
Graphic Element						

Figure 4.16. Grafcet language elements. Source: Gonzalez.

A step is used to define an action. The step is numbered and its principal function may be described via mnemonics on the programming screen.

The transition is identified by the step number that leads to it and the step number that follows it and is also described by a mnemonic on the screen. Flowlines tie steps and translations together to form complete charts. See figure 4.17.

During design a particular method of programming is associated with each step and the concept of "zoom" allows the nesting of code in the graphics environment [70].

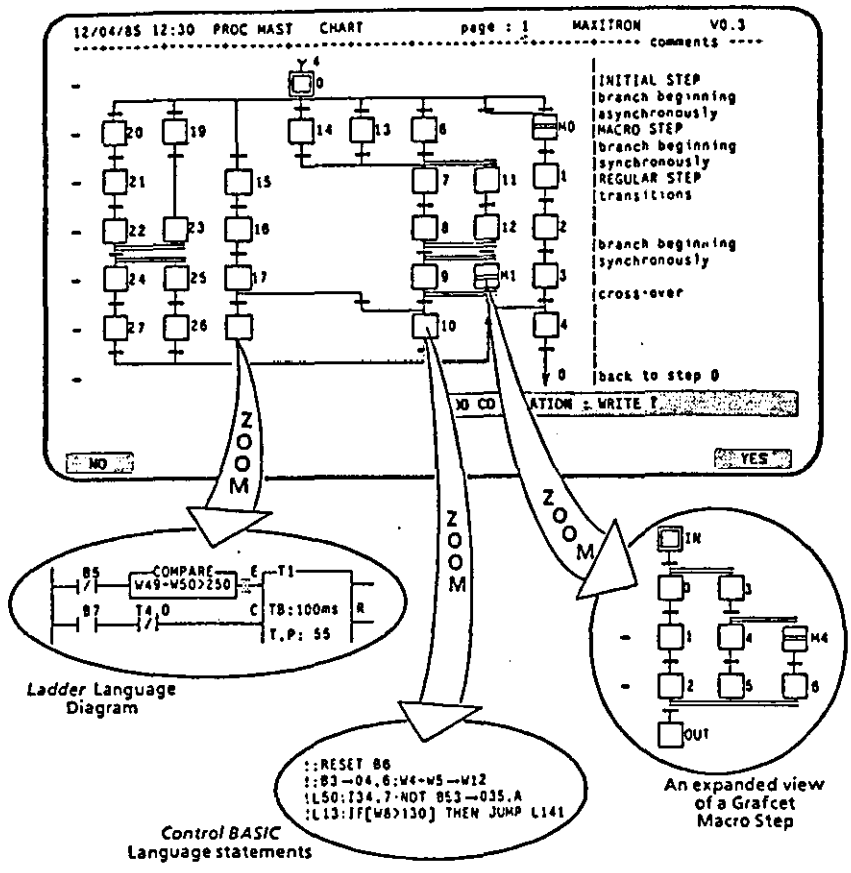


Figure 4.17. The Grafcet programming environment supports a hierarchy of languages. Source: Gonzalez.

4.5.7 Future Trends in Programming

If the programming of industrial control systems follows the trends in the business computing field, better problem description languages with more appropriate and more highly refined internal models will eventually replace the older languages. The use of a standard base level operating system and system programming language also appear highly desirable goals.

Function chart programming environments provide the opportunity for code reuse which currently very rarely occurs. A segmented structure approach gives a user the ability to define and store routines that are commonly used in their company's applications. The resulting applications library becomes a valuable asset for future development. There is also the potential for a third party packaged software market as seen in business computing. Third party software might include specialised control algorithms, statistical routines, management information collection, diagnostics and external intelligent device interfaces to motion controllers, vision

systems etc. Such an approach might also allow multiple parties to collaborate on the same program. By using a graphic, process-defining, top down approach, a common "meeting ground" can potentially be created for the process or machine engineer and the control engineer; for the client and the systems house; and for the user and the original equipment manufacturer.

There is a pressing need for the widespread adoption of a high level problem description languages that permits the simple specification of any kind of process activity, that is easy to modify and self documenting. Lytle [62] maintains however that the current IEC attempts to standardise programming languages are so broad there are still no practical machine level implementation standards.

4.6 CONCLUSIONS

The acceptance of problem orientated approaches for real-time control systems has been much slower than in the case of business systems. Typical reasons given for this disparity include [75], [76]:

- Real-time problems are usually more demanding.
- Applications are much more diverse.
- There is generally lower capital investment in new technology in what is considered a very traditional customer driven market sector.
- Users are often unaware of the potential of new methods.

Recently, formal problem formulation techniques have been used for the development of real-time software and graphical problem description methods have been proposed. The application of these methods to industrial control has however been largely uncoordinated and unstandardised.

Chapter 4: References

1 Ward, T.W. and Mellor, S.J., "Structured Development for Real-Time systems", Vol. 1, Chapter 1, Yourdon Press, Prentice-Hall, 1985, pp 7-8.

2 Lui, J., "The CAM-I Advanced Factory Management System", Proceedings on Factory Standards Model Conference, National Bureau of Standards, November 1985.

3 Flatau, U., "Digital's CIM Architecture", Revision 1.1, Digital Equipment Corporation, April 1986.

4 Klevers, T. "The European approach to an `Open System Architecture' for CIM: CIMOSA", Proc. 5th CIM Europe Conf., May 1989, pp 109-120.

5 Anon., "Minutes of CIM Reference Model Committee", Int., Purdue Workshop on Industrial Computer Systems, September 1986, Purdue Laboratory for Applied Industrial Control, West Lafayette, USA, pp 13-46.

6 Lui, D., "Factory Management Architecture and Information Infrastructure", Proceedings on Factory Standards Model Conference, National Bureau of Standards, November 1985.

7 Mayer, R.J., "The ICAM MCS Architecture and its Implementation", Proceedings on Factory Standards Model Conference, National Bureau of Standards, November 1986.

8 Anon., "Reference Models for Manufacturing Standards", International Standards Organisation Report ISO TC184/SC5/WG1 N51, Version 1.1, September 1986.

9 Albus, J.S. et al., "An Architecture for Real-Time Sensory-Interactive Control of Robots in a Manufacturing Facility", Centre for Manufacturing Engineering, National Bureau of Standards, Washington, D.C., 1984.

10 Parunak, H.V., "Viewing the Factory as a Computer: A Cognitive Approach to Material Handling", Industrial Technology Institute, Ann Arbor, MI, USA, July 1987.

11 Parunak, H.V. et al., "An Architecture for Heuristic Factory Control", Proceedings of the 1986 American Control Conference, March 1986.

12 Naylor, A., "The Manufacturing Game: A Formal Approach to Manufacturing software", IEE Transactions on Systems, Manufacturing and Cybernetics, Vol. SMC-16:3, June 1986.

13 Anon., "Minutes of CIM Reference Model Committee", Int., Purdue Workshop on Industrial Computer Systems, September 1986, Purdue Laboratory for Applied Industrial Control, West Lafayette, USA, pp 63-82.

14 Doumeingts, G., "Integration of Decisional and Physical Systems in the Design of Manufacturing systems", Proc. 25th IEEE Conf. on Decision and Control, December 1986, Athens, Greece, Vol. 3, pp 2212-2217.

15 Suda, H., "Future Factory System Formulated in Japan", Japanese Journal of Advanced Automation Technology, September 1989, Vol. 1, pp 67-76.

16 Weston, R.H. et al., "The Need for a Generic Framework for Systems Integration", in "Advanced Information Technologies for Industrial Material Flow Systems", Ed. Nof S Y and Moodie C L, NATO ASI Series F53, pp 279-309, Springer-Verlag, 1989.

17 Pan, Y.C., "A Framework for Knowledge-Based Computer-Integrated Manufacturing", IEEE Transactions on Semiconductor Manufacturing, Vol. 2, No. 2, May 1989, pp 33-46.

18 Parunak, H.V. et al. "A Synthesis of Factory Reference Models", Industrial Technology Institute, Ann Arbor, Michigan, USA, September 1987.

19 Parunak, H.V.D. et al., "Fractal Actors for Distributed Manufacturing Control", Proc. of 2nd IEEE Conference on Artificial Intelligence Applications, 1985, pp 653-660.

20 Albus, J. E. et al., "Theory and Practise of Hierarchical Control", Proc. 23rd IEEE Computer Soc. Int. Conference, October 1981, pp 497-505.

21 Haynes, L.S. and Wavering, A.J., "Real-Time Control Systems Software, Some Problems and an Approach", Real-Time Control Group, National Bureau of Standards, 1986.

22 Anon, "The CIMPHONY Pilot Project Technical Papers", Philips, CFT-CAM Centre, 5600 MD Eindhoven, 1987.

23 Hormann A. et al, "A Concept for an Intelligent and Fault-Tolerant Robot System", Journal of Intelligent and Robotic Systems, Vol. 1, Kluwer Publishing, 1988.

24 Lent, B., "Dataflow Architecture for Machine Control", Research Studies Press Ltd., Wiley, 1989.

25 Paul, R.P. et al, "A Robust, Distributed Sensor and Actuation Robot Control System", 3rd Int. Symp. on Robotic Research, MIT Press, 1986.

26 Shin, K.G., "Intertask Communication in an Integrated Multirobot System", IEEE Journal of Robotics and Automation, Vol. RA-3, No. 2, April 1987, pp 90-100.

27 Lehtinen, M. "Control of a Mechatronic Machine", Tampere University of Technology, Institute of Hydraulics and Automation, 33101 Tampere, Finland, June 1989.

28 Schmitz, D. et al. "The CMU Reconfigurable Modular Manipulator System", Technical Report CMU-RI-TR-88-7, Robotics Institute, Carnegie Mellon University, Pittsburgh, USA, May 1988.

29 Shin, K.G., "A Modular Architecture for an Integrated Multi-Robot System", Centre for Research and Integrated Manufacturing (CRIM), The University of Michigan, Ann Arbor, Tech. Report RSD-TR-10-84, July 1984.

30 Anon., "ROBOTWORLD Systems Controller Group", Document #PS-RB-21, Automatix Inc., 755 Middlesex Turnpike, Billerica, MA 01821.

31 Ashton, M., "A flexible assembly system controller", Proc. 2nd Int. Conf. Machine Control Systems, May 1987, pp 165-174.

32 Taylor, R.H., "A General-Purpose Control Architecture for Programmable Automation Research", 3rd Int. Symp. on Robotic Research, MIT Press, 1986.

33 Koren, J. et al, "A Configurable System for Automation Programming and Control", Manufacturing Research Department, IBM, T.J. Watson Research Centre, Yorktown Heights, New York, 10598 USA, February 1986.

34 Bharu, B et al, "CAOS: A hierarchical robot control system", Int. Conf. Robotics and Automation, Vol 3, 1987, pp 1603-1608.

35 Silberschatz, A., "Advanced Real-Time Languages for Distributed Industrial Process Control", Industrial Computing, February 1984, pp 37-46.

36 Kopetz, H. et al., "Distributed Fault-Tolerant Real-Time Systems: The Mars Approach", IEEE MICRO, February 1989, pp 25-40.

37 Kosa, C., "Scheduling of Hard Real-Time Tasks in the Fault-Tolerant Distributed Real-Time System MARS", Proc. 4th IEEE Workshop Real-Time Operating Systems, July 1987, pp 31-36.

38 Holding, D.J., "The Design of Distributed, Software Fault Tolerant, Real-Time Systems Incorporating Decision Mechanisms", Microprocessing and Microprogramming 24, Vol 24, August 1988, pp 801-806.

39 Gaglianello, R.D., "A Distributed Computing Environment for Robotics", AT&T Bell Laboratories, Holmdel, NJ. 07733 USA, February 1986.

40 Sakamura, K, ed., "TRON Project 1988: Open Architecture Computer Systems," Proc. 5th TRON Project Symp., Springer-Verlag, Tokyo, 1988.

41 Orr, K. et al, "Methodology: The Experts Speak", Byte, April 1989, pp 221-233.

42 Glad, A.S., "Software Engineering Guide for Real-Time Control Systems Development", Sixth Control Engineering Conference, Rosemont, IL, May 1987, pp 188-199.

43 Ward, T.W. and Mellor, S.J., "Structured Development for Real-Time systems", Vol. 1, Chapter 5 - The Modelling Process, Yourdon Press, Prentice-Hall, 1985.

44 Frost, S., "Making the best method mix", CASE in the '90s, Microsystem Design, February 1990.

45 Hughes, T. "A model for the future", Microsystem Design, February 1990.

46 Ward, T.W. and Mellor, S.J., "Structured Development for Real-Time systems", Vol. 1, Chapter 4 - Modeling Heuristics, Yourdon Press, Prentice-Hall, 1985.

47 McMenamin and J. Palmer, "Essential Systems Analysis", Yourdon Press, New York, 1984.

48 Yourdon, E., "The Yourdon Approach", Byte, April 1989, pp 227-230.

49 Simpson, H. R., "MASCOT- A modular approach to software construction, operation and test", RRE Technical Note No. 778, 1975.

50 Simpson, H. R., "MASCOT 3", IEE Colloquium, Paper No. 3, IEE, Savoy Place, London, 1984.

51 Gibson, M.L., "The CASE Philosophy", Byte, April 1989, pp 209-218.

52 Banks, D., "Systems thinking, systems practice", Microsystems Design, February 1990, pp 16-17.

53 Anon., "Analyst/Designer Toolkit", Technical Note, Yourdon International Limited, 15-17 Ridgemount Street London.

54 Anon., "HOOD", Microsystem Design, February 1989, pp 12-13.

55 Arai, T. et al, "Standardization of Robot Software in Japan", 15th ISIR, pp 995-1001.

56 Dybeck, M. "Control Requirements for the Process Industry", Sixth Control Engineering Conference, Rosemont, IL, May 1987, pp 200-205.

57 Volz, R.A., "Report of the Robot Programming Language Working Group: NATO Workshop on Robot Programming Languages", IEEE Journal of Robotics and Automation, Vol. 4, No. 1, February 1988, pp 86-90.

58 Anon., "IEC Draft Standard for Programmable Controllers", Part 3: Programming Languages, 65A (Secretariat 67), January 1987.

59 Ambler, A.P., "Languages for Programming Robots", NATO ASI series, Vol. F11, Robotics and Artificial Intelligence, Edited by M. Brady et al., Springer-Verlag, 1984, pp 219-227.

60 Gruver, W.A., Soroka, B.I., Craig, J.J. and Turner, T.L., "Evaluation of Commercially Available Robot Programming Languages", Proc. of 13th ISIR Chicago, 1983, Section 12, pp 58-68.

61 Richardson, J.H., "Programming Languages Vie for Control", I&CS, December 1986, p37.

62 Lytle, V.A., "A Survey of PLC Language Features", Seventh Control Eng. Conf., May 1988, Section XXIII, pp 14-20.

63 Rock, S.T., "Developing Robot Programming Languages Using an Existing Language as a Base - A Viewpoint", Robotica, Vol. 7, pp 71-77, 1989.

64 Considine, D.M. (ed), "Standard Handbook of Industrial Automation", Chapter 3 Control Systems, Chapman and Hall, 1986, pp 133-235.

65 Notte, A.J., "Multitasking Capability Simplifies Process Control Design", Seventh Control Eng. Conf., May 1988, Section VIII, pp 15-29.

66 Benedick, D.R., "Structured Programming to Develop Machine Control", Seventh Control Eng. Conf., May 1988, pp 21-26.

67 Taylor, R.H., "A General-Purpose Control Architecture for Programmable Automation Research", Manufacturing Research Department, IBM, Yorktown Heights, New York, USA, 1986.

68 Pelland, R., "The State Logic Model for Programmable Control Languages", Seventh Control Eng. Conf., May 1988, Section XXIII, pp 1-13.

69 Sinha, P.K., "Transmitting Power Intelligently", Professional Engineering, Mech. Eng. Publications, April 1990, pp 52-53.

70 Gonzalez, M.R., "Blend Multiple Languages in a Programmable Controller using Grafcet", Sixth Control Engineering Conference, Rosemont, IL, May 1987, pp 458-469.

71 Brandl, D.L. and Clark, W. "Directions in Control System Programming", Seventh Control Eng. Conf., May 1988, Section XXIII, pp 6-13.

72 Anon "Diagrams, Charts, Tables: Control System Function Chart 'Grafcet' for the Description of Logic Control Systems", Union Technique de Electricite (UTE), French Standard NF C03-190, Paris, June 1982.

73 Loyd, M. "Graphical Function Chart Programming for Programmable Controllers", Control Engineering, October 1985.

74 Williams, D. "PLCs Beat Relays in the Cost Race", Professional Engineering, Mech. Eng. Publications, April 1990, pp 40-41.

75 Blue, S, "Why hang on to ladder logic?", Industrial Technology, Hanover Publications, UK, February 1989, pp 34-35.

76 Muir K., "Stating the CASE for Industrial Automation", Drives and Controls, June 1990, pp 22-24.

5
A Methodology for Machine Control

5.1 INTRODUCTION

The remainder of this thesis relates to an evolving methodology termed UMC (Universal Machine Control), to facilitate the design and implementation of computer controlled manufacturing machines in a generalised manner. The objectives of the work are to offer not only more cost effective initial control system installations, but moreover to provide a consistent method for both system configuration and the integration of additional functions as new requirements evolve. It is therefore seen as essential to create a methodology which allows the controller to develop over a period of time and to adapt to evolving requirements and technologies.

This chapter outlines the thinking of the author with regard to the overall aims, concepts and potential of the UMC methodology. Research activity over a four year period carried out principally by Mr Alan Booth and the author has centred on a proof of concept implementation of this methodology which is described in chapters seven and eight. These studies were undertaken under the guidance of Professor R.H. Weston.

In a broader context the methodology described here can be naturally extended to support a fresh approach to total machine design. This philosophy aims to integrate mechanism and controller design and to allow the modelling of their behaviour in a consistent manner in order to determine the optimum combination of the two. Such extensions to the methodology are the topic of on going research by the Modular Systems Group at Loughborough [1].

5.2 REQUIREMENTS FOR A GENERALISED APPROACH TO MACHINE CONTROL

As discussed in chapter four, manufacturing control systems are typically distributed systems that can be viewed and intuitively modelled as a hierarchy of in-

teracting parts under separate control, cooperating to realise the function of the total system. Machines must logically be viewed as a part of such a hierarchy.

A machine controller must respond to and serve both business objectives and real-time control objectives during its life cycle. Section 2.5 considered the necessary attributes which a machine control system should embody in order to effectively support an integrated manufacturing environment which is responsive to change. These attributes are summarised in table 5.1. A machine control system fulfilling these attributes would typically be complex and thus difficult to understand and specify as one integrated item. A model of such a system would however enable its representation as a number of co-operating parts that could be specified more easily.

As discussed in chapters two and three, and considered in some detail by Thatcher [2], there are many common parts in different machine control systems. However the wide variations in specific machine requirements and the continual improvements in enabling technology make the adoption of a standardised approach to machine control a difficult objective. A system model is therefore required to cater for real-time machines in the widest context. There is the need for a structure which can support the required functionality from reusable parts. To answer this need the UMC methodology has been evolved at Loughborough.

	CONTROLLER ATTRIBUTES
LOCAL MACHINE/PROCESS RELATED REQUIREMENTS:	REAL-TIME CONTROL CAPABILITIES TO MAXIMISE MACHINE PERFORMANCE
	MAXIMISE CONTROL SYSTEM FUNCTIONAL VISIBILITY
	FUNCTIONAL FLEXIBILITY
	PROBLEM ORIENTED OPERATOR INTERFACE
GLOBAL MANUFACTURING RELATED REQUIREMENTS:	MAXIMISE INFORMATION VISIBILITY
DESIGN REQUIREMENTS:	EFFECTIVE ARCHITECTURE AND SYSTEM BUILDING TOOLS

Table 5.1. Controller attributes.

5.3 MODELLING TERMINOLOGY

Discussion of control systems and their modelling is often hampered by the lack of standard terminology. In an effort to encourage consistency and avoid duplication existing terminology has been adopted wherever it is considered appropriate from the CIMOSA [3], ISO [4], DEC/Philips [5], AUTOMAIL [6] and HOOD [7] modelling studies. However the existing terminology used by these bodies is not considered consistent or completely appropriate to machine control systems. This section therefore defines the terms used in this thesis for describing the modelling and architecture of machine control systems.

Model

When designing any system there is always a *Model*, explicitly defined on paper or implicitly in the mind of a developer as a starting point for the development process.

Reference Model

If the system model is well known and used by many people who refer to the model when they discuss developments or standardisations that model is a *Reference Model.* A reference model thus provides a common framework for discussion. A reference model shows:

- a structure saying which parts may cooperate
- the tasks of the parts, which determine what should be done by a system part, so that its contribution to the complete system function is realised.

Architecture

Architecture is the art of designing and realising a complex system. Architecture is a powerful term, in being evocative of structure and style. The *structure* and *interconnection* of building blocks is the basis of architecture.

A strong motive for enforcing a consistent architecture is to enable *design reuse.* This is based on the reuse of architectural building blocks and frameworks.

Reference Architecture

A *Reference Architecture* is developed based on a Reference Model. It describes the function of system parts, as opposed to how they operate internally.

A Reference Architecture is composed of a set of generic guide-lines, constraints which provide a framework together with building blocks of defined generic types.

It is important to put the consideration of architecture in the context of the control application being considered. For the reuse of control system building blocks to be worthwhile there must be common characteristics in the range of applications to be addressed by system users. Clearly there are big differences across machine control applications but there are also many major commonalities. These commonalities can be addressed by a Reference Architecture.

Partial Models	*Partial Models* contain sets of building blocks of the defined generic types becoming increasingly application and implementation specific. These building blocks conform to the Reference Architecture but also meet the more specific requirements of a given category of machine function and/or implementation method.
Particular Model	A *Particular Model* is unique to a specific machine and is composed of a specified set of building blocks for that application.
Design	The term *Design* generally denotes the activity of or result of creating something. A design is the description of how something is put together internally. In this context it is taken to refer to how an architectural building block operates internally.
Aggregation	*Aggregation* is the collection of parts into one body. In this context it is taken to refer to how architectural building blocks are combined to form a run time control system.
System Build	*System Build* is the implementation of a Particular Model in hardware and software.
View	A *View* is a *distinct* aspect or dimension of a model architecture which may be considered independently (although it may be coupled to other views).

5.4 MODELLING PHILOSOPHY

A major reason for the use of system models of any type is the management of complexity. The human mind is capable of understanding a complex problem as long as it is presented in manageable parts. Hence the need for methodologies and tools to make the structuring and manipulation of systems manageable. Modular decomposition is one of the most commonly proposed systematic approaches to achieving problem simplification [8, 9, 10].

The application of modular decomposition to industrial machines is depicted in figure 5.1. The principle behind the creation of a *Reference Architecture* is that specific manufacturing machines can be considered as a sub-set of the general manufacturing machine. At the lowest level of decomposition of the general machine is assumed to be a set of modular building blocks which can then be combined in appropriate hierarchies to create any required machine and associated controller.

This concept has been applied at LUT in the UMC approach to machine control which is summarised in the schematic shown in figure 5.2. The functionality of any *particular* machine controller can be composed from *reusable* building blocks from a larger family of modules for the *general* machine controller.

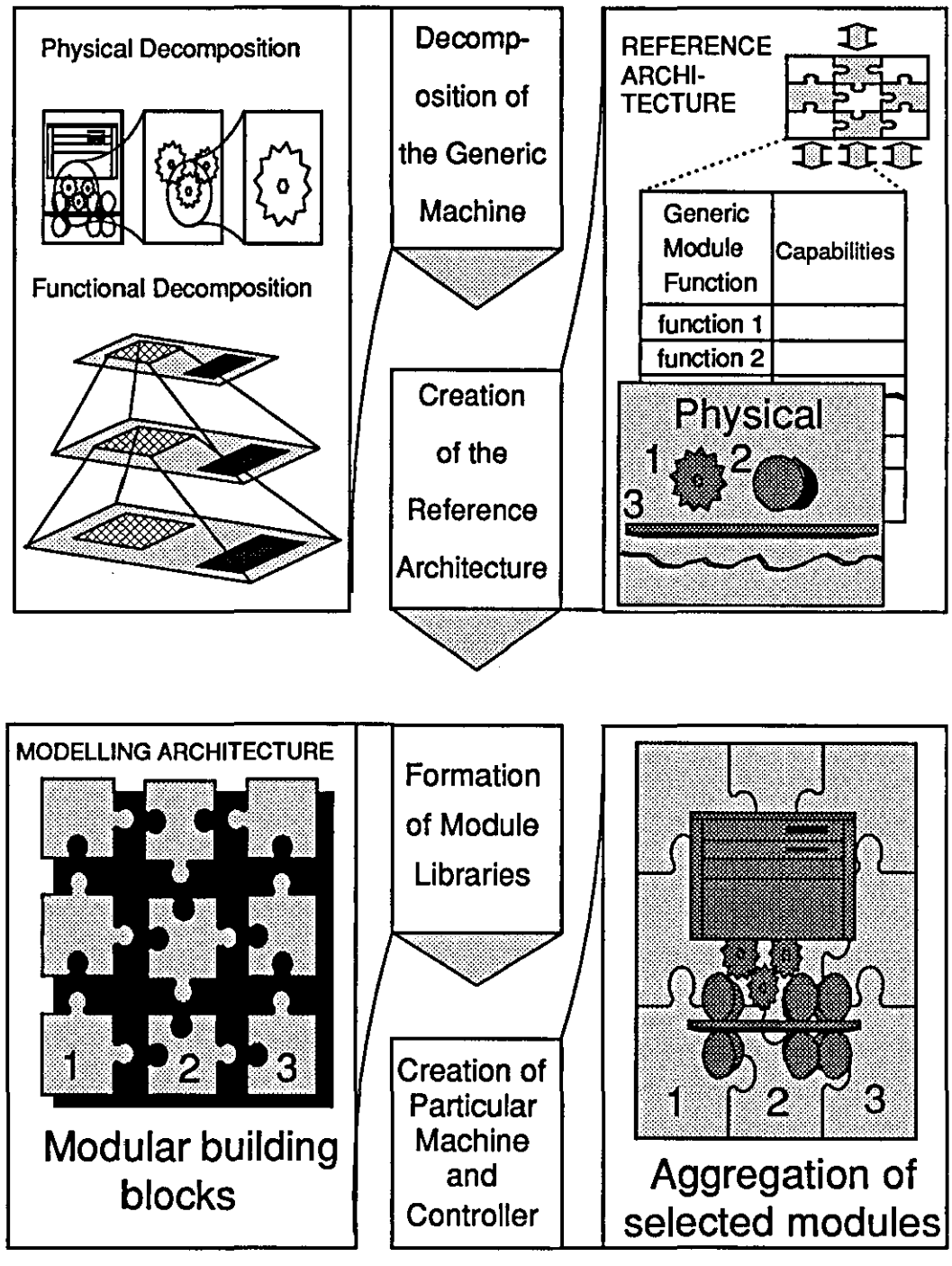


Figure 5.1. Concepts behind the UMC modelling methodology.

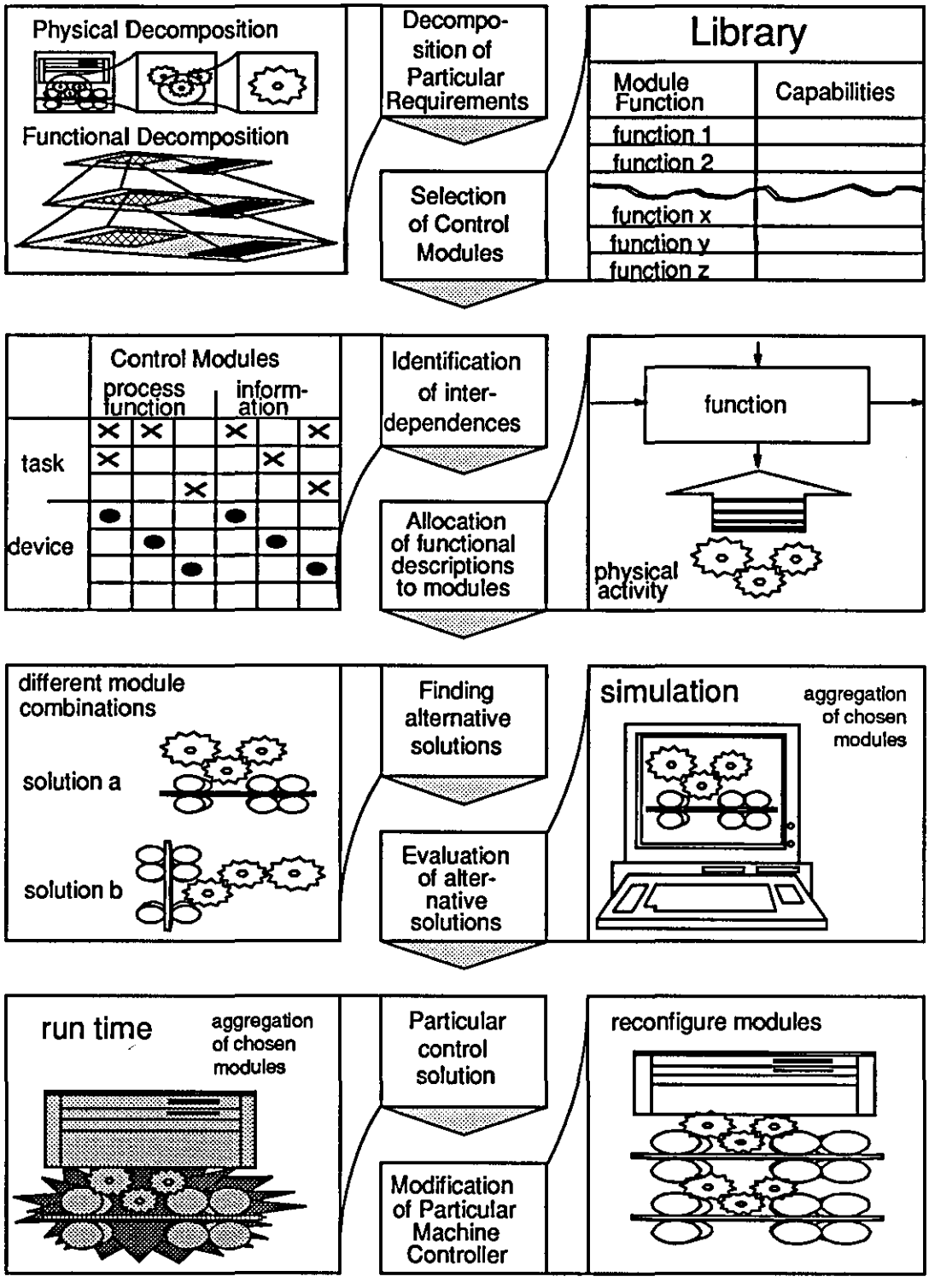


Figure 5.2. The UMC approach to the creation of particular machine controllers.

5.5 UMC REFERENCE MODEL

5.5.1 Introduction

The basis of any reference architecture must be a model providing a common framework for discussion. The UMC Reference Model is very simple in both concept and structure.

5.5.2 Reference Model Levels

The UMC Reference Model explicitly defines only three hierarchical levels *Machine*, *Task* and *Handler*. See figure 5.3.

5.5.2.1 Machine Level

Any required machine is "driven" (i.e. initiated, referenced, modified and removed) by the Machine Level which provides a single processing and information interface to the level above in the manufacturing systems environment.

5.5.2.2 Task Level

Since the design of particular machines (and hence Particular Models) is seen as very application dependent, the application task structure is deliberately *not* explicitly defined. The Task Level therefore supports the concept of multiple tasks which can be arranged both hierarchically and heterarchically in a user defined manner as determined by the requirements of individual applications. Machines may thus have a complex internal structure of control tasks to support the specific device control required in a very flexible manner.

5.5.2.3 Handler Level

The *Handler Level* provides isolation between the internal unified UMC system representation and the external device controllers, thus restricting the impact of changes. A handler provides a means of translating device specific information flows into a standard form for each class of machine I/O device. This requires defined types of handler to cater for generic classes of I/O device.

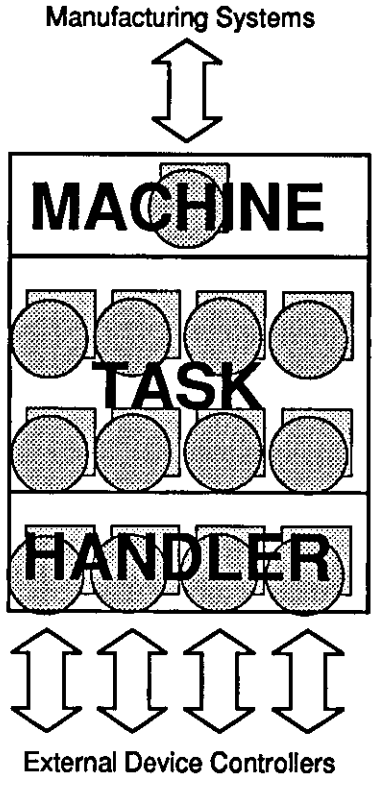


Figure 5.3. Hierarchical levels in the UMC Reference Model.

5.5.3 Reference Model Views

Two distinct views can be identified within the UMC Reference Model. These views enable the operation of the system to be separated from its data requirements aiding reusability. See figure 5.4

The *process view* is a representation of the operation of the system in terms of a set of concurrent activities. This view is composed of the machine, task and handler processes.

The *information* view is composed of a set of information modules containing structured data. These modules each consist of structures and substructures composed of individual data items. All functional processes have potential access to this global information structure. The information view describes the underlying information requirements of the system. It is composed of machine, task and handler information modules.

Both the process and information views of the model are initiated, indexed and referenced from the machine level. This is to enable the structure and content of

particular control system models to be maintained at all times during development and at run time.

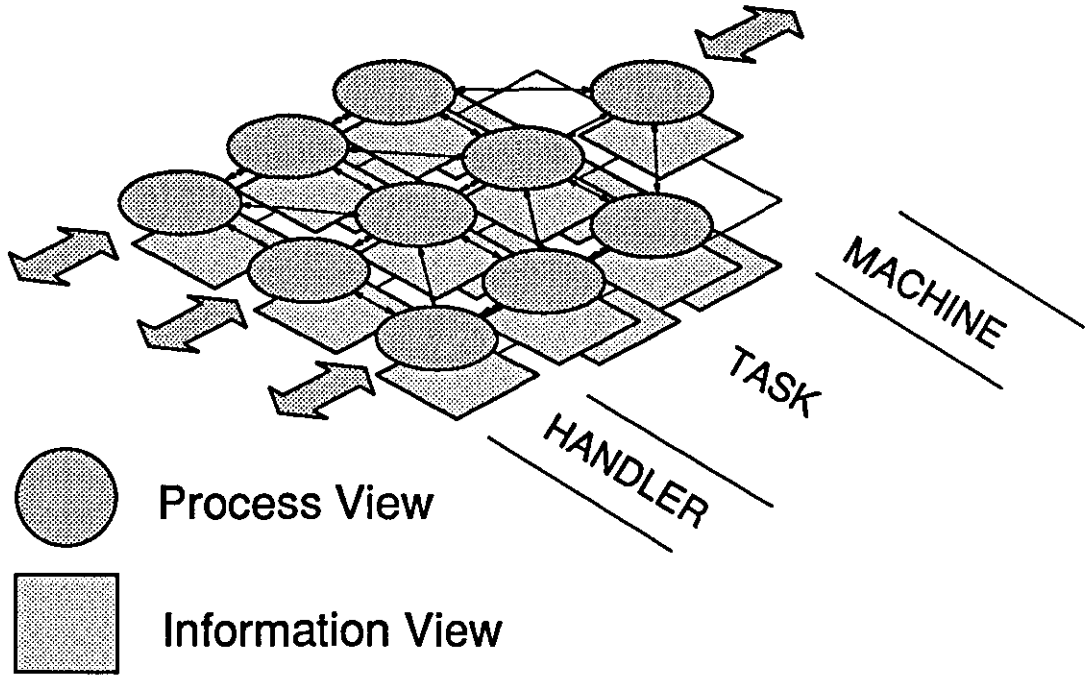


Figure 5.4. Reference Model process and information views.

5.6 UMC REFERENCE ARCHITECTURE

5.6.1 Introduction

The Reference Architecture takes the Reference Model and adds to it the definition of the building blocks and the allowable I/O between them. It is important to note that the Reference Architecture describes what the system building blocks do as opposed to how they operate internally. See figure 5.5. The Reference Architecture thus defines the content of the information and processing views.

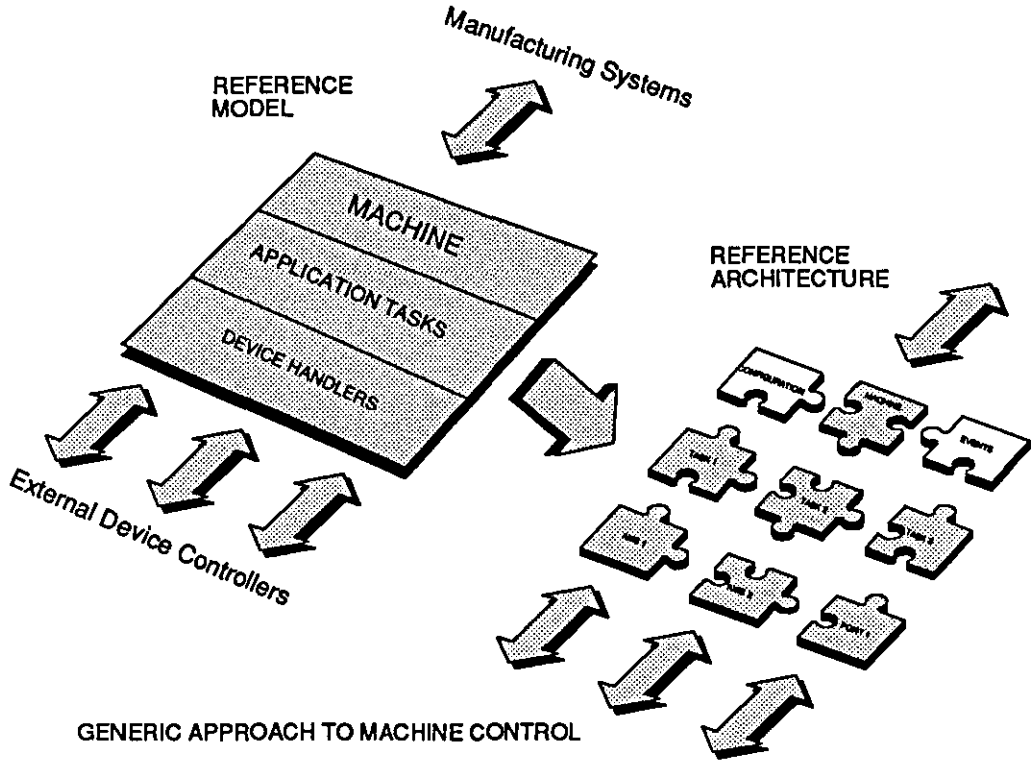


Figure 5.5. Derivation of the Reference Architecture.

5.6.2 Reference Architecture Building Blocks

UMC Modules

At the lowest level in the decomposition of the UMC Model are *Modules*. The Reference Architecture is composed of UMC Modules. These are identifiable items of software. UMC modules may contain activity programs (process code) or information (global structured data). See figure 5.6.

UMC Component

The term *Component* is used to describe an associated set of UMC Modules at Machine, Task or Handler Level which embodies the required behaviour for *that* level.

UMC components are composed of program modules (for processes) and information modules (containing structured data) together with defined interface mechanisms. The number of modules and their types is dependent on the required behaviour of the component. All machine components (and in turn all modules) are referenced from the machine configuration module which holds the configuration information for a Particular Machine. See figure 5.7.

UMC Machine	The top level component in the UMC Architecture, the *Machine* component is composed of a machine configuration module and associated program modules. The machine information module describes the Particular Model of a given machine at run time. Machine processes map the particular machine control model onto its target hardware, allow dynamic modification of its operation or structure and enable its removal. See figure 5.8.
UMC Tasks	*Tasks* are application specific components in the UMC Reference Architecture. They consist of user defined program and associated information modules. Complete applications can be divided into concurrent tasks with defined synchronisation and information access mechanisms. See figure 5.9.
UMC Handlers	*Handlers* provide a uniform software view of I/O devices to UMC tasks. Each Handler integrates a specific external I/O device into the system, communicating with it via an implementation specific device driver. The UMC Reference Architecture defines specific types of handler for given types of I/O device. See figure 5.10.

5.6.3 Reference Architecture Communication

The permitted communication paths are illustrated schematically in figure 5.11.

Machine To Above	Unified manner with defined instruction sets dependent on the chosen model for the manufacturing systems environment.
Tasks To Global Information View	Consistent information access control mechanisms to any global data within the run time system.
Task To Task	Defined information transfer and synchronisation mechanisms. Design of Particular Model effects specific use.
Task To Handler	Defined in a unified manner using generic instruction sets. Communication mechanism is device independent. Note particular devices many not be able to support all the capabilities of a generic device type.
Handler To Device	Device specific communication formats and instruction sets.

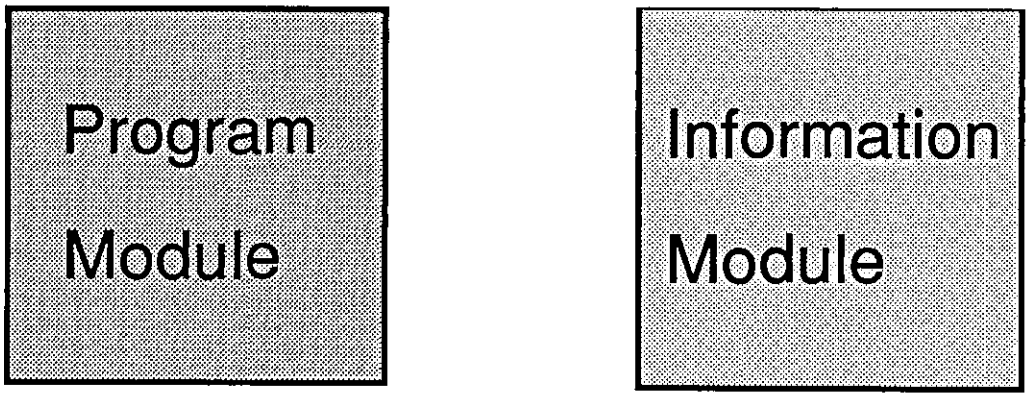


Figure 5.6. Symbolic representation of UMC modules.

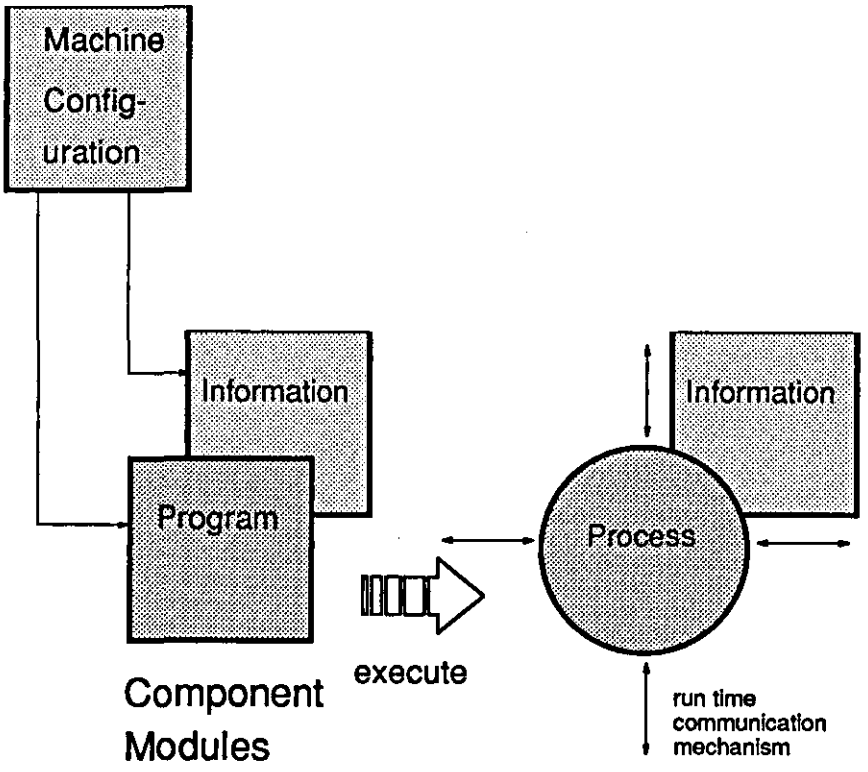


Figure 5.7. Symbolic representation of UMC components both during storage and at run time.

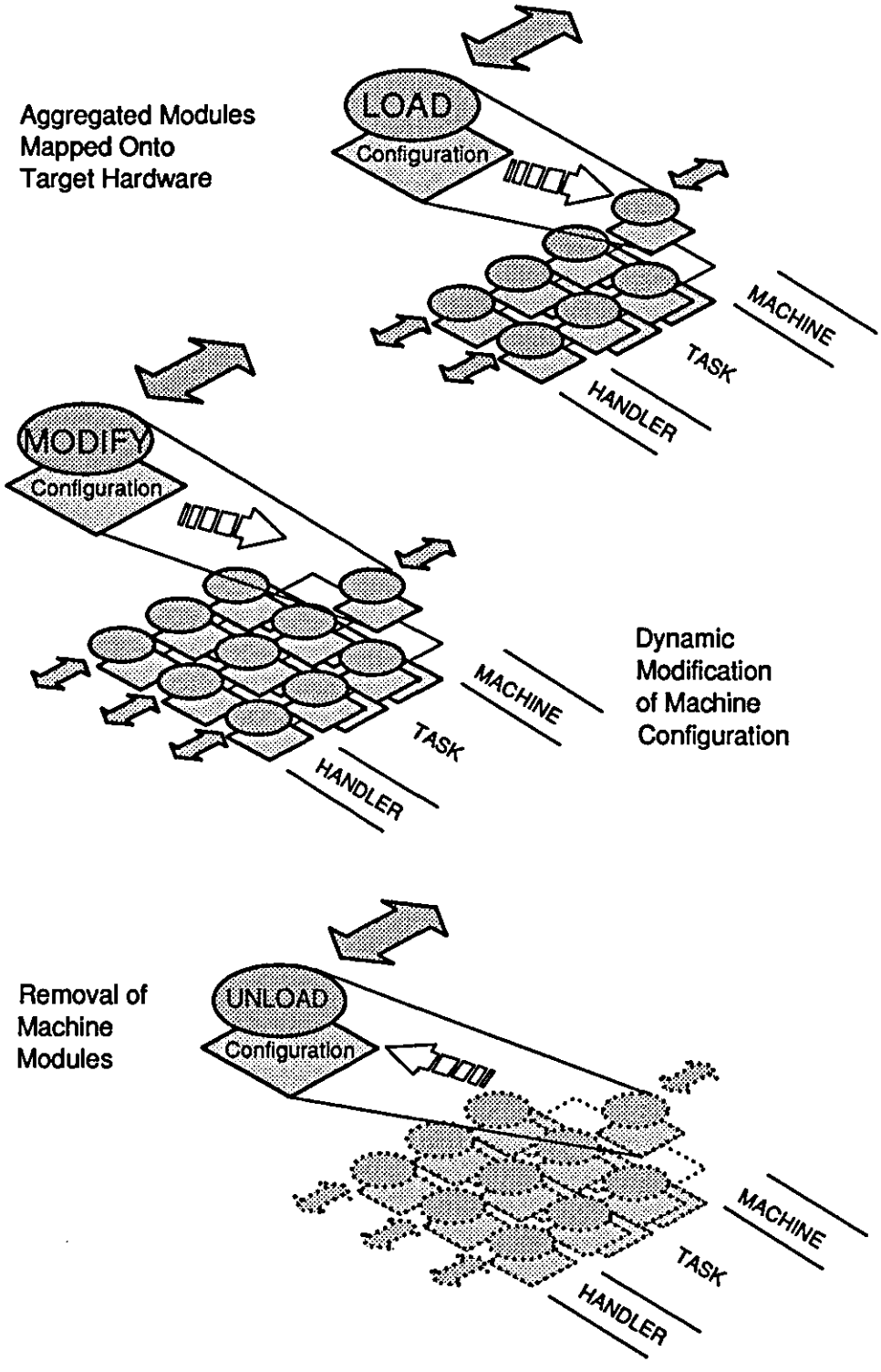


Figure 5.8. The concepts of machine aggregation, dynamic modification and removal.

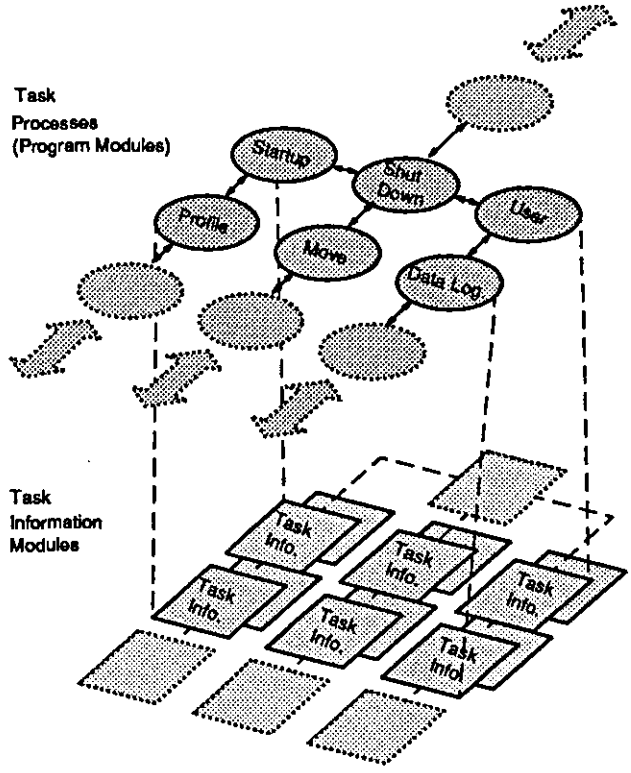


Figure 5.9. Multiple concurrent run time tasks.

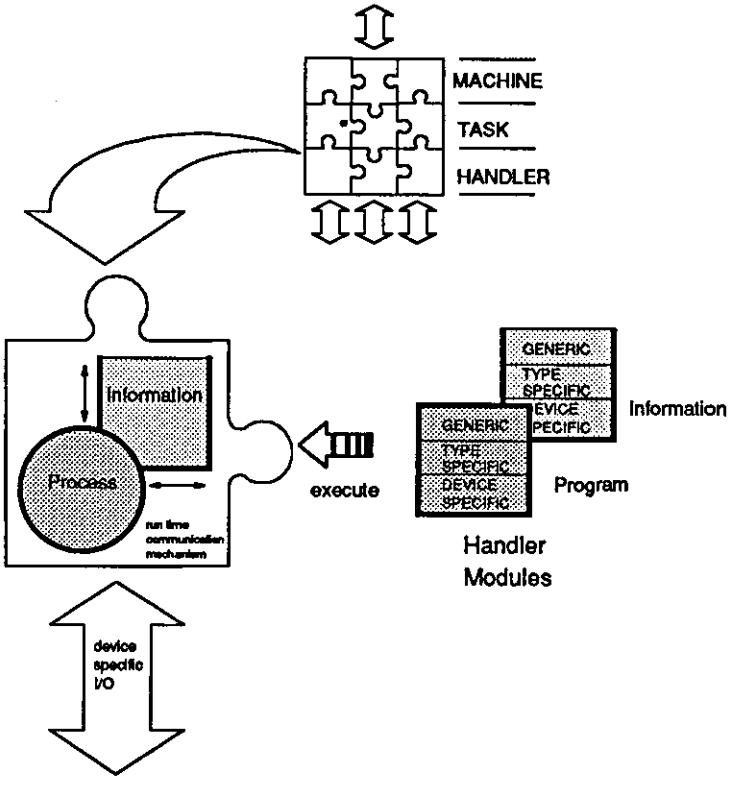


Figure 5.10. Handlers integrating specific devices of defined types in a consistent manner.

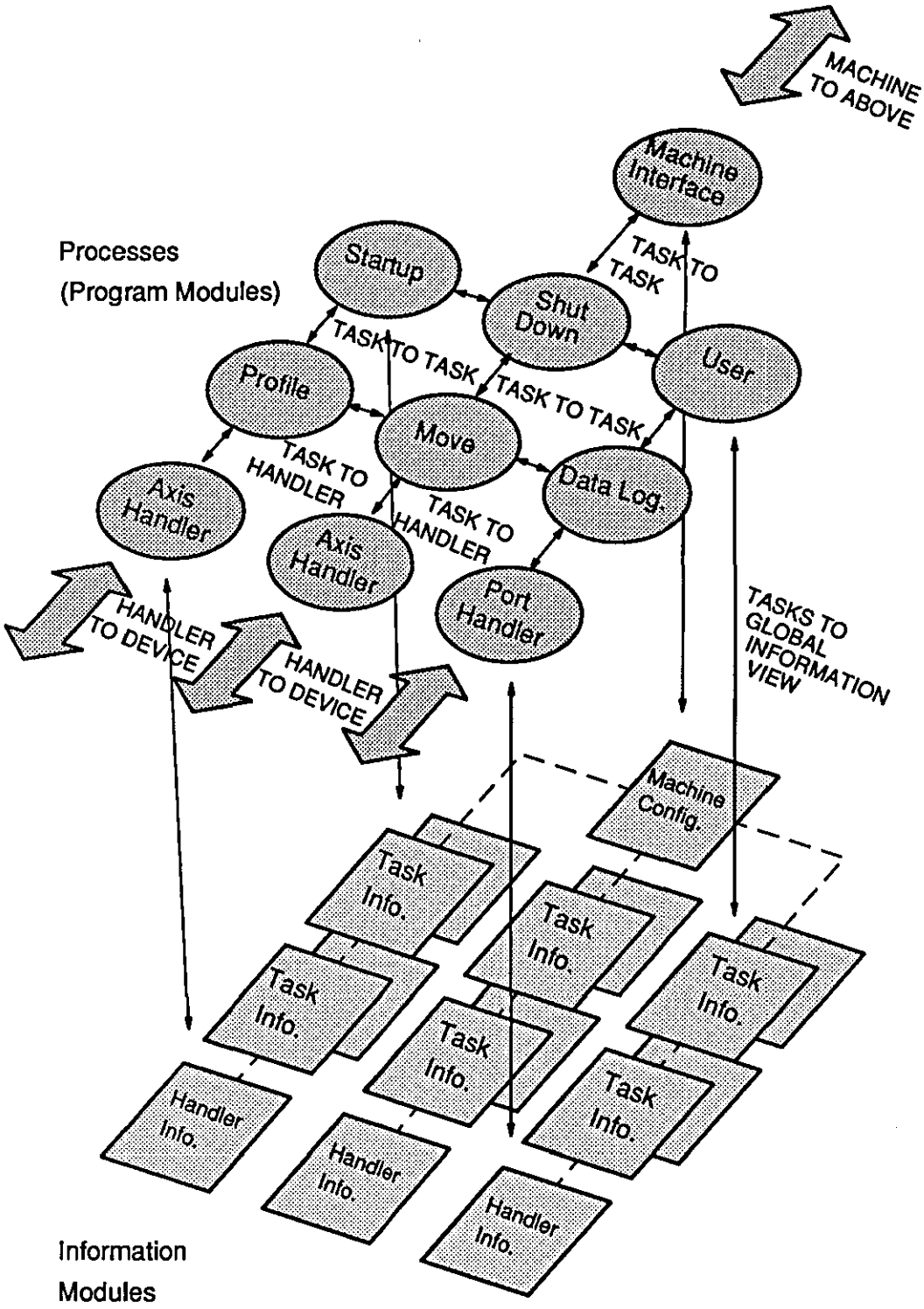


Figure 5.11. Reference Architecture communication paths.

5.7 SCOPE OF UMC REFERENCE MODEL AND ARCHITECTURE

The aim of using a Reference Model is to identify the essential parts, constraints and information sources required to describe a machine controller. The Reference Architecture defines the behaviour of each component of the Reference Model.

The UMC Methodology endeavours to be applicable to the widest possible range of industrial machine and process control applications. In order to encompass such varied manufacturing operations (with all kinds of applications related technology and performance requirements) it is obvious that *UMC cannot provide a Reference Model catering for all the details of each particular implementation. It is intended that the Reference Model and resultant Architecture will rather govern a system's inherent characteristics* providing information visibility and consistent interfaces with other elements in the manufacturing system. Therefore in addition to the Reference Model and derived Architecture the UMC methodology must guide machine controller vendors and users through the stages of a control systems life cycle. See figure 5.2. It is important to provide:

- methods for the description of *particular* machine controller functional requirements in terms of Reference Architecture modules and
- tools for the *aggregation* of these requirements into a run time machine controller.

5.8 PARTIAL AND PARTICULAR MODELS

The Reference Architecture is applied by including aspects specific to a particular class of machine including specific input and output devices, modes of operation and finally application specific requirements. The UMC machine control system development methodology must thus support models at different levels of specificity; the UMC Reference Model, libraries of modules termed Partial Models and Particular Implementation Models for each specific system. See figure 5.12. The concept of models at different levels of specificity is explained by Parunak [11] and was discussed in section 4.2.3.

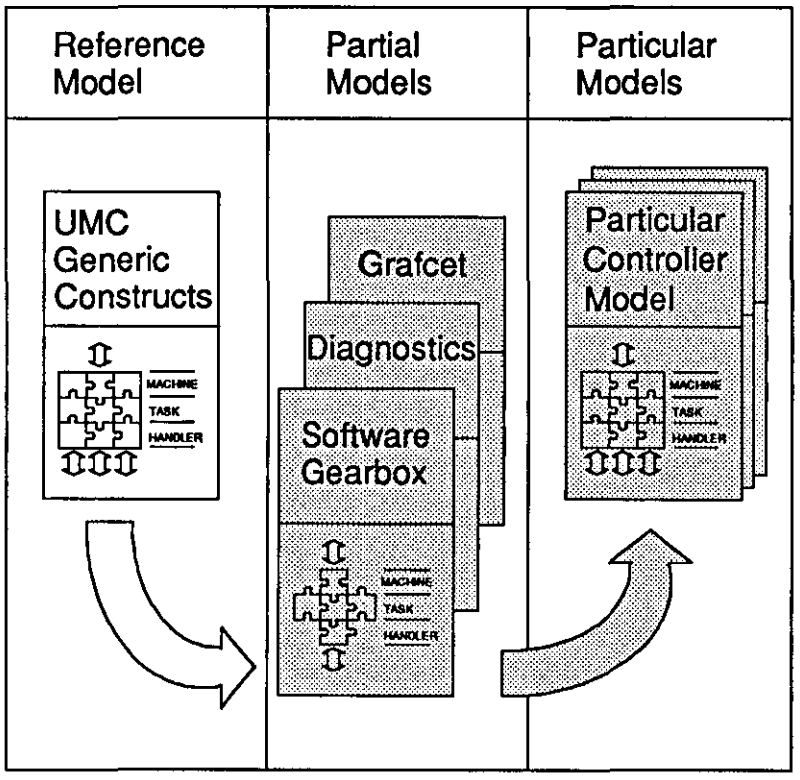


Figure 5.12. UMC modelling levels.

Partial Models are stored as a library of modules for given class(es) of machine which may be progressively extended. The generic modules in the Reference Architecture guide the creation of sufficient Partial architectural modules to serve the needs of a given class of machine and associated control devices. These modules exhibit function and/or implementation dependences but are not yet specific to a particular application.

Particular Models relate to a specific machine and are unique to that machine. An appropriate set of Partial control system modules are finally selected offering the functionality for a given application in a Particular Model which will be unique to that machine. It should be noted however that Particular Models are expected to evolve during a machines life cycle and are easily modified and/or extended.

The UMC framework thus contains three increasingly more application and implementation specific modelling levels which are:-

Reference Model and derived Reference Architecture	The generic level containing the basic constructs used to form components conforming to the Reference Architecture.
Partial Models	The Partial Model level can be seen as a library containing sets of modules conforming to the given generic types which cater for a given class of machine, task or device.
Particular Models	Each Particular Model contains the specific UMC components for a particular application. Mapping this Particular Model onto its target environment results in the aggregation of all the specified and selected modules which satisfy the requirements for the real-time control of a specific machine.

The Reference Architecture itself is adaptable to change. If a new component type is required (e.g. a new type of Machine, Task, or Handler), new generic types can be added to the Reference Model and a new Reference Architecture defined. Partial and in turn Particular variants of this new component can then be created in a consistent manner. The more specific UMC component modules are thus derived from more general entities. Figures 5.13 and 5.14 illustrate the modelling dimensions provided by the UMC methodology.

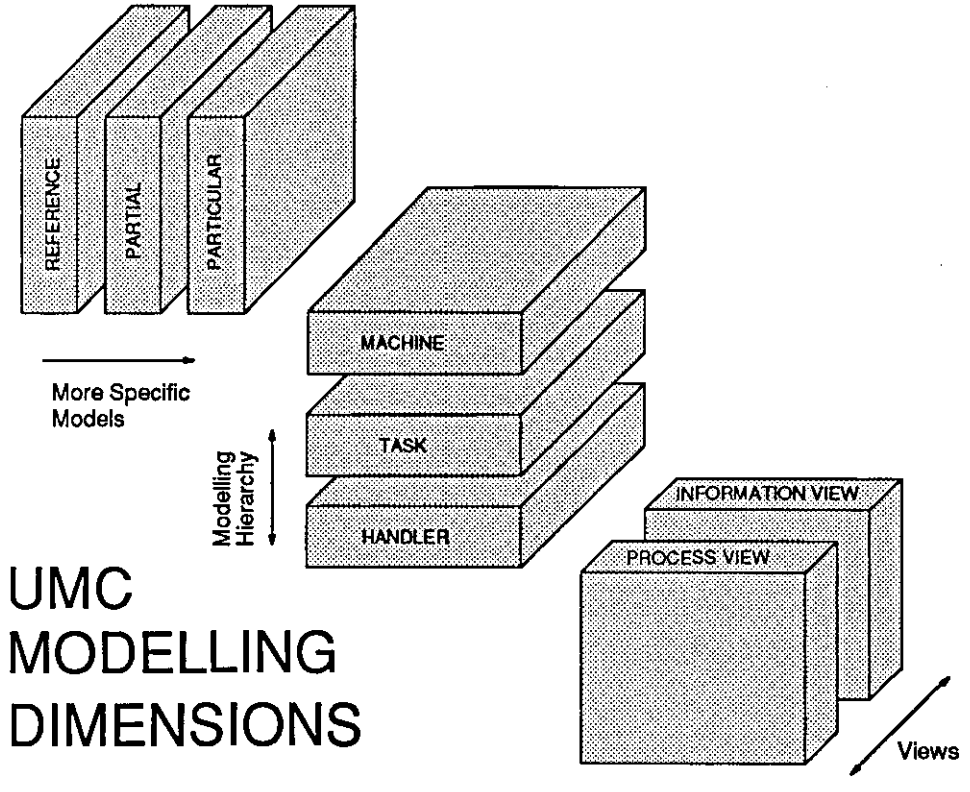


Figure 5.13. Dimensions of the UMC methodology.

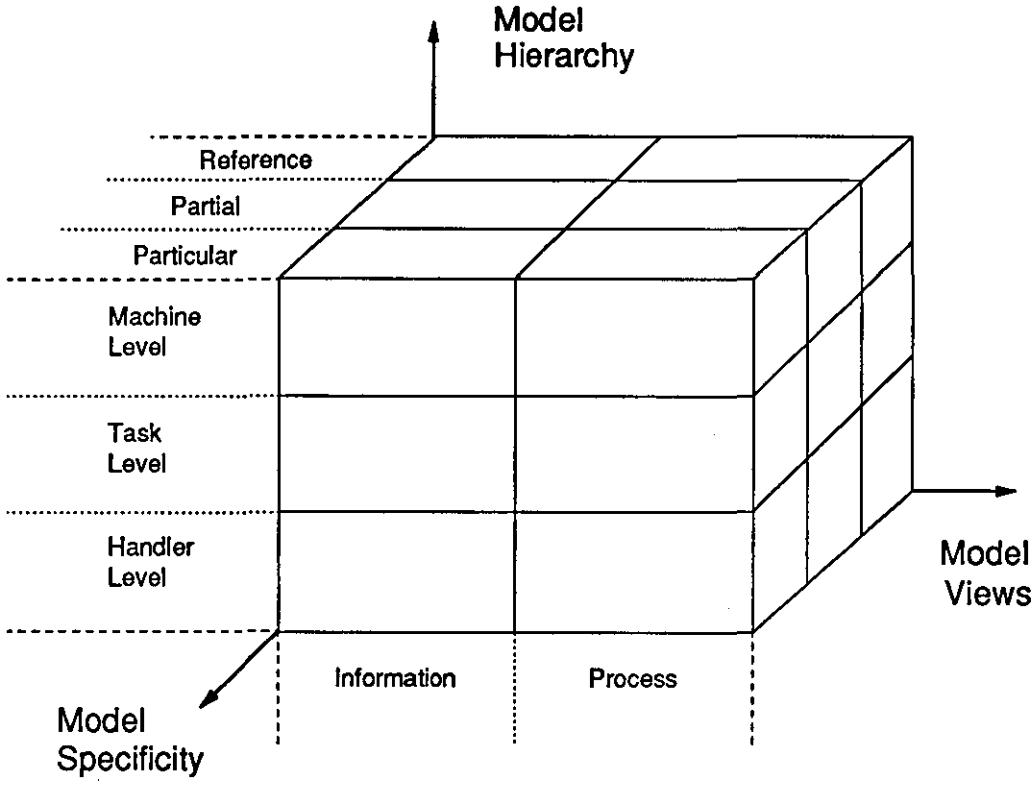


Figure 5.14. UMC modelling framework.

5.9 UMC IMPLEMENTATION CONSIDERATIONS

5.9.1 Requirements

The implementation of Particular UMC control systems obviously requires the initial implementation of appropriate development and run time environment(s). Tools within these environments must consistently support the design and run time aspects of the UMC methodology including the provision of:

- An extensible set of Reference Architecture guide-lines and constraints that govern how modules can be combined.
- Mechanisms for interconnecting Reference Architecture Components.
- Reuse libraries for UMC components with powerful search and retrieval mechanisms.

- Methods for assimilating new components into the reuse libraries (e.g. the creation of new types of Machines, Tasks and Handlers).

- Configurations of components for particular applications (i.e. the creation of Particular Models).

- The aggregation of Particular Models in target environment(s) for run time use or simulation.

- Maintaining existing Particular Models; maintenance, diagnostics, modification etc.

5.9.2 Development and Target Environments

UMC methodology must encompass and support the stages in a control system's life cycle from conception through design and implementation to maintenance and modification. Consistent development and run time target environments are required to support this life cycle. See figure 5.15. These environments can be categorised as follows:

- The *development environment* needs to provide support for the design and modification of machine control systems which is an on going activity for controller manufacturers, system builders and users. Within this environment many modelling techniques and associated tools will be applicable for the support of the Reference Architecture and the development of Partial and Particular control system models. Different sets of tools will be appropriate to each class of user. The development environment has at its core a data base holding libraries of Reference, Partial and Particular modules.

- The *target environment.* This is the target implementation environment for a Particular Model. It must support the real-time control requirements of a particular system. The adequate implementation of real-time target systems is therefore a critical part of the UMC methodology. The target environment consists of the physical control structure which may be distributed and a suitable software environment which supports the implementation of the Particular Architecture on the chosen hardware.

These two environments will be used interactively during a machine's life cycle and fully consistent development and target environments are therefore required by the UMC methodology. In some cases the same hardware may be utilised

for both development and target environments with appropriate software modules. Program and information modules will thus be interchanged between these environments during a controller's life cycle to enable its creation and subsequent support for diagnostics, maintenance and expansion.

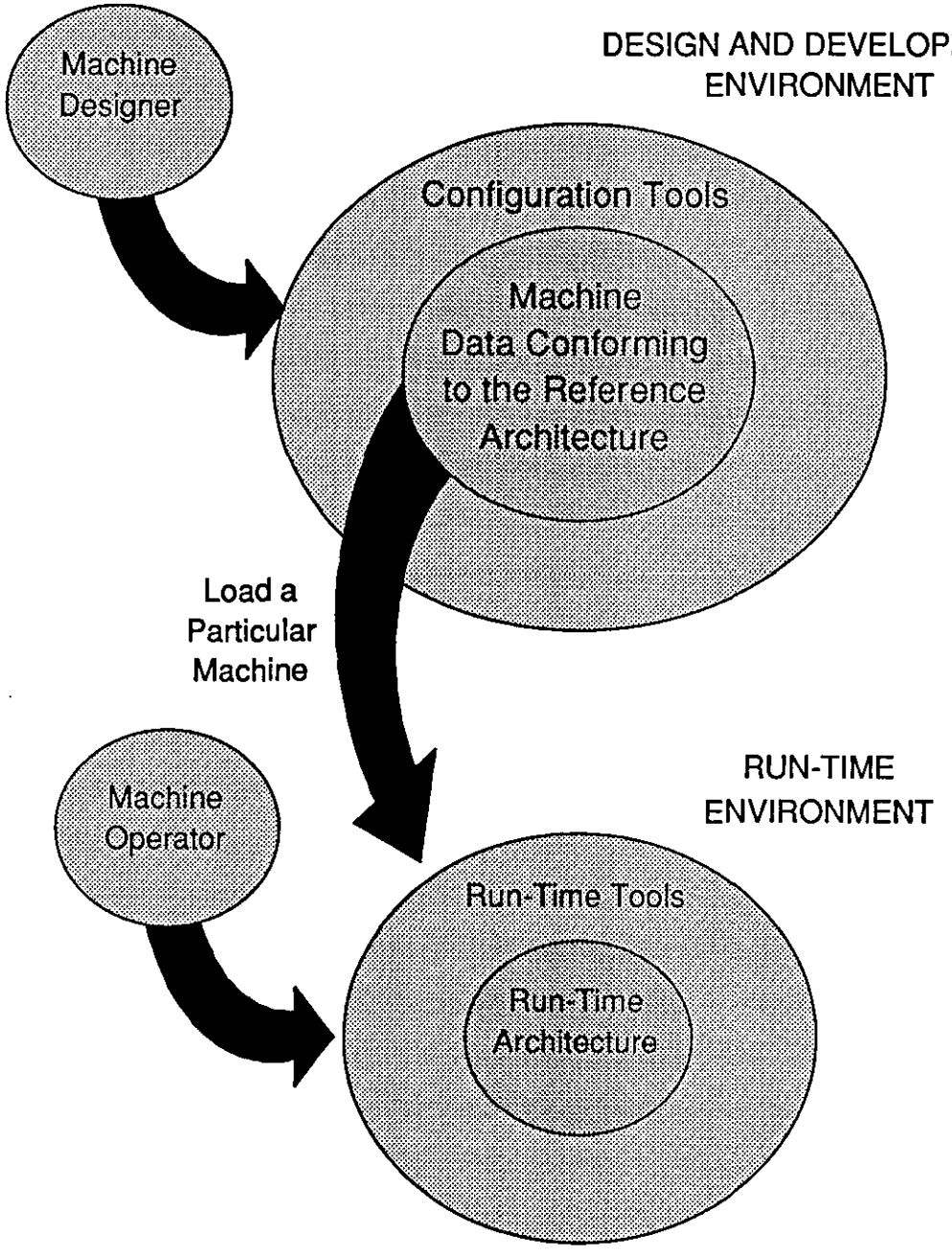


Figure 5.15. Schematic of UMC development and target environments.

5.10 SUMMARY

The UMC methodology aims to facilitate the design and implementation of computer controlled manufacturing machines in a generalised manner.

The UMC Reference Model provides a common framework for the discussion of machine control problems. The motive for using a model is the management of complexity. The Reference Architecture takes the Reference Model and defines the modular building blocks of which it is composed and how they fit together. A major motive for enforcing a consistent architecture is to enable design reuse.

The UMC Reference Model does not provide all the details of particular implementations or applications. It allows the description of progressively more application/implementation specific Partial and Particular Models from associated reuse libraries of UMC modules. Particular Models are used for the creation of run time control systems.

The next chapter looks at the selection of suitable enabling technology and chapter seven then describes the implementation of the UMC methodology using selected hardware and software.

Chapter 5: References

1 Anon., "Advanced Architecture for Universal Machine Control", SERC Progress Report, No. 2, Modular Systems Group, Manufacturing Engineering, Loughborough University, October 1990.

2 Thatcher, T. W. "Control System Architectures for Distributed Manipulators and Modular Robots, Phd. Thesis, Loughborough University, Dept. of Manufacturing Eng., October 1987.

3 Klevers, T. "The European approach to an 'Open System Architecture' for CIM: CIMOSA", Proc. 5th CIM Europe Conf., May 1989, pp 109-120.

4 Anon., "Reference Models for Manufacturing Standards", International Standards Organisation Report ISO TC184/SC5/WG1 N51, Version 1.1, September 1986.

5 Flatau, U., "Digital's CIM Architecture", Revision 1.1, Digital Equipment Corporation, April 1986.

6 Weston, R.H. et al., "The Need for a Generic Framework for Systems Integration", in "Advanced Information Technologies for Industrial Material Flow Systems", Ed. Nof S Y and Moodie C L, NATO ASI Series F53, pp 279-309, Springer-Verlag, 1989.

7 Anon., "HOOD", Microsystem Design, February 1989.

8 Boak, J. "A Fourth Generation Tool for Rapid Development of Reconfigurable Manufacturing Systems", 16th ESD/SMI Int. Conf. on Programmable Controllers, Detroit, Michigan, April 1987, pp 139-146.

9 Erkes, K. F. "Modelling a Physical Structure for Component Manufacturing", Proc. 3rd. CIM Europe Conf., IFS Publications, May 1987, pp 83-96.

10 Mayer R. J. "Unified SDM: The ICAM Approach to Systems and Software Development", Proc 3rd Int. Computer Software and Applications Conf., November 1979, pp 331-336.

11 Parunak, H.V. et al. "A Synthesis of Factory Reference Models", Industrial Technology Institute, Ann Arbor, Michigan, USA, September 1987.

6
Enabling Technology and the Role of Standards

6.1 INTRODUCTION

The previous chapter has described the essential features of a generic approach to machine control UMC which has been conceived at Loughborough University. This chapter aims to assess the capabilities of currently available technology and the role of standards in enabling an implementation of UMC. It is argued that the next generation of machine controllers should be based on a consistent set of industry standards.

In reviewing enabling technology it has been possible to suggest where existing standards are appropriate and where new standards are required. Chapter seven goes on to describe a proof of concept implementation of UMC using selected hardware and software.

6.2 THE NEED FOR STANDARDS

The role of standards in the lifecycle of a complex control device such as a machine controller is to facilitate the system design and development processes. Two categories of standards are commonly recognised.

Firstly de facto standards derived from accepted practice. This type of standard is prevalent in most areas of computing and control. Such standards become established simply because something is achieved in a manner which has been widely accepted by a large group of users. Examples include RS-232 serial communications between computers, the dominance of the MSDOS/PCDOS operating system, the use of relay ladder logic for the programming of PLCs etc. Often de facto standards are evolved by a single manufacturer or user.

A second class of standards can be viewed as those derived by a group of interested parties (often in the guise of a standards committee, body or institute, with representation from users, vendors and researchers in the relevant field) who ideally would begin their standardisation activity by conducting basic research into the re-

quirements of the problem being addressed. Thus standards from within this class originate from the derivation of a basic reference model that defines a generic approach to a problem which can then be addressed by appropriate implementation standards. Examples of this type of approach include the ISO Open Systems Interconnect (OSI) Model [1], the manufacturing reference models discussed in section 4.2.2 and the UMC reference model proposed in chapter five. Reference models therefore have a role in encouraging the adoption and standardisation of appropriate enabling technology.

In the context of aiding reference model implementation it has been perceived by Sauter and White [2] that standards if widely adopted can provide the following benefits:

- The user can concentrate on solving the real problem rather than being burdened by technical details.
- The user can employ components from multiple sources and be assured that they will integrate if they conform to the same standards.
- The technology used to implement systems can be determined more by the problem than by the need to integrate with non-standard devices.
- Users can be easily moved from one project to the next and can more easily perform system maintenance.
- Applications can be moved from one project to another.
- Appropriate standards can enable the reconfiguration of existing systems and also the progressive addition of new technology.
- Standards common to, and understood by both user and vendor are key to the successful implementation of a system to meet the real problem.

There are also potential drawbacks to the adopting standards. The use of standards may result in additional overheads, reducing overall system performance. Inappropriate standards may also restrain improvements in enabling technology and therefore become obsolete.

6.3 THE IMPLEMENTATION DOMAIN

The development process related to any system can be conveniently categorised within two domains; the essential problem domain and the implementation domain [3]. See also section 4.4.2. The essential problem domain is where a user applies his expertise to the solution of a problem and generates innovative ideas free from implementation considerations. The role of the UMC Architecture is seen as central to this activity. The implementation domain addresses the capabilities of the technology that is required to enable the practical implementation of the specified system. For the realisation of a practical UMC system this enabling technology includes the creation or selection of processing hardware, communications and information services, operating systems, programming languages, system component packaging and so forth.

6.4 IMPLEMENTATION REQUIREMENTS

Implementation of the UMC Architecture requires the creation of configurable and expandable physical and functional control structures in the form of electronic hardware and software.

Creating the necessary physical structure requires the use of modular hardware with suitable bus systems and/or networks to achieve interaction between the application processes involved.

The creation of the required processing functions and communication services which facilitate meaningful information transfer between these processes (and with external devices outside the UMC system boundary) is fulfilled by the system software. The concurrent activities carried out by a given machine controller will typically require the operation of many concurrent computer software and hardware based processes. As discussed in section 5.6.3 a UMC machine controller must be capable of communicating with interface devices (typically for sensing and actuation) within its scope of control, with its peers and with higher level manufacturing control elements. These processes may be distributed across several processors. Standards are needed to define how these processors should interact. Here the term interaction is used to delineate the need for:

(1) the sharing of information resources in a meaningful manner,

(2) the co-ordination, supervision and control of concurrently operating activities to achieve the desired goals and,

(3) the communication services required to facilitate (1) and (2).

The physical structure and performance requirements of a given machine obviously strongly influences the configuration of its control system hardware [2]. Hardware configuration includes:

- the selection of processor(s) with appropriate power, distribution and inter-relationships,
- selection of sensor and actuator interfacing devices,
- consideration of communication distances and required rates of data exchange,
- methods of representing, storing and accessing information,
- environmental factors etc.

As discussed in chapter 3 many current industrial machine controllers most notably PLCs have adopted proprietary bus systems, communication protocols and hardware architectures unique to each controller type but there has recently been a progressive shift towards more open standards [4], [5]. Integration has become recognised as an important issue. Without standard interconnections, the creation of an environment to support every type/brand of unit controller (offering appropriate interfaces to the myriad of control devices and associated computer equipment) seems an unattainable goal.

Standards have a vital role to play in information engineering which encompasses the representation, access, storage and management of information [6]. The expression and decomposition of the actions also needs to be achieved in a consistent manner and future standardisation of CASE tools is likely to have significant impact on this issue. See section 4.4.4.

The remainder of this chapter focuses on the choice and selection of enabling technology for the proof of concept UMC system implementation described in chapter seven.

6.5 PROCESSING HARDWARE

Today most machine control system hardware is packaged in the form of circuit board level products. These are usually categorised by the form of interconnection bus which they employ to enable their integration into complete control systems by the addition of suitable peripherals. See section 6.6.

In board level control systems, currently two processor families predominate the Motorola 680x0 and the Intel 80x86 although the choice is almost limitless [14], [7]. The selection of processing hardware is currently often dictated by the chosen operating system environment since currently most operating system implementations are hardware specific being at least partially coded in assembler language in order to achieve adequate real-time performance [8]. The emergence of the new generations of more powerful reduced- and complex-instruction-set computer (RISC and CISC) processors will allow portable operating system implementations based on C as the development language [9]. See section 6.8.

6.6 BUS SYSTEMS AND NETWORKS

6.6.1 Introduction

Buses and networks are used for a wide variety of control system communication needs ranging from integrated circuit interconnection on the same board up to local area networks connecting systems within the same factory.

The wide variety of open architecture bus and network systems now available and the multitude of applications to which they are put makes their classification difficult. They may however be broadly categorised according to their range and speed [10]. Figure 6.1 shows the application areas for a number of major interconnects which are applicable to machine control.

Parallel backplane bus systems generally fulfil the needs of integrating master processor(s) with interface devices to form tightly coupled "local" control configurations [15]. In such configurations the hardware elements are typically physically located within a metre of one another and are able to communicate at raw data rates greater than 1 Mbit per second [10]. See figure 6.2. These local controllers may then require connection with other distributed devices at the same,

higher or lower levels in the manufacturing system. Communication may be achieved with a variety of serial bus or network systems. See figure 6.3.

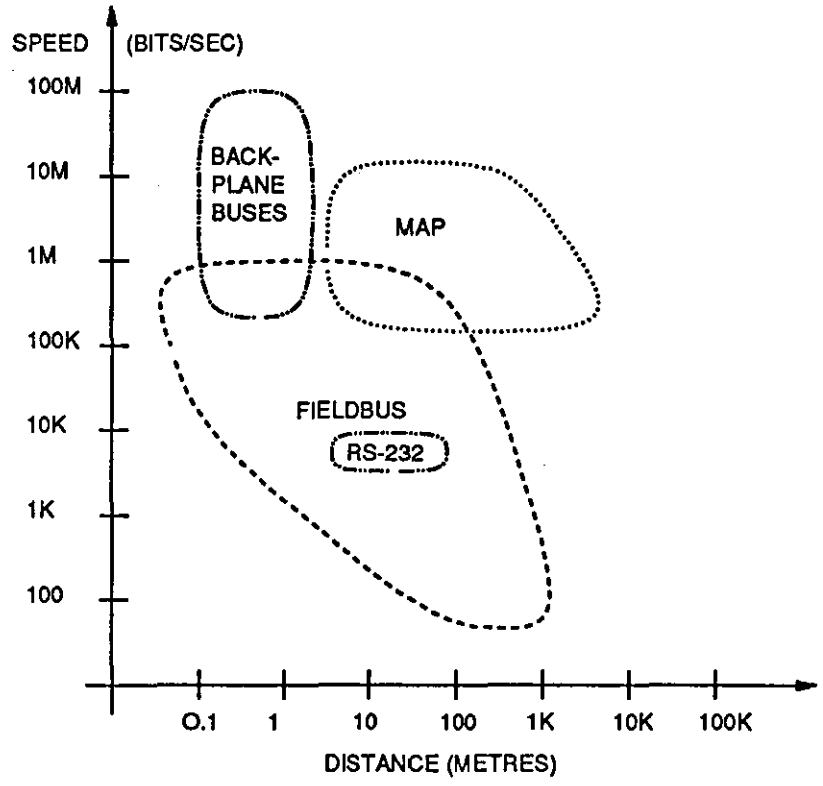


Figure 6.1. Application areas for different types of interconnect.

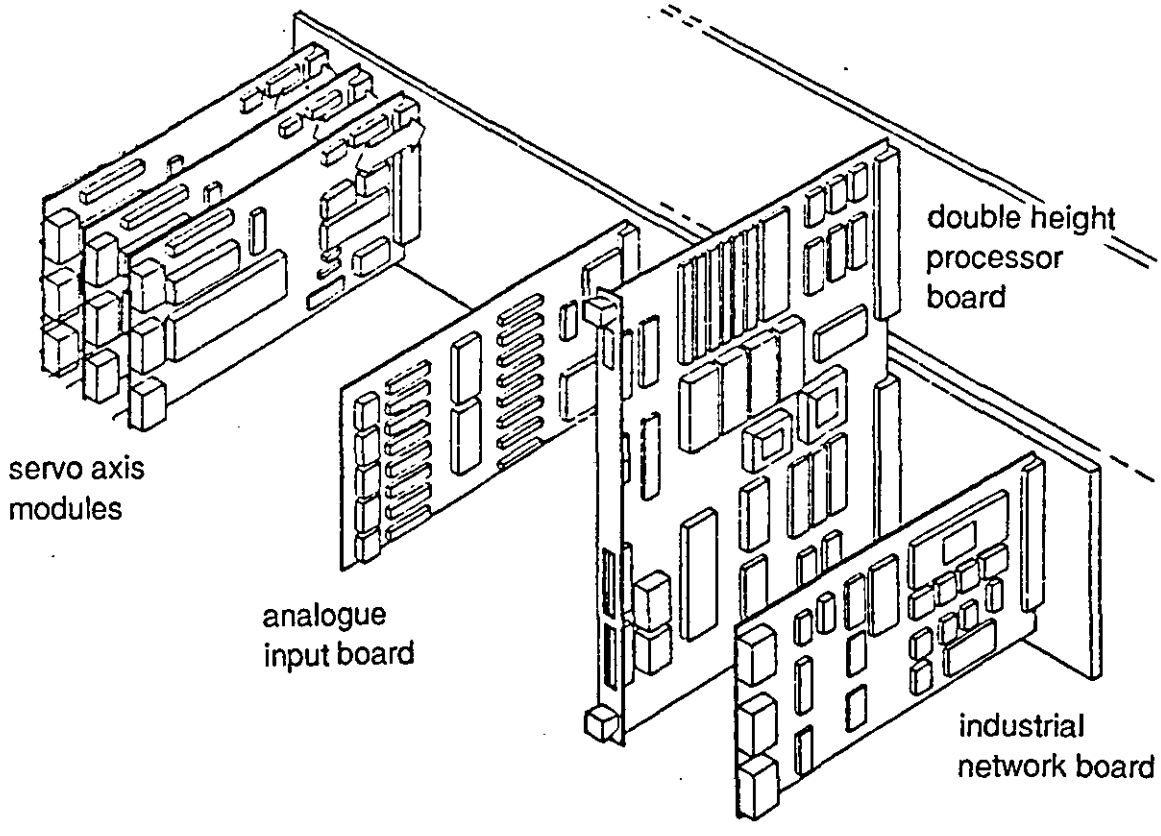


Figure 6.2. Typical physical form for a parallel bus system.

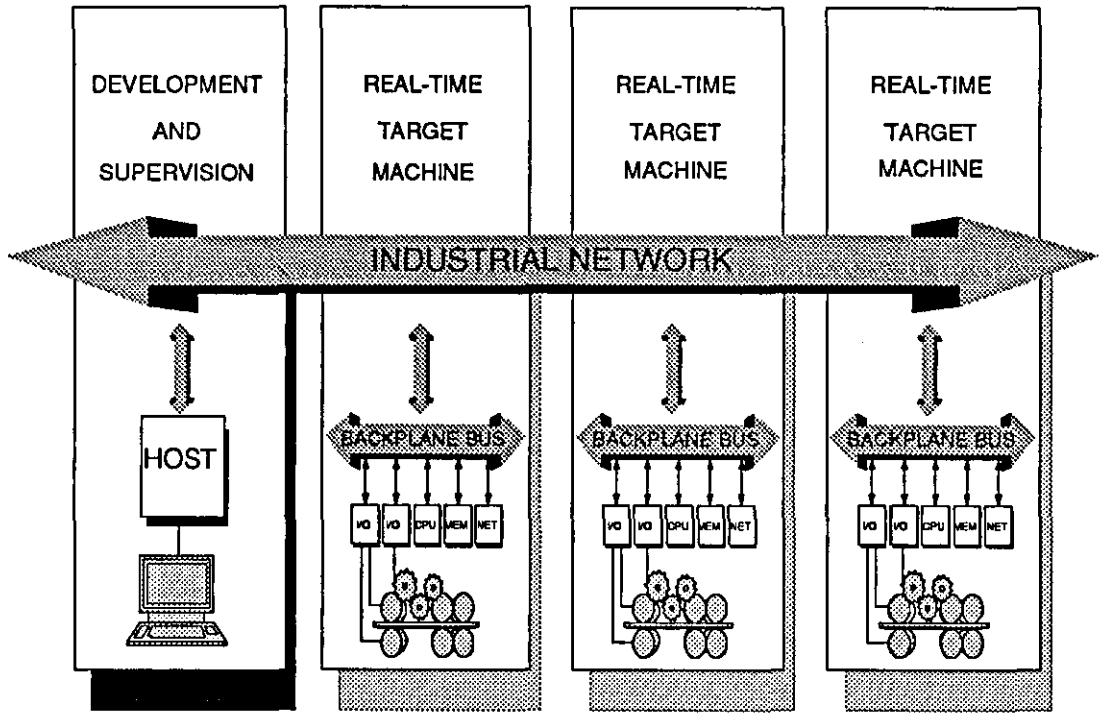


Figure 6.3. Typical physical form for the interconnection of local control configurations.

6.6.2 Parallel Bus Standards

Machine controllers commonly require the capability to interface in a consistent manner to a wide range of sensor and actuator interface devices which may themselves possess local processing capabilities e.g. motion controllers or vision systems. The emergence of open architecture parallel bus standards has provided the opportunity for multiple vendors to supply a range of solutions to meet such needs in a compatible manner.

When computers were first introduced they were generally manufactured with all their processing, memory and interface devices directly attached to an internal processor specific bus system in a dedicated manner. This approach made it difficult if not impossible to connect devices made by other manufacturers. The Digital Equipment Corporation (DEC) played an important role in overcoming this restriction through the implementation of backplane bus systems in their computers. The introduction of the PDP-11 and Unibus with memory mapped I/O brought to the microprocessor industry the concept of using a well defined bus as a standard digital interface for microprocessor based products and peripherals [11]. All microcom-

puter parallel bus standards today tend to be variations on the same theme providing a mechanical and electrical specification (in terms of connector dimensions, topology, signal levels etc.) and a corresponding set of protocols.

A standard bus allows a variety of board level devices from multiple vendors to be put together to meet the requirements of a particular application. Board level computer systems are now well established and standardisation has allowed widespread hardware compatibility. The choice of bus systems is still a bewildering one with many competing standards [12]. A wide variety of both intelligent and passive interface devices are available at the board level e.g. motion controllers (see section 3.3.6), image processing systems and other I/O devices of almost every conceivable type [13], [14]. Industrial control system builders are utilising these bus standards to achieve hardware compatibility within their equipment [15]. The machine controller designer may thus quickly access new technologies without having to develop the hardware himself. This modus operandi has made a significant impact on the automation of complex manufacturing operations [15].

A standard bus for machine control must meet many requirements. The following requirements are commonly suggested [16], [17]:

- Support for 16 and 32 bit processors to enable the efficient implementation of high level languages and standard operating systems in real-time applications.

- Support for multiple processors. Communication with LANs, intelligent control devices, human interface functions etc. often requires the support of dedicated processors on the same backplane.

- Efficient inter-processor communications.

- Support for multiple interrupt sources. Many machine control functions tend to be interrupt driven.

- Good noise immunity, good mechanical support for boards, connectors design to withstand shock, repeated insertions and a harsh environment, support for different board sizes.

There are many 8-bit and 16-bit bus systems which are widely established [18], [19], [20]. The most widely utilised of these are G64bus, STDbus, STEbus, Multibus I and IBM-PCbus although they all fail to meet one or more of the requirements presented above.

There are currently five 32-bit busses which are being standardised by the IEEE in the US namely VMEbus, Multibus II, Futurebus, Fastbus and Nubus [21], [17]. Of these VMEbus (IEEE P1014, IEC 821) currently has the widest vendor and product support of all the 32 bit buses [22]. VMEbus adequately meets the requirements for an industrial bus. Its single height variant is both compact and cost effective for the majority of industrial applications [23] and can be utilised as an I/O bus on complex full 32 bit double height VME processing systems.

6.6.3 Network Standards

Network standards are intimately related to the achievement of effective manufacturing integration. Their scope therefore encompasses not only interconnection issues but issues of high level protocols, including information representation, storage and access [24]. Detailed evaluation of this field is beyond the scope of this thesis and the aim of this section is to highlight emerging standards which are considered likely to have a significant impact on machine controller interconnection.

Various solutions are now commercially available to enable communication between distributed devices at the upper levels of a manufacturing hierarchy and have attained relatively high degrees of standardisation. The MAP and TOP standards are notable examples for factory and office communications respectively. The most important feature of these standards is the adoption of vendor independent open communication using the ISO-OSI seven layer model [1].

The specification and development of MAP represents a major step forward in communication standards within the industrial environment [25]. MAP 3.0 onwards includes RS-511, the standard for Manufacturing Message Services (MMS). RS-511 provides a basic set of functions for communicating with any programmable device on the factory floor. Companion standards then define the communications with a specific device (such as a PLC, CNC or robot) [26] which are regarded as specific classes of machine. Controllers created using the UMC methodology could use such a messaging service perhaps together with the more application related information services provided by AUTOMAIL [27]. See section 10.2.

The MAP and TOP standards are however optimised for the requirements prevailing at the upper levels of the manufacturing hierarchy, e.g. user-friendly, non-time-critical transfer of large quantities of data over long distances. Standard solutions for transferring information between distributed devices at lower levels have so

far been lacking despite evident demand for the standardisation of sensory interfaces, for example on machine tools, robots and special purpose machines [28].

There have been efforts particularly in the process industries to define a lower level network than carrier-band MAP. See figure 6.4. The Field Instrumentation Bus (or "fieldbus") aims to specify the communication link between intelligent sensors and actuators close to a process, and machine or process controllers. The primary requirements for this bus are reliability and low cost [29].

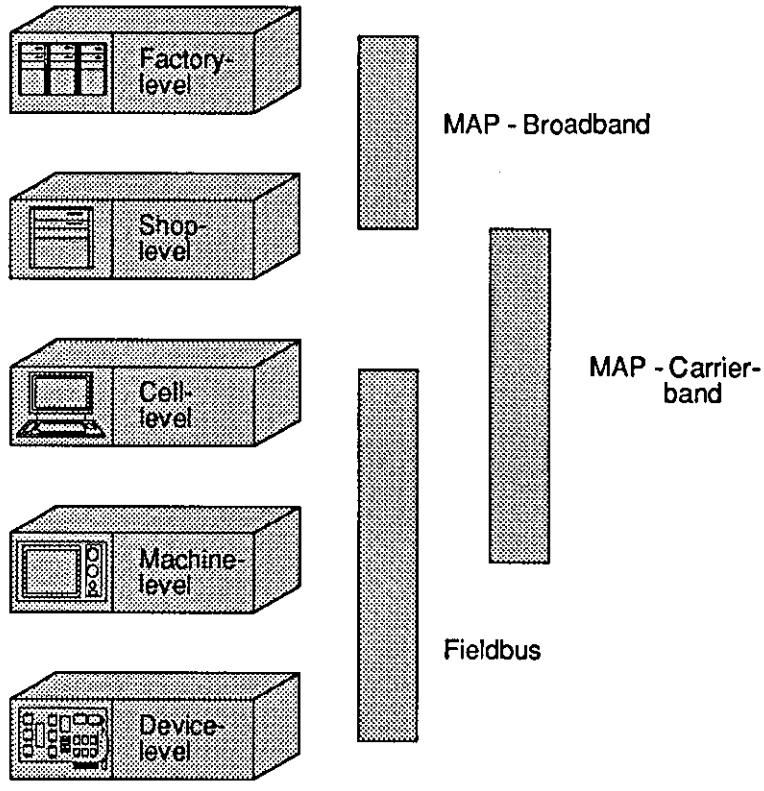


Figure 6.4. Classification of manufacturing networks.

In the context of machine control two main types of application for fieldbus systems would appear to exist in the manufacturing sector:

(1) Small closed loop systems used to link intelligent I/O devices to machine controllers. See figure 6.5.

(2) The mutual interlinking of machine control systems and their connection to higher level cell or line controllers. See figure 6.4.

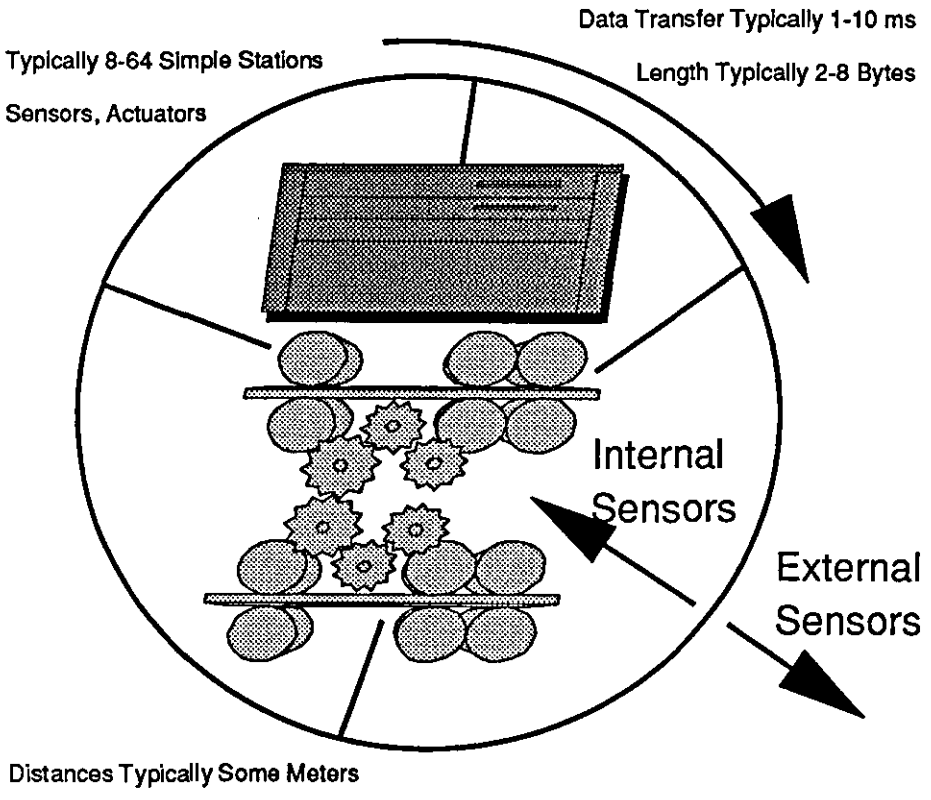


Figure 6.5. Fieldbus in a local control loop.

The creation of fieldbus is still in its infancy and will become meaningful only if accompanied by appropriate standardisation of physical media, communication protocols etc. Working parties have been formed in ISA SP-50 [30] and IEC WG6 of TC65C [31] to define these requirements. Intel's Bitbus [32] which already has a wide user base and a separate proposal by Rosemount [2] are being considered as candidates for the fieldbus standard. MIL-STD-1553 originally defined as a sensor network for military aircraft is another candidate for this low-level sensor network [33]. Industrially many vendor specific systems are however in widespread use for example Bitbus, Arcnet [34] and Topaz [35].

6.7 CONTROL SYSTEM SOFTWARE

Control system software has traditionally been written in the form of specific solutions to specific problems [4]. Typically a controller implemented in this manner will be completely tied to the hardware on which it was first written with no portability and no obvious upgrade path. The system software will (particularly after the inevitable bug fixes and enhancements) invariably be untransferrable, unreadable and largely undocumented [36]. This type of dedicated software can be observed in

the majority of current robot controls, PLCs, process controls, CNC controls, assembly and packaging machines for example.

In contrast to the dedicated approach described above the UMC methodology (see chapter five) takes a generic approach to machine control through the use of a reference architecture and requires a modular real-time software environment to support its implementation. While the concept of modular hardware is now well established, the technology to support modular software solutions has only recently become available [15]. UMC seeks to develop new applications quickly and ease the migration of existing software onto new hardware platforms, as required. Standard programming languages and operating system environments can help to achieve these goals.

The author perceives three important needs which have to be satisfied for the efficient generation of standardised real-time software:

(1) A standard real-time operating system for the target machine controller(s) with the required features and adequate efficiency. The operating system needs to support software modularity and hardware independence allowing applications software to be moved easily to different manufacturers' hardware as technology progressively improves.

(2) A standardised programming language which is efficient for system programming and which is able to fully support all the features of the chosen operating system.

(3) A standardised system development environment with software tools designed specifically to support the selected target operating system and programming language.

6.8 REAL-TIME OPERATING SYSTEMS FOR MACHINE CONTROL

6.8.1 Introduction

Machine controllers are usually classified as examples of real-time embedded microprocessor systems [37]. Such embedded systems contain microprocessors programmed to carry out control functions but they are not in themselves regarded primarily as computers. Indeed the user of an embedded system need not even be aware that a microprocessor lies within the device.

The traditional approach to creating embedded real-time systems has been to use assembly language to implement monolithic designs on early eight bit microprocessors such as the Zilog Z80 [38]. The low performance and small address space of eight bit microprocessors usually required non-modular, non-portable assembler language based implementations which were costly to design, code and test, and difficult to maintain.

As embedded systems including industrial machine controllers became more complex, new development methods had to be considered to help the system designer. With progressive improvements in processing power there was a migration away from assembler language to more productive high level languages. As hardware costs continued to decrease and processing power improved, applications evolved from single thread control loops towards more complete systems with many tasks and external interrupts. With software costs rising dramatically real-time operating systems were developed offering the promise of enabling more productive methods for system design and implementation [49].

From the point of view of a control system manufacturer or system builder an operating system can cost a great deal of money and very often the uninitiated user has no real picture of its function. An operating system must be learned and understood before it can be used and there is complete reliance on the company supplying it for support. These negative statements are made at the outset of this discussion as they may provide part of the reason for the initially slow uptake of real-time operating systems for industrial control.

The benefits gained from using an operating system are very important but initially not so obvious. An operating system can potentially provide a "virtual", hardware independent environment which offers a set of built-in functions or system calls closely related to the system programmers requirements [39] as illustrated in figure 6.6.

Real-time operating systems now include provision for high level languages, modern software development techniques and portability. These features can be exploited to enable the implementation of the UMC Architecture. Factors of particular significance include adequate provision for modularity, multiple tasks and adequate response to real-time events.

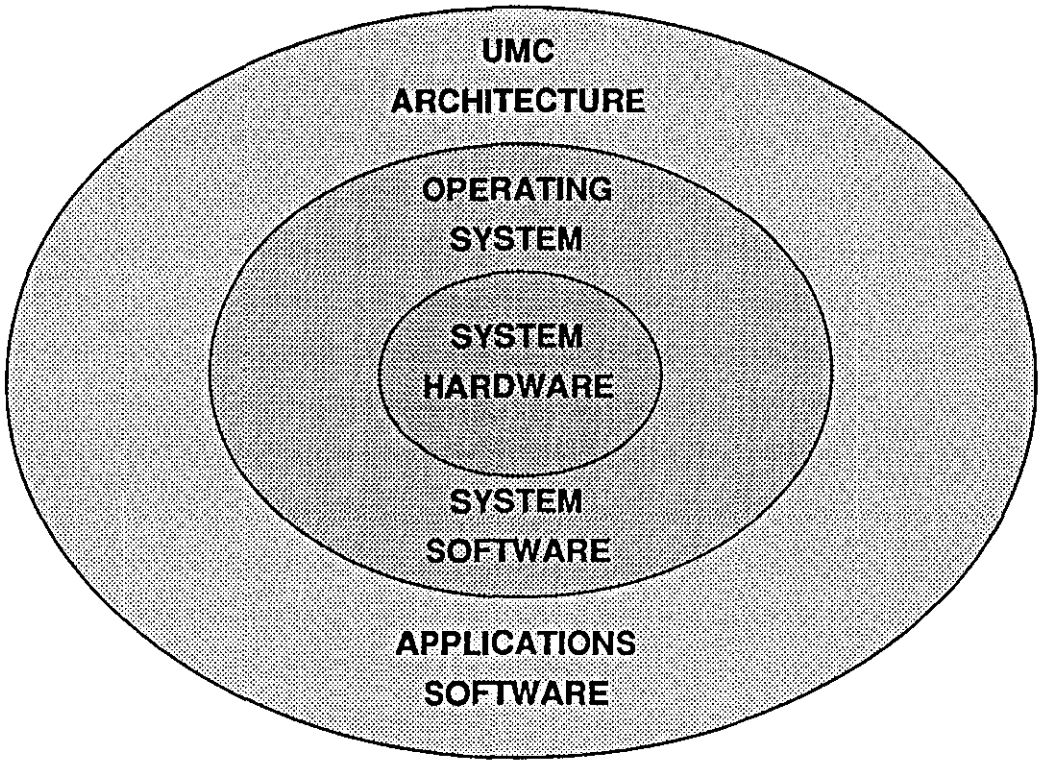


Figure 6.6. Relationship between hardware and software in enabling the implementation of UMC.

6.8.1.1 Modularity

In view of the desire to provide extendible and reconfigurable control systems in a standardised manner there is the need for an operating system which is inherently modular. Modular software is for example needed to support both applications programs and device interfaces enabling these items to be added or removed from a system at will, without the need to reconfigure the system software each time modifications are made [15].

6.8.1.2 Multi-Tasking

In simple computer operating systems for example MSDOS, there is a unique CPU/memory/code combination for each process [39]. There are however compelling reasons for introducing a more flexible relationship between processes and the hardware resources which run them. A rigid allocation of a process to CPU/memory implies that no new processes can be introduced into a system without altering the controller's physical configuration. In machine control applications this

is usually a serious limitation. Most applications would naturally be best split into many processing activities as discussed in section 4.5.2. The size of each process is typically too small however to justify its own individual processor/memory (except in the case of low level processor intensive tasks, for example closed loop motion control). Multi-tasking allows many processes to share the same processor. For real-time operation different priorities are usually associated with different tasks depending on their relative processing requirements.

6.8.1.3 Response to Events

A real-time computer system must respond to events that occur in the process being controlled. Machine control requires a response to real-time events which must be both fast and predictable.

6.8.2 Development and Target Environments

A typical programming environment for control system design and implementation consists of two parts; an "office" based development environment and industrially packaged target machine controller(s) as illustrated in figure 6.7. The development environment may employ the same real-time operating system as the target system allowing direct code development or may use a non-real-time system programming environment typically MSDOS, UNIX or VMS [60]. In either case the development environment will offer software tools for communication, remote debugging and supervision of real-time processes on the remote target system(s).

Real-time executives (sometimes referred to simply as real-time kernels) are similar to real-time operating systems except that they are designed only for embedded use and are thus supplied with a set of development tools for use in a non-real-time host environment.

Many alternative development and target environments now exist and may be used in different combinations as figure 6.8 illustrates for a few notable examples. Real-time operating systems and executives may be produced by computer hardware manufacturers specifically for their own hardware or developed by a software systems company for one or sometimes a range of processor families.

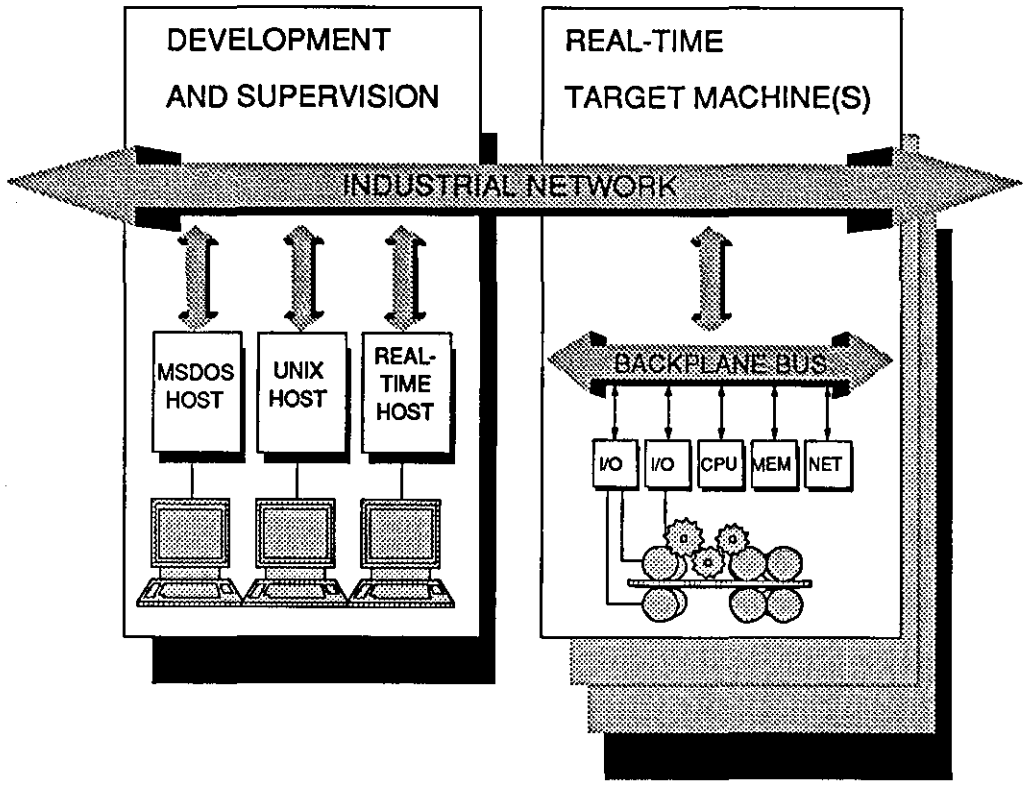


Figure 6.7. Schematic of development and target environments.

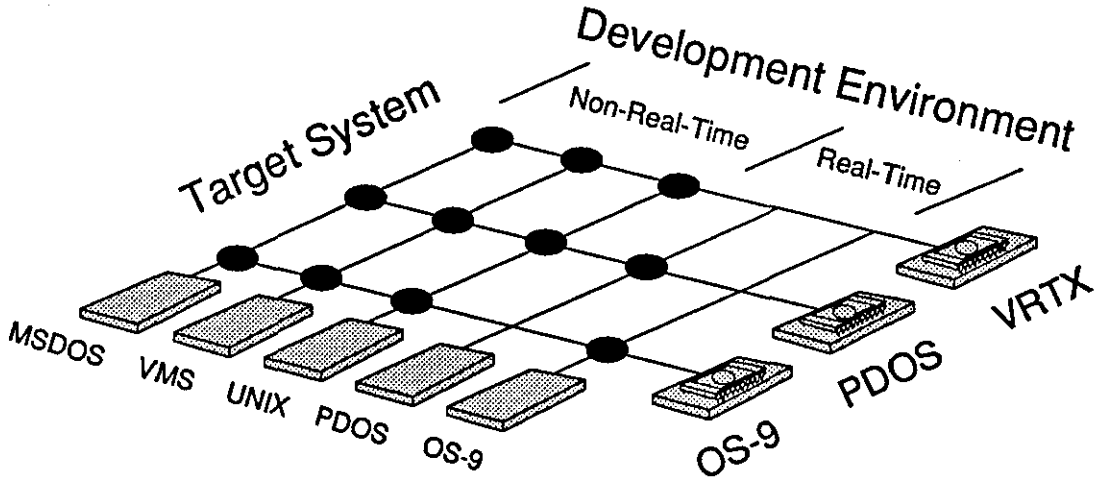


Figure 6.8. Possible combinations of development and target operating system environments.

6.8.3 Real-Time Operating Systems

Real-time operating systems provide a set of functions and utilities to support both embedded target and system programming activities. Real-time operating systems offer sufficient capabilities to provide a complete environment for program development and a reduced version of this environment is then employed on the run time target machine.

In the United States DEC's VAX and MicroVAX computers have attained a dominant position in major industrial applications. VAXELN is a real-time operating system developed by DEC specifically for its own computers [40]. Similarly Intel offer RMX II.3 for its range of 80286 and 80386 microprocessors [41].

OS-9 [42] and OS-9000 [43], Uniflex [50] and PDOS [44] for example are established products from independent vendors with a wide user base and many third party suppliers of software. These operating systems have been ported mainly to the Motorola 680x0 family of processors although they are coded predominantly in C and designed to enable portability.

TRON is a potentially very significant operating system standard under development in Japan. It is known to be a portable, real-time operating system with extensive user and device interfacing capabilities. Although targeted at both office and control environments TRON has as yet seen little practical application outside Japan [45].

Several UNIX "compatible" real-time operating environments have become available in recent years either in the form of complete operating systems for example Regulus [46] or real-time executives with UNIX like functionality [50].

6.8.4 Real-Time Executives

VRTX [47], pSOS [48] and C Executive [49] are well known real-time executives. Real-time executives or kernels are essentially run time only environments which utilise a large non-real-time environment (typically MSDOS, UNIX or VMS) for development support [50]. Real-time executives are thus supported by cross development facilities, cross-compilers etc. Prototype software modules may however often be executed on the host machine before cross compilation to one of a range of target environments [51]. Since they manage fewer functions, real-time executives

generally perform somewhat faster than real-time operating systems when used in appropriate applications [50].

6.8.5 Distributed Real-Time Systems

An increasing number of operating systems are now emerging offering varying degrees of capability to support a distributed operating environment. This may be localised multi-processor operation on a parallel backplane or operation over a network in a physically distributed manner to serve the needs of large distributed machines or process control applications.

The level of support and transparency provided by distributed operating systems varies widely. Meglos a distributed environment for robotics used for research at AT&T Bell Laboratories [52] and Mars an experimental distributed real-time process control operating system environment [53], provide comprehensive facilities which are completely transparent. 0S-9 for example provides more limited facilities of file transfer, remote login, virtual terminals and the remote execution of processes [54]. A fully distributed variant of the OS-9 operating system may however emerge in the future [55]. Distributed variants of UNIX are now available for example D-NIX and BiiN [62].

6.9 SYSTEM PROGRAMMING LANGUAGES

As discussed in section 4.5.3 the use of a standard system programming language is considered highly desirable. The practical choice for a system programming language is intimately related to the selected operating system. Such a language is required for the implementation of the UMC Reference Architecture. As illustrated in figure 6.6 this system software programming is distinct from the programming of run time applications tasks although the same programming language might be used in both cases depending on functional requirements. See chapter seven.

C [56] has now almost become a de facto standard for system software and is the preferred language of most software houses [10]. Its adoption has enabled the transfer of software packages between computers of widely differing types. It offers the required features for system programming and has the great strength of being highly standardised with libraries of functions handling operating system and device dependent features. Other newer languages notably Ada [57], [58] and Modula-2

[59] may offer advantages in terms of productivity but currently have neither the maturity nor widespread acceptance of C as a standard system programming language [10]. The specification of Ada for use in U.S. military systems will however exert a strong influence for its more widespread adoption in the future [62].

6.10 OPERATING SYSTEM STANDARDISATION

6.10.1 Introduction

Effective standards for operating systems have been slow to emerge [2]. A few operating systems have dominated particular market sectors because of the predominance of certain computer hardware vendors notably IBM and DEC.

Whether used for system development or run time use any "standard" operating system should be capable of being ported to different types of processor in order to effectively exploit the rapid developments in hardware technology [60]. In embedded machine control systems the need to achieve adequate real-time performance has been a serious obstacle to the adoption of portable methods of operating system implementation since portability is generally achieved at the expense of some operating efficiency [10].

The goal of offering a more standardised approach has recently focused attention on the operating system UNIX as a development environment, the system programming language C and UNIX "compatible" real-time operating systems for the target machine controllers themselves.

6.10.2 UNIX as a Standard Development Environment

UNIX is a powerful multiuser, multi-tasking operating system for developing software. UNIX offers several attractions in the search for appropriate standard system software [61]. It is available for a wide variety of computers, it is portable, it provides a good development environment and has well structured support for computer networking. UNIX is now a de facto standard for many system programmers [62]. Users can take advantage of an extensive set of utilities supporting program development and system to system communication. In 1989 there were over 15,000 commercially available applications available under UNIX [63].

Unfortunately many different variants of UNIX currently exist although there are efforts both in the US (/usr/group and IEEE P1003) and Europe (X/Open) towards defining a common operating system standard based on UNIX System V, release 4. This process has however been complicated by the Open Software Foundation which is working to promote AIX from an IBM-only version of UNIX as a portable standard [62].

In the longer term the IEEE POSIX (Portable Operating System Interface for Computing Environments) standard seeks to define a UNIX-style software interface that operating systems should present and thus provides a firm technical basis for building UNIX-type operating systems [64]. The underlying notion of the POSIX standard is that any program designed to use the standard interface will run under any operating system that presents that interface. The standard defines different levels of conformance. "Strictly conforming" applications will use only the facilities described in the POSIX standard and the ANSI C language [65].

6.10.3 Real-Time UNIX

In its standard form UNIX is unsuitable for use in real-time target systems and does not provide a practical or cost effective operating system for machine control [66]. There are however obvious attractions to providing a single consistent environment for both development and run time use. This is particularly applicable to the iterative development cycles which evolving industrial machine control systems are likely to require.

The following list presents some of the problems which currently make UNIX unsuitable as a target environment for machine controllers:

- Writing Device Drivers. It is very difficulty to write device drivers for UNIX. (This also applies to most other operating systems. POSIX seeks to overcome this problem [65]).

- Real-Time Response. Many processes within a machine controller need to be executed within a guaranteed amount of time. UNIX does not however provide a known maximum time to activation.

- System Resources. A machine control system of the UMC type might typically be expected to have between 10 and 100 concurrent processes. UNIX is very inefficient both in terms of system resources

and time when supporting a large number of relatively simple processes.

- Provision of an Interrupt Mechanism. There is no provision for pre-emptive hardware interrupts to allow a rapid response to external events.

- Event Management. Mechanisms are needed to support the management of multiple events. There is no event management mechanism in most current versions of UNIX.

A number of working parties are now trying to address these problems. A real-time subcommittee of /user/group [67] and a real-time group at General Motors [68] are cooperating on defining the necessary extensions to UNIX. AT&T has also begun to address some of these problems and System V release 2 onwards adds messages, semaphores and shared memory which are all needed in machine control applications [69].

The POSIX standards activity includes a subcommittee for real-time UNIX. Draft standards were expected by early 1990 [62]. It is intended that the POSIX standards will also include the best features from distributed and parallel versions of UNIX [70]. The fact that GM have chosen UNIX as the standard operating system for the factory floor means that there will be a large market force to correct the deficiencies of UNIX in the real-time environment [71].

6.11 DATA BASE MANAGEMENT SYSTEMS

In addition to their primary control function modern machine control systems are facing increasing requirements for information storage and retrieval. Not only do machine controllers require local real-time information processing capabilities but they must also be able to effectively exchange information with other systems in the manufacturing environment. The management of complexity and change are key issues which have led to increased requirements for information storage and retrieval. For example:

- The potential complexity of distributed/reconfigurable control systems demands extensive data base management capabilities to document and support the system lifecycle e.g. machine description, configuration and modification. See section 5.9.

- At run time increasing emphasis is now being placed on product quality and process improvement. Accurate product related information is critical in order to implement more flexible processes/machines able to run multiple product variants and switch products quickly.

The three traditional data base structures commonly used to implement information systems are hierarchical, network, and relational. Hierarchical and network data base structures are defined at the time of data base definition. This results in low data redundancy and very fast access time. The drawback with these configurations is that modifying the structure later can be very cumbersome [72]. Relational data bases, on the other hand are built from straight linear files which are related through key fields. This allows configurability by adding or modifying data base relationships. Although it involves substantial overheads, the relational approach is generally adopted where an extendible centralised information resource is required [73]. Performance limitations can however often restrict their use in real-time applications [74].

Normally associated software known as the Data Base Management System (DBMS) allows the applications user to access information with minimum effort and technical knowledge. The DBMS is a complex software system which constructs, expands and maintains the data base and is of key importance [75].

Access to an information resource from application specific programs is commonly achieved by the use of a query language. SQL (Structured Query Language) is the most common query language and has been accepted as a reference product by ANSI (American National Standard Institute) [76].

The information capabilities of most current machine control systems have been based on proprietary data structures and proprietary data managements systems. There are now many attractions in integrating machine control systems with more standardised data management systems, utilising products like EMPRESS, INFORMIX, ORACLE and INGRES. The benefits of such an approach include:

- The reliability of a standard proven product once established.

- Faster development time using a standard product with standard development tools rather than developing a new, proprietary data base manager.

- The need to interface with third party systems. SQL for example provides a de facto standard application and user interface to data.

- The creation of standard software tools and applications like report generators and statistical analysis packages which can be used with standard data base management products.

6.12 HUMAN INTERFACES

Efficient control system *design and development* requires a very flexible user interface. Windowing systems, now offer appropriate interfacing capabilities for the multitasking development and target environments currently being utilised. See section 6.8. Examples of proprietary windowing environments include products from SUN [77], Microsoft [78], and Digital Research [79]. The most notable emerging standard in this field is the MIT X-Window system which is designed for controlling windows and graphics on intelligent terminals, PCs and workstations.

X-Windows sits between the operating system and the application programs and defines how application programs can communicate with the user in a graphical and visual way. X-Windows is not in itself a user interface but it allows the developer to produce such interfaces to a system. One of the main reasons for the acceptance of X-Windows as a standard is that it is hardware independent [80].

The use of a standard interface for the *target machine operator* has generally received little attention by machine control system designers. The operator is currently often faced with a bewildering variety of displays and controls. Each vendor has a different approach to displaying information to the operator and obtaining input. As yet little research has been done regarding the most effective way of communicating with factory personnel [81], [82].

6.13 SELECTED ENABLING TECHNOLOGY

6.13.1 Introduction

The machine control system research at LUT has spanned about six years and over this period enabling technology has progressively evolved. The flexible manner in which the UMC development and target environments have been able to

incorporate technological improvements illustrates one of the strengths of adopting a generic approach to control.

6.13.2 System Hardware

The use of appropriate hardware and communications standards has enabled the utilisation of physical system components from a variety of sources. Processing hardware as been obtained predominantly from Syntel Microsystems [83] with Intelligent Motion and I/O controllers from Quin systems[1], [84], Galil Motion Control [85] and Martonair [86].

The initial development and target environments were G64bus based systems utilising Motorola MC68008 processing hardware. These systems have been progressively supplemented by more powerful VMEbus systems with MC68020 processors and cross-development facilities.

As networking standards have emerged and been supported by suitable hardware and operating system vendors a distributed computing environment has been evolved at Loughborough for UMC research. See figure 6.9. From initial point to point RS-232 links, Topaz and Ethernet based communications have been adopted. These facilities have enabled the interconnection of development/supervisory and target machine environments for file transfer, remote process execution and debug as illustrated in figure 6.7. See also section 6.13.4.

6.13.3 Real-Time Operating System

The real-time operating system selected for UMC machine control research at Loughborough is OS-9. It incorporates many features of UNIX such as, multitasking, a hierarchical file structure, a "shell" user interface, utility programmes, resident assemblers, linkers, debuggers, and a number of high level languages including a Berkeley compatible C compiler [87]. Unlike standard UNIX however OS-9 executes in real-time and is fully ROMable [88].

---

[1]Quin Systems specialise in the development of computer control systems for high speed machines and are the major collaborator on the research grant.

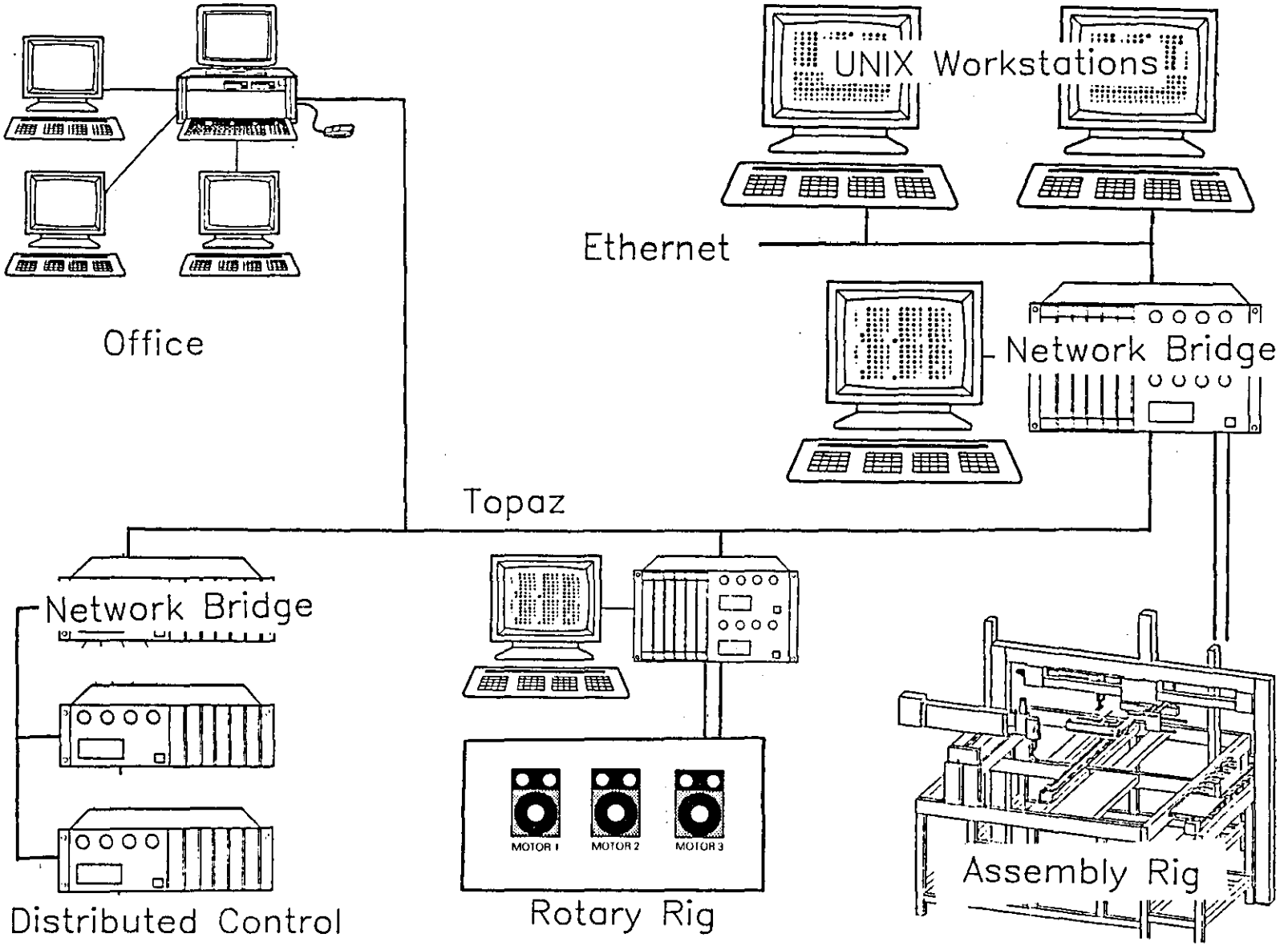


Figure 6.9. UMC processing network

6.13.4 Development Environment

OS-9 is a self hosting real-time environment since the same environment can be used for both code development and real-time control. See section 6.8.2 and figure 6.8. OS-9 can also be used with a UNIX, VMS or MS-DOS/OS-2 host environment by utilising the UniBridge/VMS-VAX/PCBridge communications and development packages [89].

The control system research environment at LUT has evolved from a single OS-9 machine, to a distributed system of host and target OS-9 computers linked with network communications. Greater flexibility has been provided through the availability of cross development facilities in both MS/PC-DOS and UNIX environments. The importance of these links is very significant in the long term since they enable the integration of support tools, including data base managements systems, CASE and advanced user programming methods such as Grafcet, with the core real-time capabilities of UMC.

6.13.5 Data Base Management

UMC development environment requires a central information base. A simple custom approach has been adopted at Loughborough in the proof of concept implementation of UMC. See section 7.6. Although this simple custom data base system has been effectively created, for more extensive usage a standardised Data Base Management System (DBMS) is considered necessary [90]. A standard DBMS interface is therefore to be adopted for UMC system development and run time support. The data base INGRES and its associated SQL library are currently being evaluated [91].

6.14 SUMMARY

With the emergence of standards for modular hardware and recent advances in real-time software environments suitable technology currently exists to allow the implementation of the UMC methodology. A proof of concept implementation of UMC using selected enabling technology is described in chapters seven and eight.

The ideal of a standard "tool box" which provides an open and consistent approach encompassing all aspects of control system creation is however still largely unrealised. For many aspects of machine control systems effective industry stan-

dards for enabling technology have yet to fully emerge. In particular there are currently no effective standards for real-time operating systems and system programming languages although a real-time variant of UNIX and the programming language C (or a close variant of it such as ANSI C) will perhaps become effective standards. There is also a severe lack of effective applications related standards for machine control which the UMC reference model may help to stimulate. Standard information models are also not yet available which can adequately describe manufacturing systems (including programmable/configurable machines) and their interrelationship with product models.

Chapter 6: References

1 Zimmerman, H "OSI Reference Model - The OSI model of architecture for open system interconnection", IEEE Trans. Commun. Vol 28, April 1980, pp 425-432.

2 Sauter J. A. and White J. F. "The Role of Standards in the Development of Workstation Controllers", Sixth Annual Control Eng. Conf., Rosemount, Il, May 19-21, 1987, pp 98-107.

3 Ward P. T. and Mellor S. J. "Structured Design for Real Time Systems", Vol 1, Section 1, Yourdon Press, 1985, pp 3-39.

4 Hauser, N. "PLC or VMEbus: An open question?", Microsystem Design, March 1990, pp 8-9.

5 Anon. "Series 90 PLCs now available", Automation, April 1990, MBC Publications, pp 3-5.

6 Banks, D., "Systems thinking, systems practice", Microsystems Design, February 1990, pp 16-17.

7 Anon. "G-64 Board Directory", A directory of all processing boards available for the G-64/G-96 bus system, G-64 Only, Vol 2, No 1, Gespac Publications, 1986.

8 Smith D. "A Close Couple", Microsystem Design, April 1989, p 15.

9 Fischer W. "Real-Time Computing Demands Standards", Computer Design, October 1988, p 79.

10 Anon. "Technology Focus, Bus Systems and Networks", Electronic Engineering March 1983, pp 117-128.

11 Stone H. S., "Microprocessor Interfacing", Addison-Wesley, 1982.

12 Lieberman, D. "The open-architecture bus world: too many choices?", Computer Design, October 1988, p 77.

13 Anon. "The OS-9 Sourcebook", Hardware Systems and Products, Microware Systems Corporation, Des Moines Iowa 50322, 1988, pp 5-148.

14 Anon. "The STEbus IEEE1000 Product Guide", Issue 5, Techpress Publishing, 1989.

15 Webb, M., "Building Sophisticated Motion Control Systems with a High-Level Computer Language", Seminar - What's New in Electric Motion Control, Drives and Controls Conf., Coventry, 10 March 1988, pp F1 - F6.

16 Titus J. "Industrial Busses", Eng. Design News, February 1987, pp 114-126.

17 Curran M. "The search for a better expansion bus", BUSCON/88-EAST, Computer Design, October 1988, p 88.

18 Pabouctsidis C. "Using the G-64 Bus in Midrange Industrial Applications", G-64 Only, February-March 1986, pp 7-14.

19 Marks P. "Which Bus?", Microsystem Design, July-August 1987, pp 10-11.

20 Prtak L. "The 'work-a-day' PC", Microsystem Design, May 1989, pp 10-11.

21 Borrill, P. L. "Microstandards Special Feature: A Comparison of 32-Bit Buses", IEEE Micro, December 1985, pp 71-79.

22 Pri-Tal, S. "VMEbus gears up for 1990s computing", BUSCON/88-EAST, Computer Design, October 1988, p 84.

23 Hunt D. and Hodson K. "The case for single height VME", Microsystem Design, April 1987.

24 MacKinnon, D. et al., "An Introduction to Open Systems Interconnection", Freeman, Oxford, 1990.

25 Weston R. H. "Industrial Computer Networks and the Role of MAP, Part 1", Microprocessors and Microsystems, Vol 10, No 7, September 1986, pp 363-370.

26 Anon. "ISO 9506-N Manufacturing Message Service: A core specification, together with a set of companion standards for specific application use. Numerical Control companion standard, Robot companion standard, PLC companion standard, Process control companion standard", 1988.

27 Weston, R. H. et al., "The Need for a Generic Framework for Systems Integration", NATO AST series book on "Advanced Information Technology for Industrial Materials Flow Systems", Springer Verlag, 1989, pp 279-306.

28 Pfeifer T. and Komischke M. "Low-level communication systems for sensors", Proc. 3rd CIM Europe Conf., pp 39-46, May 1987.

29 Kochar V. "Network solutions for sensors", Automation, November 1987, pp 45-48.

30 Anon., "ISA SP50", Instrument Society of America, Standards and Practices Working Group 50, PO Box 12277, Research Triangle Park, NC 27707, USA.

31 Anon., "IEC/TC65C/WG6", International Electro-technical Commission, Technical Committee 65 Industrial Process Measurement and Control; Sub Committee 65C Digital Data Communications for Measurement and Control Systems; Working Group 6 Inter-Sub-System Communication.

32 Rant J., "Intel and Industrial Automation", The Bitbus Architecture, Solutions, Intel Publications, September-October 1985, pp 2-5.

33 Ledamun D. and Goodwin M. "Exploring the possibilities of the 1553B data bus", Electronic Engineering, March 1983, pp 147-152.

34 Anon. "ARCNET Factory LAN Primer", Contemporary Control Systems Inc., 2500 Wisconsin Avenue, Downers Grove, Illinois USA 60515, April 1988.

35 Anon. "Topaz - A guide to industrial local area networking", Syntel Microsystems, Huddersfield HD1 3PG.

36 Glad, A. S. "Software Engineering Guide for Real-Time Systems Development", Sixth Annual Control Eng. Conf., Rosemount, Il, May 19-21, 1987, pp 188-199.

37 Foulger, R. "Choosing Languages for Realtime Embedded Microprocessor Systems", National Computing Centre, Oxford Rd, Manchester, 1981.

38 Osborne, A. "Introduction to Microprocessors", McGraw-Hill, Berkeley, 1978.

39 Allworth, S.T., Introduction to Real-Time Software Design", Springer-Verlag, New York, 1981.

40 Shapiro, S. F. "Robotic systems learn through experience", Computer Design, November 1988, pp 54-68.

41 Willis T., "What Makes Up Real-Time?", Seventh Annual Control Eng. Conf., May 1988, Section IX, pp 1-11.

42 Smith D. "Real-Time ROMable OS-9 and the VMEbus", Microsystem Design, June 1987.

43 Anon. "OS-9000 Real-Time Operating System, Product Code: OS NANA NASL, Microware Systems, Des Moines, Iowa 50322, September 1989.

44 Roper P. "Gespac 68000 PDOS An Overview", G-64 Only, Spring 1987, Gespac Publications, pp 9-11.

45 Sakamura K. "TRON Project 1988: Open Architecture Computer Systems," Proc. Fifth TRON Project Symp., Springer-Verlag, Tokyo, 1988.

46 Anon. "Regulus Real-Time UNIX", Alcyon Corporation, 6888 Nancy Ridge Drive, San Diego, CA 92121.

47 Anon. "VRTX - Versatile Real-Time Executive", Product Description HRI#02701, Hunter Ready, 445 Sherman Ave., Palo Alto, CA 94306-0803.

48 Hudson J. "pSOS Real-Time Operating Systems", Technical Report 43, Unit-C Ltd., Worthing, Sussex, BN14 8NT.

49 McMahon, S et al. "C Executive - not just another real-time kernel", Real Time Systems Ltd., PO Box 70, Viking House, Douglas, Isle of Man.

50 Falk H. "Developers target UNIX and Ada with real-time kernels", Computer Design, April 1988, pp 55-56.

51 Wison R. "Real-Time Executives take on the Newest Processors", Computer Design, February 1989, pp 88-105.

52 Gaglianello R. D. and Katseff H. D. "A Distributed Computing Environment for Robotics", IEEE Publication CH2282/86-0000-1890$01.00, 1986.

53 Kopetz H., et al. "Distributed Fault-Tolerant Real-Time Systems: The Mars Approach", IEEE Micro, February 1989, pp 25-40.

54 Anon. "The OS-9 Catalog", Microware Systems, Des Moines, Iowa 50322, 2nd Edition, September 1989.

55 Dibble, P. "OS-9 MP - The Multi-Processor OS-9", Proc. of European OS-9 Conference, Vivaway Ltd, John Street, Luton, LU1 2JE, 1987.

56 Kernighan, B. W. and Ritchie, D. M. "The C Programming Language", Prentice-Hall, 1978.

57 Anon. "Reference manual for the Ada programming language", United States Department of Defense, July 1982.

58 Volz, R. et al. "Using Ada as a programming language for robot-based manufacturing cells", IEEE Transactions on Systems Man and Cybernetics, Vol 14, No 6, 1984, pp 863-878.

59 Wirth, N. "Programming in Modula-2", 2nd edition, Springer-Verlag, 1982.

60 East, I. "Computer Architecture and Organisation", Chapter 10 - Survey of contemporary processor architecture, Pitman, 1990.

61 Howard A. "UNIX all round", Computerised Manufacturing, April 1989, pp 33-34.

62 Falk, H. "UNIX rivalries cloud developers' choices", Computer Design, March 1989, pp 49-54.

63 Stuart P. "UNIX bell tolls for corporates", Computer Equipment, March 1990, p 17.

64 Singh I. M. "Will The Real Real-Time UNIX Please Stand Up?", Automation, April 1989, pp 34-36.

65 Anon. "IEEE Standard Portable Operating System for Computer Environments (IEEE Std. 1003), Institute of Electrical and Electronics Engineers Inc., New York, NY, 1989.

66 Adelin F. "Linking UNIX to a real-time operating system", Microsystem Design, September 1987, pp 10-11.

67 Corwin B. and Schermerhorn L. "/usr/group & P1003.4 Realtime Working Groups", Seventh Annual Control Eng. Conf., May 1988, Section IX, pp 24-29.

68 Fong K. L. "General Motors Real-Time Computer Systems Requirements Survey Report", Seventh Annual Control Eng. Conf., May 1988, Section IX, pp 16-23.

69 Anon. "System V Interface Definition", AT&T, Volumes 1 and 2, 1989.

70 Grey G. "Towards Multi-Processor UNIX", Microsystem Design, July-August 1989, pp 10-11.

71 Weingart J. "GM Demands UNIX in its Future Buys", Computer Systems News, April 1986, p 1.

72 Gillenson, M.L., "Databases Step by Step", John Wiley, New York, 1984, pp 194-196.

73 Palmer, L.B., "Using dBASE for DCS Configuration", Seventh Annual Control Eng. Conf., May 1988, Section XIX, pp 1-4.

74 Ruckman, R.P., "Data Base Requirements in Process Control Systems", Seventh Annual Control Eng. Conf., May 1988, Section XIX, pp 8-16.

75 Vail, P.S., "Computer Integrated Manufacture", Chapter 9: Design of the Data Base", PWS-KENT, 1988, pp 181-210.

76 Anon. ANSI/SQL, "ANSI American National Standards Standard Database Language SQL", New York: American Standards Institute Inc., 1986.

77 Anon., "Sun's Software Overview", BE125-0/75K, Sun Microsystems Inc., Mountain View, CA 94043, 1987.

78 Anon., "Microsoft Windows - User's Guide", Version 2, Microsoft Corporation, 1987.

79 Anon., "GEM/3 User's Guide", Digital Research, Monterey California 93942, December 1987.

80 Anon., "What is the MIT X-Window System?", Amarante, AIM Business Centre, Welham Green, Hertfordshire, 1990.

81 McCready, A. K., "User Interfacing to Process Computer Systems", Standard Handbook of Industrial Automation, Chapman and Hall, 1986, pp 424-432.

82 Dallimonti, R. "Challenge for the 80s: Making Man-Machine Interfaces More Effective", Control Engineering, January 1982, pp 22-60.

83 Anon., "A Guide to Microsystem Design", Syntel Microsystems, Huddersfield HD1 3PG.

84 Anon, "DSC-1 Intelligent Motion Controller", Quin Systems Ltd, Oaklands Park, Wokingham, Berkshire.

85 Anon, "DMC-200 Two-Axis Multibus DC Motor Controller", Galil Motion Control, Mountain View, CA 94043.

86 Anon,. "Programmable Position Controllers MES38", No. 5.6.191-1, Issue 1, January 1986, Norgren Martonair, Eastern Avenue, Lichfield, Staffordshire.

87 Anon., "Microware 68000 C Compiler", Microware Systems Corporation, Des Moines, Iowa 50322.

88 Dibble, P., "OS-9 Insights: An Advanced Programmers Guide to OS-9/68000", Microware Systems Corporation, Des Moines, Iowa, November 1988.

89 Anon., "The OS-9 Catalog", Microware Systems Corporation, Des Moines, Iowa, 2nd. Edition, 1989.

90 Prieto, F et al. "Information Management in Manufacturing", Proc. 5th CIM Europe Conference, May 1989, pp 389-398.

91 Anon, "INGRES/ESQL Reference Manual", INGRES Relational Technology, Alameda, California 94501, 1990.

7
Proof of Concept Implementation of UMC

7.1 INTRODUCTION

This chapter discusses a proof of concept implementation of the UMC methodology targeted at the control of modular distributed manipulators. An industrially representative evaluation environment has been constructed at the University. This consists of a reconfigurable modular machine for demonstrating printed circuit board handling.

The potential for the UMC methodology is seen as very broad and consideration has been given to the achievement of adequate real-time performance for a wide range of possible applications. The ability to accommodate change is key to the potential of UMC as a generalised approach to control. Although currently offering relatively modest features it is important to recognise the inherently extendible manner in which the methodology has been implemented. This represents a significant step towards proving the concept of using an "open" control architecture for manufacturing machines.

The operating system OS-9 provides the core services for the present implementation of the UMC Reference Architecture. See appendix A. The implementation consists of:

- a reconfigurable real-time target control system environment conforming to the UMC Reference Architecture, and
- development tools to enable the required UMC machines and associated applications tasks to be described and configured.

7.2 MANIPULATOR AND MACHINE CONTROL TERMINOLOGY

The UMC philosophy encourages the design of machines with optimised mechanical structure and without redundant degrees of freedom. The terminology introduced here relates to the implementation of configurable machines and their associated control systems and extends the set of definitions provided in chapter five.

7.2.1 Machine Configuration

Conventional Robot	Any robot (usually pedestal mounted) which is available only in one or at most a few configurations of serial kinematic chain.
Modular Robot	A robot having any number of axes all mechanically coupled in a serial kinematic chain and constructed from a family of modular elements
Modular Distributed Manipulator	A robotic device constructed from a family of modular elements or axes which are not necessarily all mechanically coupled. (i.e. A number of serial kinematic chains may exist).
Manufacturing Machine	Any automatic machine used in an automated production facility which obviously includes each of the above machine configurations.

7.2.2 Universal Machine Control

Machine	A UMC machine consists of a user definable set of concurrent tasks and interface handlers for real-time control.
Task	Tasks are user defined processes describing the application of a UMC machine.
Handler	A handler is an interface used to control and monitor a UMC machine related device. Two types of handlers, Axis Handlers and Port Handlers are currently implemented.
Axis	An axis is a programmably positioned mechanical system with one degree of freedom. Any axis forming part of a UMC machine is controlled via an Axis Handler.
Axis Group	UMC axes can be grouped together in order to achieve collective movement in many degrees of freedom (typically as part of a distributed manipulator). The axes of an axis group need not be mechanically coupled (i.e. may belong to separate kinematic chains). Theoretically there is no restriction on the number of axes in a group and any individual axis may be part of more than one axis group of a machine.
Port	A UMC port is any binary I/O device controlled via a port handler. Ports are divided into I/O channels. The configuration and number of channels is related to the specification of the I/O device which might typically be a PLC or intelligent I/O board.
Event	Events are user defined flags employed within UMC to coordinate the concurrent application tasks in a machine.

7.3 IMPLEMENTATION PHILOSOPHY

7.3.1 Introduction

UMC aims to provide enhanced machine controller functionality and reduced lifecycle costs through the adoption of an open approach.

During machine build UMC seeks to provide sufficient computational power to support the functional requirements of each machine to be controlled in an optimised manner. A goal is to offer an approach which can provide a cost effective solution to diverse control requirements and which does not compromise the control system enhancement.

7.3.2 Ensuring Adequate Real-Time Performance

The speed of response required to adequately cater for events which occur in real-time is an important fundamental consideration in control system design. It is also important to consider the relative amounts of code which are typically associated with the implementation of different controller functions. Whilst these factors are obviously application dependent and generalisations must always be considered with care, Lent [1] has considered the relationship between response requirements and code size for a typical machine controller used for discrete parts manufacture. See figure 7.1. In general terms the larger and less time critical functions require the most frequent and extensive modification since they are predominantly applications related. The more time critical code typically represents a relatively small percentage of the total code, which is less frequently changed. Many of these more time critical functions may therefore be conveniently embedded in dedicated device controllers, offering cost effective state of the art performance at the time of installation. However due to ad hoc implementation methods these devices generally restrict the future flexibility of control systems and hinder their reconfiguration. See section 3.4 for an assessment of current machine control methods.

As discussed conceptually in sections 5.6 and 5.7 UMC allows for the use of intelligent device controllers in a well defined manner through the use of *device handlers* at the boundary of the UMC environment.

7.3.3 External Device Handlers

In the purest environment the device controller interface could be standard by design. However practicality dictates that a diversity of controller implementations must be accommodated at some level.

The lowest level in the UMC hierarchy is therefore formed from external device interfaces termed *handlers*, as illustrated in figure 7.2.

When using currently available device technology, control methods are invariably device specific. Typical examples are, motion and intelligent I/O controllers which support a wide variety of unstandardised programming methods, communication protocols and computer interfaces [2]. See figures 7.3 and 7.4 also section 3.3.6. The UMC handler enables the utilisation of these devices, which are seen as vital components in state of the art machine control systems, in a consistent manner without restricting their performance.

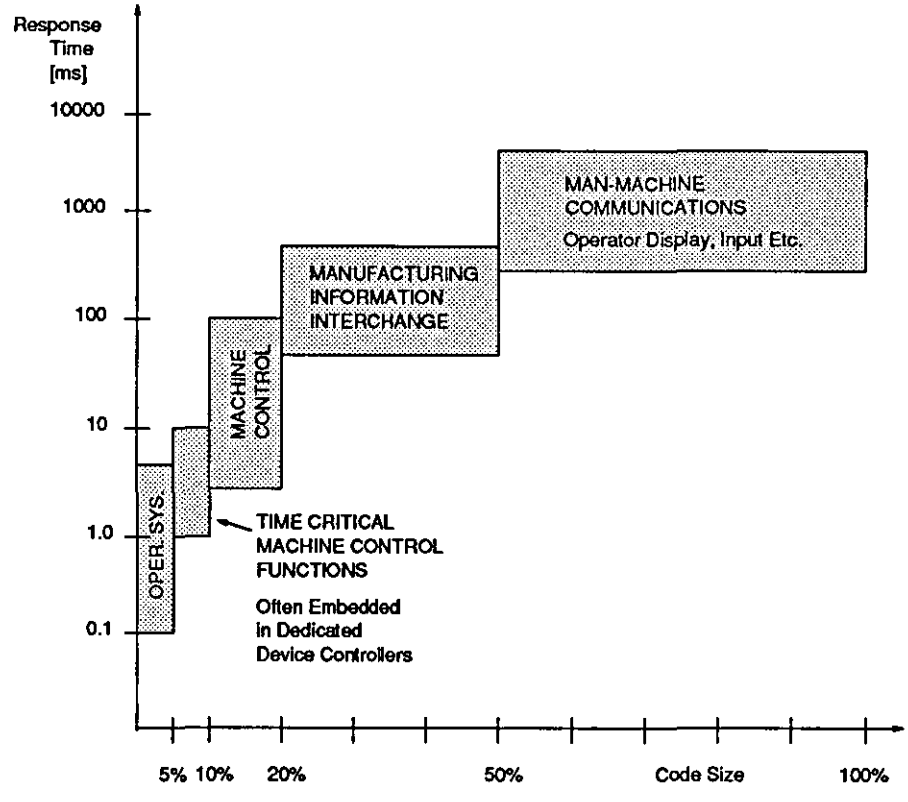


Figure 7.1. Typical relationship between response requirements and code size for a machine controller.

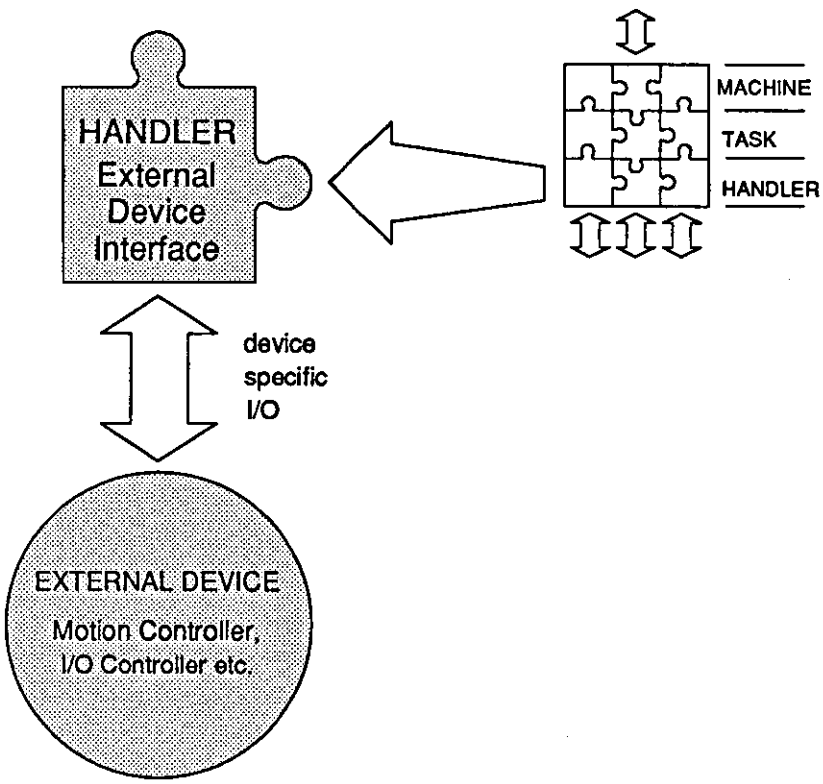


Figure 7.2. The role of device handlers within UMC.

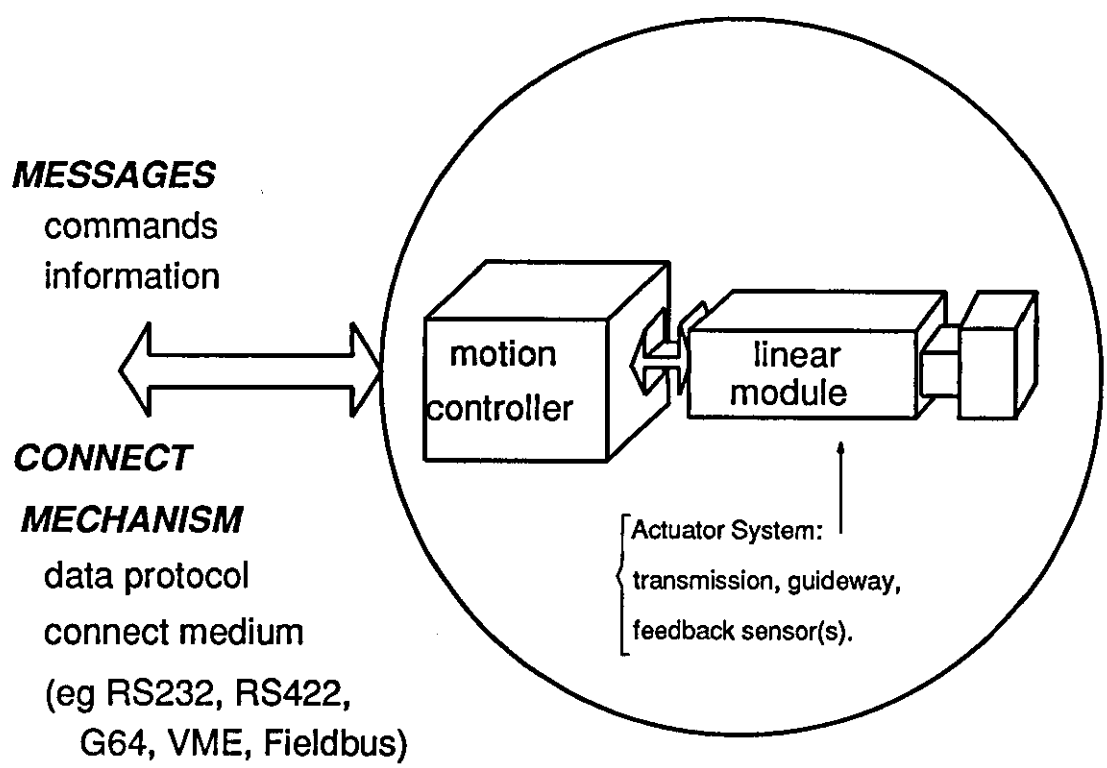


Figure 7.3. Motion control device interaction.

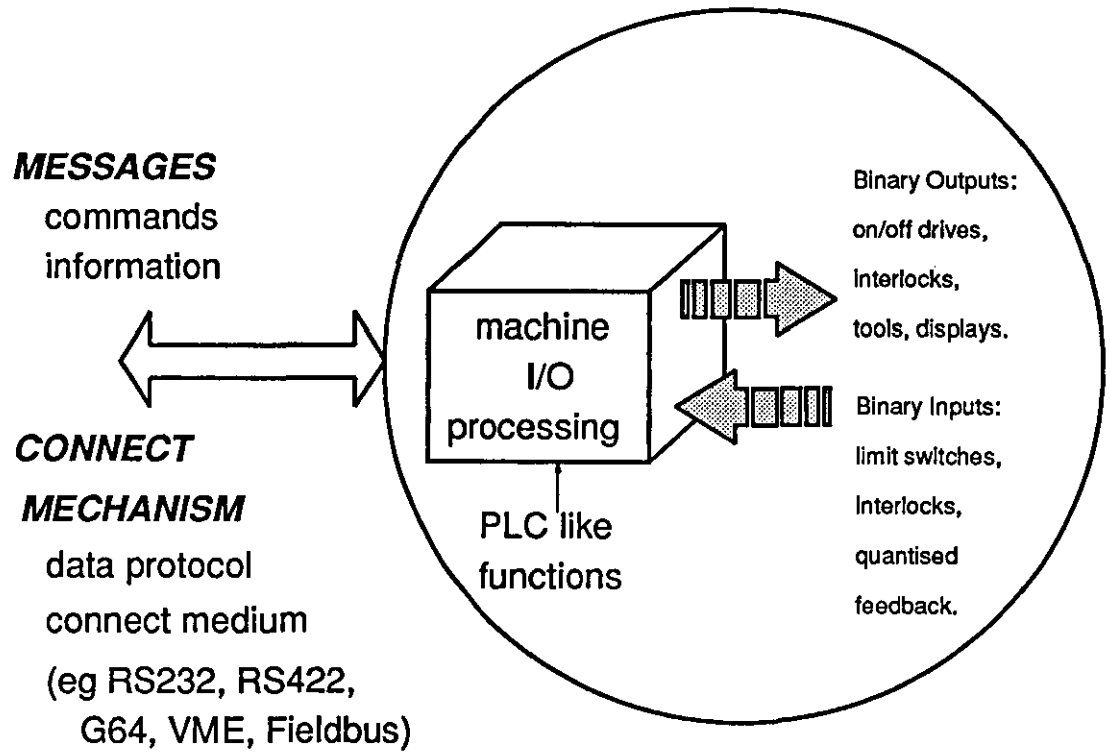


Figure 7.4. Intelligent I/O control device interaction.

Handlers are required to provide a well defined mechanism for making external devices appear consistent within the UMC system. An appropriate interface needs to be provided for each type of device so that both old and new control components of widely varying complexity can be integrated together and programmed in an efficient manner. Compatibility with current systems and vendors' future systems is the dominant feature required by users [3]. Handlers unify the logical appearance of each type of control device, enabling consistent command sets to be used for all axes of motion or all binary I/O ports. A given device may obviously only be able to support a subset of the available UMC control commands depending on its capabilities [4]. See section 7.8. Each handler describes the control capabilities, hardware interface and communication protocols for an individual external device. Once created a given handler can be reused each time that device recurs on the same or subsequent machines.

7.3.4 Determining the System Boundary

It is important to consider at the outset of any control system implementation the position of the system boundary. An appropriate system boundary must be

chosen. The position of this system boundary, formed by the UMC handlers, will typically vary with changes in both application requirements (speed of response etc.) and the capabilities of the available enabling technology (processing power etc.).

The UMC architecture neither prescribes nor precludes that different parts of a given application are implemented internally or externally to the UMC system boundary as illustrated in figure 7.5. The potential also exists for more than one UMC machine to be executed within a single operating system environment [5]. See figure 7.6.

The performance capabilities of any selected processing hardware and associated operating system are obviously finite. The combined performance capabilities of a particular UMC machine controller will depend on the processing power of the UMC system itself together with the capabilities of intelligent device controllers outside the UMC system boundary. As suggested in figure 7.5 there may be many acceptable combinations of processing elements and their selection requires careful consideration.

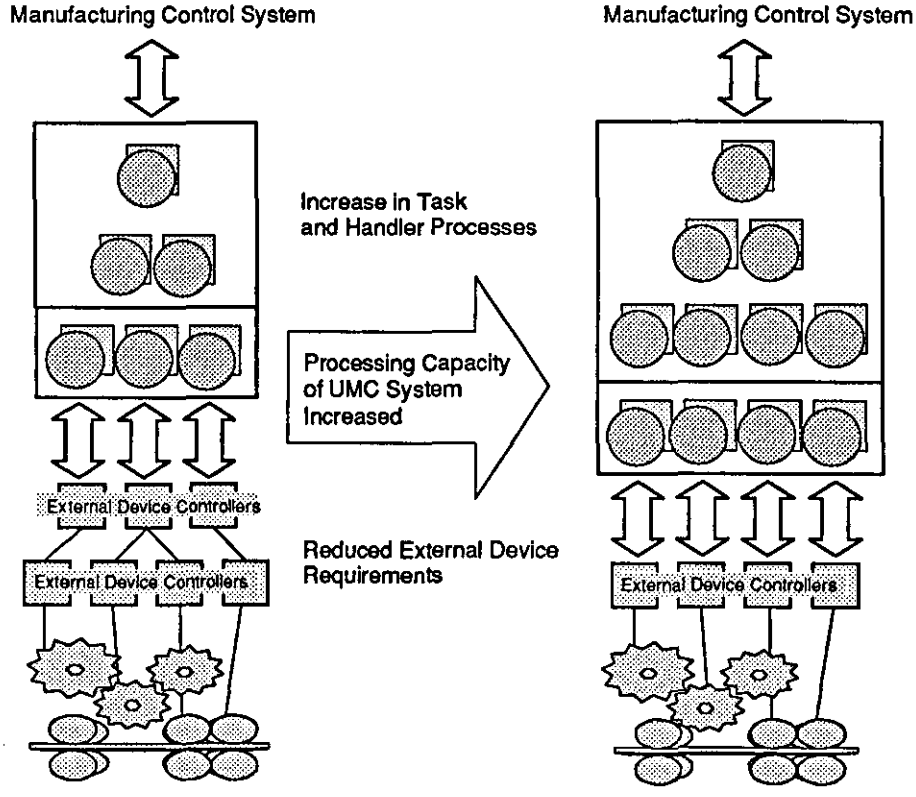


Figure 7.5. Variation in UMC system boundary with changes in processing power.

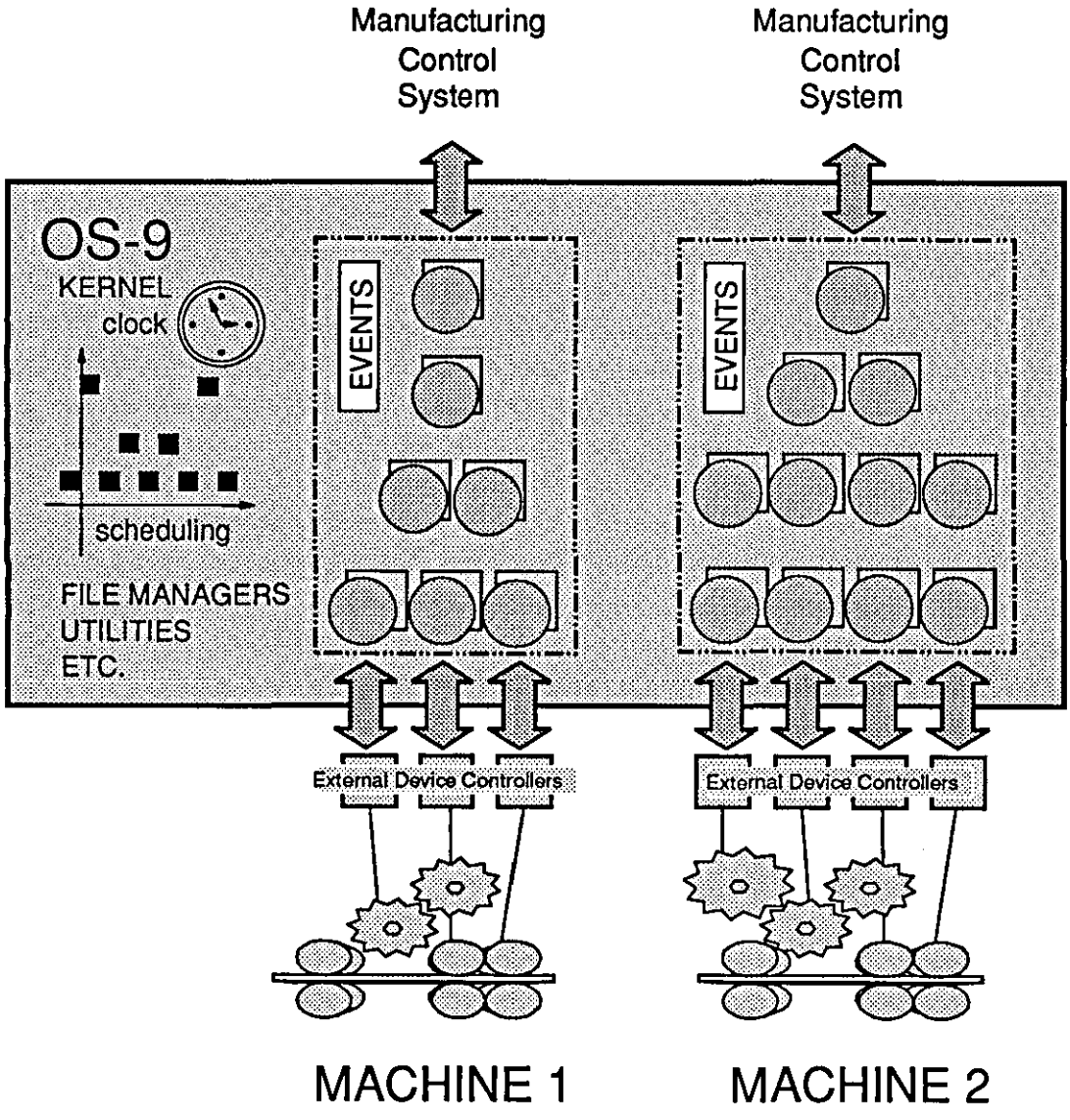


Figure 7.6. Multiple UMC machines in a single operating system environment.

7.3.5 Modular Distributed Manipulators

7.3.5.1 Introduction

A major advantage sighted for the use of robotic manipulators for automation in preference to task-specific hardware is their flexibility [6]. In theory, a robot's task can be changed simply by loading a new program into its controller. However in practice this is rarely the case. Each conventional robot has a specific configuration that supports a limited range of capabilities, appropriate only to the particular application for which it was designed. The concept of modular distributed manipulators is to create robots with different optimised configurations for each manufacturing task based on specific task requirements.

The design and application of modular distributed manipulators and the requirements for their effective control both at axis and supervisory levels have been the focus of research by the Modular Systems Group at Loughborough for many years [7], [8], [9], [10]. The inherent concepts of configurablity and machine optimisation within UMC stem from the requirements for the control of modular distributed manipulator systems and the provision of machines capable of providing an inherently flexible and responsive approach to manufacturing. See also section 2.5 and the paper in appendix B.1 which reviews a study of application areas for modular robots undertaken by the author.

The chosen application for the initial proof of concept implementation of UMC was a distributed manipulator system for printed circuit board handling and assembly illustrated in figure 7.7. The facility which uses industrial hardware wherever possible, has been configured to provide a realistic environment for practical control system evaluation.

The system consists of modular actuators and tooling to transport printed circuit boards between pallets and work piece fixtures. The demonstrator extends the concept of modularity throughout the machine's mechanical hardware and control systems. The mechanical modules are mounted on a modular extruded section aluminium framework which is also reconfigurable.

The modular manipulator system can be summarised as follows:

Conveyor System	Three electric conveyors with transfer cylinders for pallet handling.

Pallet Transportation	Pneumatic servo actuated rodless cylinder with pallet/carrier location tooling.
PCB Handling	Pneumatic servo activated gantry module with electric servo activated vertical module and programmable PCB gripper.
PCB Registration	Electric servo actuated slideway and two axis stepper motor positioned PCB registration fixture.
Component Insertion	Four electric servo actuated modules with interchangeable tooling.
Pick and Place	Two servo actuated modules and tooling for reject PCBs.

The paper in appendix B.2 reviews the application of the demonstrator system.

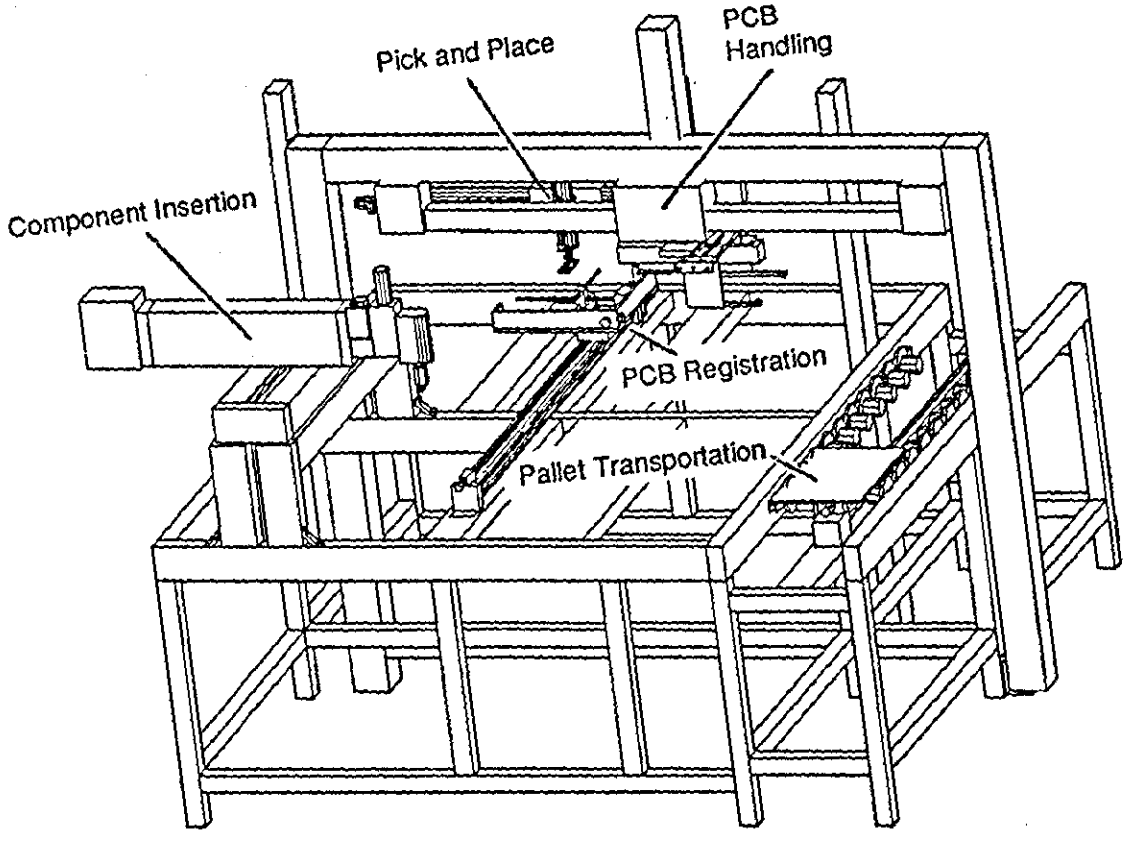


Figure 7.7. LUT distributed manipulator demonstrator.

7.3.5.2 Demonstrator Control System Configuration

A typical target system hardware configuration is illustrated in figure 7.8. The system comprises of a single supervisory processor which supports the UMC environment, together with a set of intelligent slave device controllers. A review of the selected development and target system hardware is provided in section 6.14.

The position of the system boundary follows generally accepted practice for high performance multi-axis motion control systems. The control of binary I/O *ports* and *axes* of motion is assigned to dedicated processing hardware controlled via UMC *handlers* with the remainder of the application programmed as a series of concurrent *tasks* within the UMC environment. See figure 7.15.

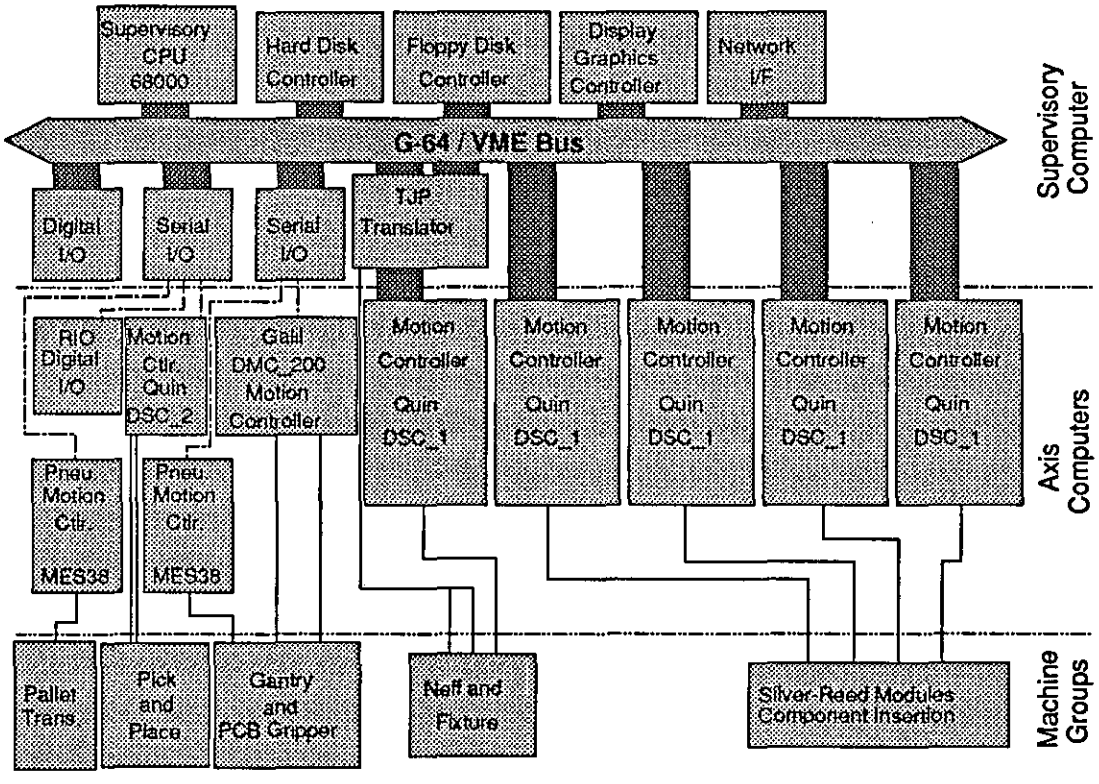


Figure 7.8. Schematic of system hardware.

The UMC device handlers are interrupt driven and the devices do not require polling. The UMC system thus executes in a predominantly event driven manner within the system boundary. Each process may typically spend a large percentage of its time waiting. The handlers initiate and monitor the activity of continuous or discontinuous processes beneath them implemented outside the UMC system boundary.

7.3.5.3 User Defined Machine Structure

The distributed manipulator illustrated in figure 7.7 is divided into a series of user defined *axis groups* with associated I/O channels. Each axis group is programmed and controlled from a separate task process and the movements of the axis group are performed between named locations. It should be noted that axes may be shared between tasks to simplify the programming of applications. Figure 7.9 illustrates the axis groups typically used for printed circuit board handling applications at Loughborough. These groupings are however user defined and can be changed as required. See section 7.7 for details of task implementation.

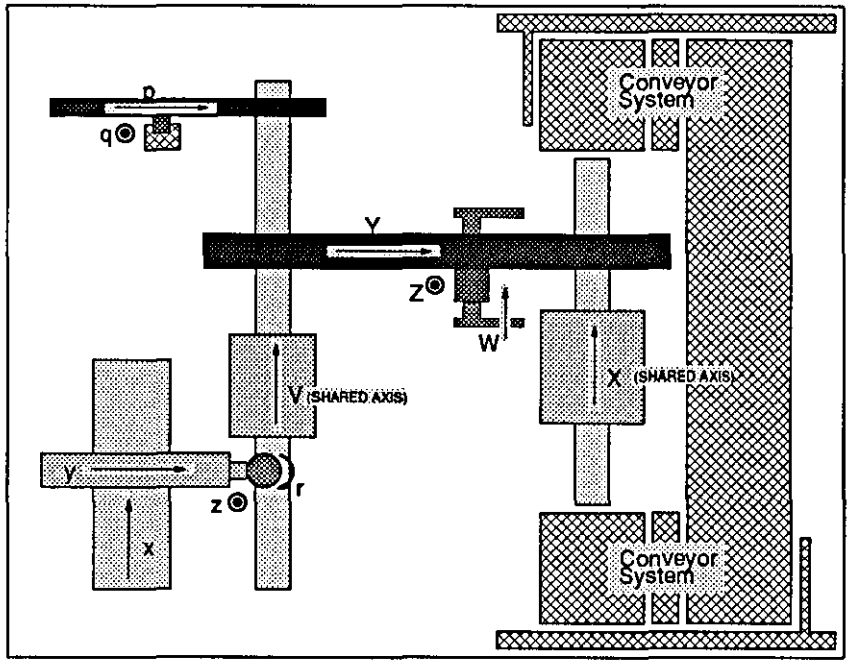


Figure 7.9. Division of the LUT distributed manipulator into axis groups.

7.4 IMPLEMENTATION OVERVIEW

7.4.1 Introduction

The UMC architecture described in chapter five consists of a number of hierarchical layers. Each UMC machine is essentially a set of components (concurrent

processes and associated information modules) which are created in a defined manner on the target computer. A UMC machine is created on the target computer by a *machine loader program* which interprets the information contained in the machine information module. The design and implementation of the machine loader is described in detail in chapter eight.

The UMC Machine information module provides an overall machine controller definition and is the root for creation of or references to all other program and information modules which make up a complete UMC controller. See figure 7.10.

7.4.2 UMC Machine Development Cycle

As discused in chapter five, (see in particular figures 5.1 and 5.2), the UMC environment has been designed and implemented to enable applications to be developed in stages allowing progressive testing and verification. The task program modules forming an application can be written and tested individually and then combined together in a very flexible manner. The state of events and inputs can be manipulated via user utilities and simulated handlers can be used in place of physical interface devices and their associated external hardware. See section 7.8.

At *run time* a UMC control system is composed of a set of memory resident modules which each contain either programs or information. A corresponding set of information and program files are used for system definition prior to run time. These *files* of defined types form a library of machine, task, axis and port components. See figure 7.11.

The user is required to edit a machine information module edit file in order to describe the structure of the machine he wishes to create. Device handlers and applications tasks then need to be written and selected as necessary. These describe the devices to be controlled and the concurrent operations the machine is required to perform. There will obviously be an iterative cycle in order to extend and refine the machine description and task programs as a given application develops. Sub-sets of a given machine can be selected for evaluation and test at any time using options provided by the machine loader. See chapter eight.

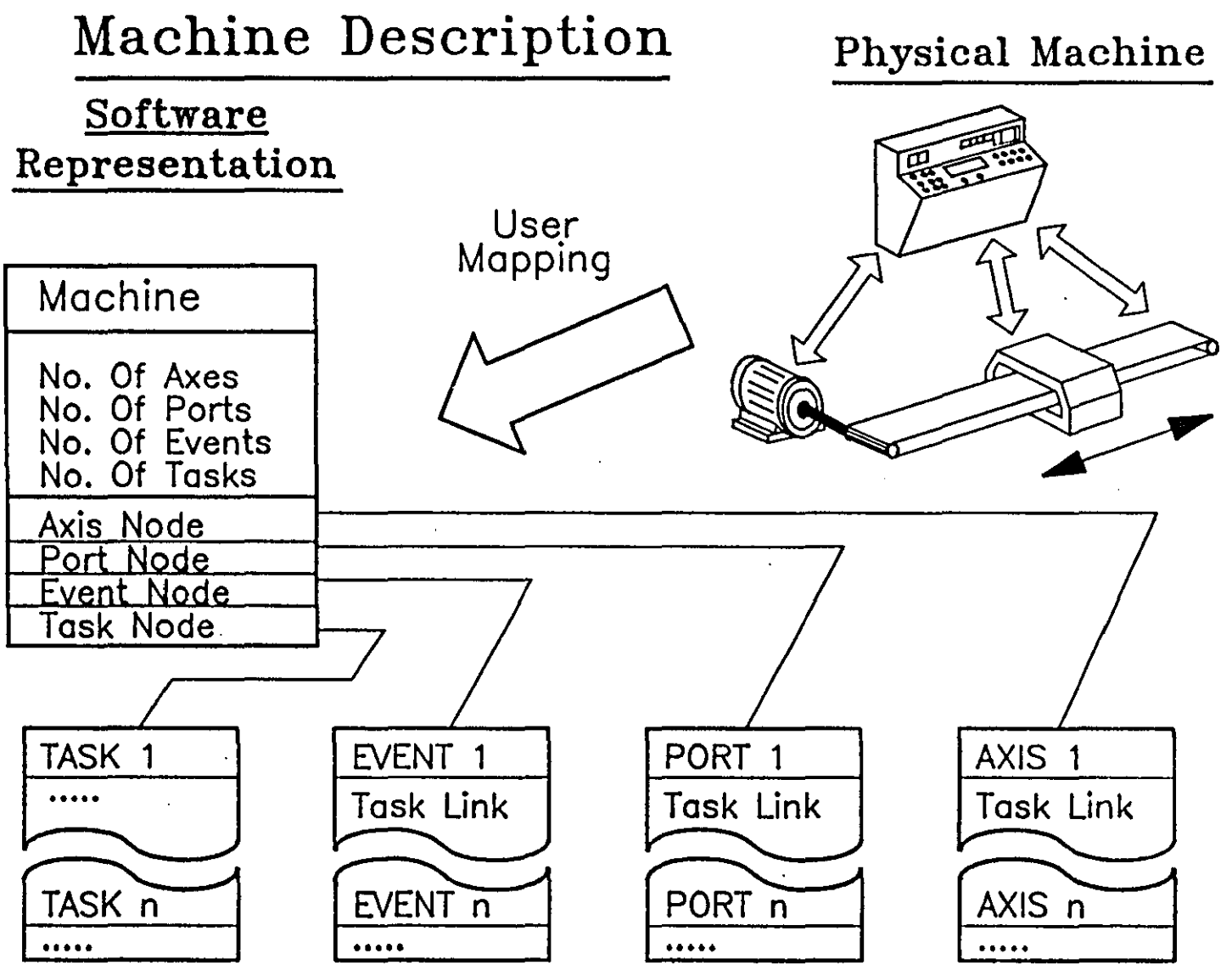


Figure 7.10. Schematic of machine information module structure.

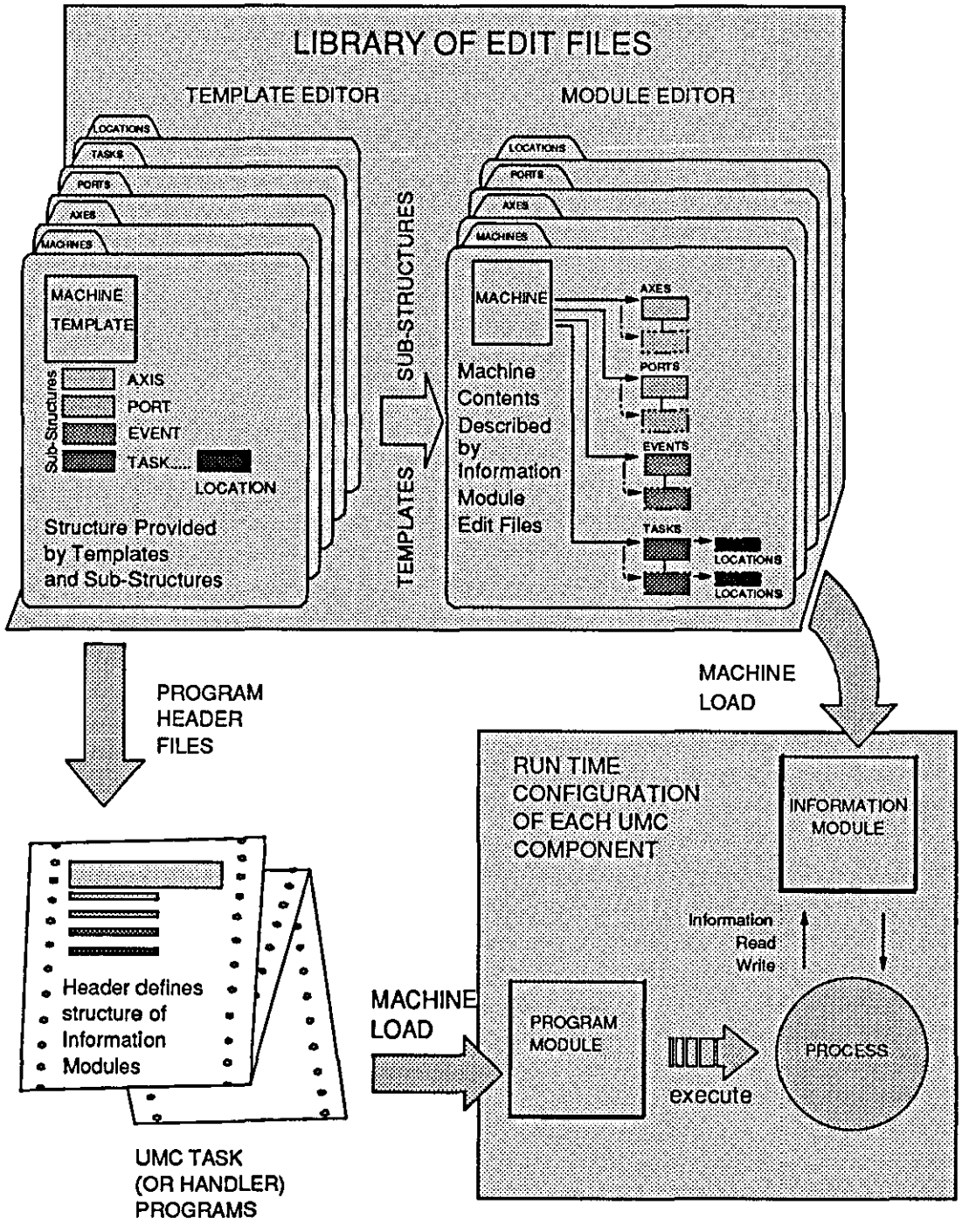


Figure 7.11. Schematic of UMC development stages.

Note 1: Location Modules are referenced via the Machine Module - Task Sub-Structure.
Note 2: Task Modules are normally created dynamically by the machine loader (ml) and not via the module editor.

7.5 COMPONENTS OF THE UMC ARCHITECTURE

7.5.1 Introduction

In simplistic terms the components of the UMC Architecture each consist of a process and one or more uniquely associated information modules of a standard format determined by *template(s)*. See figure 7.11. The lower levels of control may be distributed on external hardware via handlers.

OS-9 memory modules are used to implement the building blocks of any UMC controller. Appropriate sets of these modules may contain programs or data to support the required functions for each UMC component. Appendix A.2 provides a more detailed description of OS-9 memory modules.

7.5.2 Program Modules

Each UMC process executes the code contained in an OS-9 program module. See appendix A.2. Program modules are created using conventional OS-9 editors, compilers, interpreters and utilities as appropriate [11].

UMC Handlers, tasks, utilities, editors etc. are all loaded as program modules prior to their execution. Program modules are re-entrant and can therefore, if appropriate, be shared by more than one process on a machine.

7.5.3 Information Modules

The speed requirements for real-time control do not allow the use of disc files for interprocess communication or information retrieval. At run time a UMC machine controller therefore uses information modules held in memory to allow the buffered exchange of information between the processes forming the control system. See appendix A.7.1 which describes OS-9 data modules.

Currently five types of UMC information module are defined and these are used in various numbers and combinations to support the information requirements of a given machine controller. The module types are machine, task, task location, axis handler and port handler.

7.5.4 Information Module Structure

Each type of UMC information module has a predefined revisable structure composed of fields which contain data in one of a number of formats. Currently the field types supported are characters (one byte), integers (four bytes), floats (four bytes) and thirty two character strings (thirty two bytes) [12].

The format of each type of UMC information module is defined by a template with associated sub-structures for repeated sets of elements. Figure 7.11 illustrates the templates and sub-structures associated with a machine information module. These templates and substructures are editable and this importantly enables the UMC information structures to be modified in a defined manner. A template editor, Booth [12], generates the templates and substructures which define the *data structure* for each type of UMC information module. Module editors support the creation and maintenance of the information module contents. Matching header files, also generated by the template editor enable information access from any UMC process to each type of UMC information module. An appropriate set of header files must be included in each UMC task or handler program depending on the types of information module to be accessed. Appendix D lists the headers required for machine information module access.

7.5.5 UMC Components

Any UMC machine controller will be composed of a single machine information module and a variable number of other modules. Figure 7.12 illustrates the module combinations typically used to form the run time components of UMC controllers. The run time machine controller uses these components in a simple structure composed of three layers, Machine, Task and Handler.

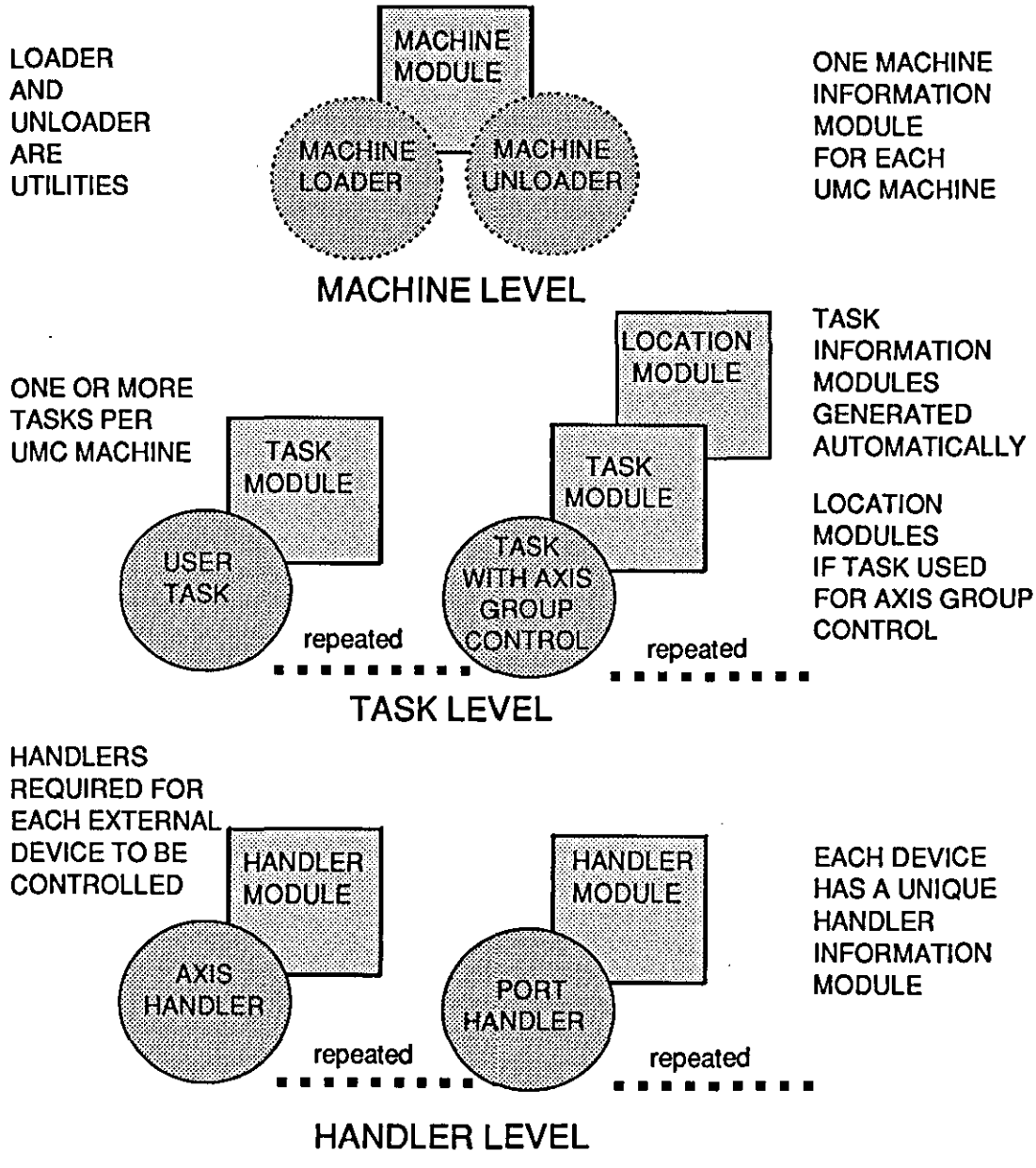


Figure 7.12. Module combinations used to form the components of a UMC run time system.

7.6 DESCRIBING A UMC MACHINE

7.6.1 Introduction

As explained in the Implementation Overview, section 7.4.2, it is important to appreciate that a set of memory modules are used at UMC run time while an equivalent set of files are used for machine and process description.

UMC machines are currently described by a library of edit files manipulated via a set of custom written editors [5]. There is current research by the Modular System Group to enable the use of an emerging industry standard data base management system for UMC machine description and storage in the future. See section 6.12.

The UMC file library is used in conjunction with machine loading and unloading utilities to enable control system retrieval at load time and saving after use. Figures 7.11 and 7.13 illustrate UMC machine description and retrieval schematically. The information structure is predefined by template files. The contents of the information structure is in turn described by edit files.

The machine information module (or its corresponding edit file) lies at the root of the machine controller structure. It defines the combination of other modules which a particular "instance" of the UMC machine requires. These modules collectively describe each axis or I/O handler, applications tasks and their I/O paths, axis location data, global events of significance and the logical relationship between tasks, axis and port handlers, and events.

7.6.2 Associating Machine Entities with Tasks

A defined subset of a machine's *axes*, *ports* and *events* can be associated with *each task*. The required associations are defined in the machine module edit file. See figure 7.14. Each task is allocated a unique *task link bit* in the task link bit field. The corresponding bit is then set in the *task link field* of every UMC entity (ie. axis, port, event etc.) to be associated with that task.

The task link bits are interpreted by the machine loader at run time to create the necessary machine structure. See chapter eight. A task cannot reference any UMC entity which it has not been associated with via link bits. See section 7.7.4.1.

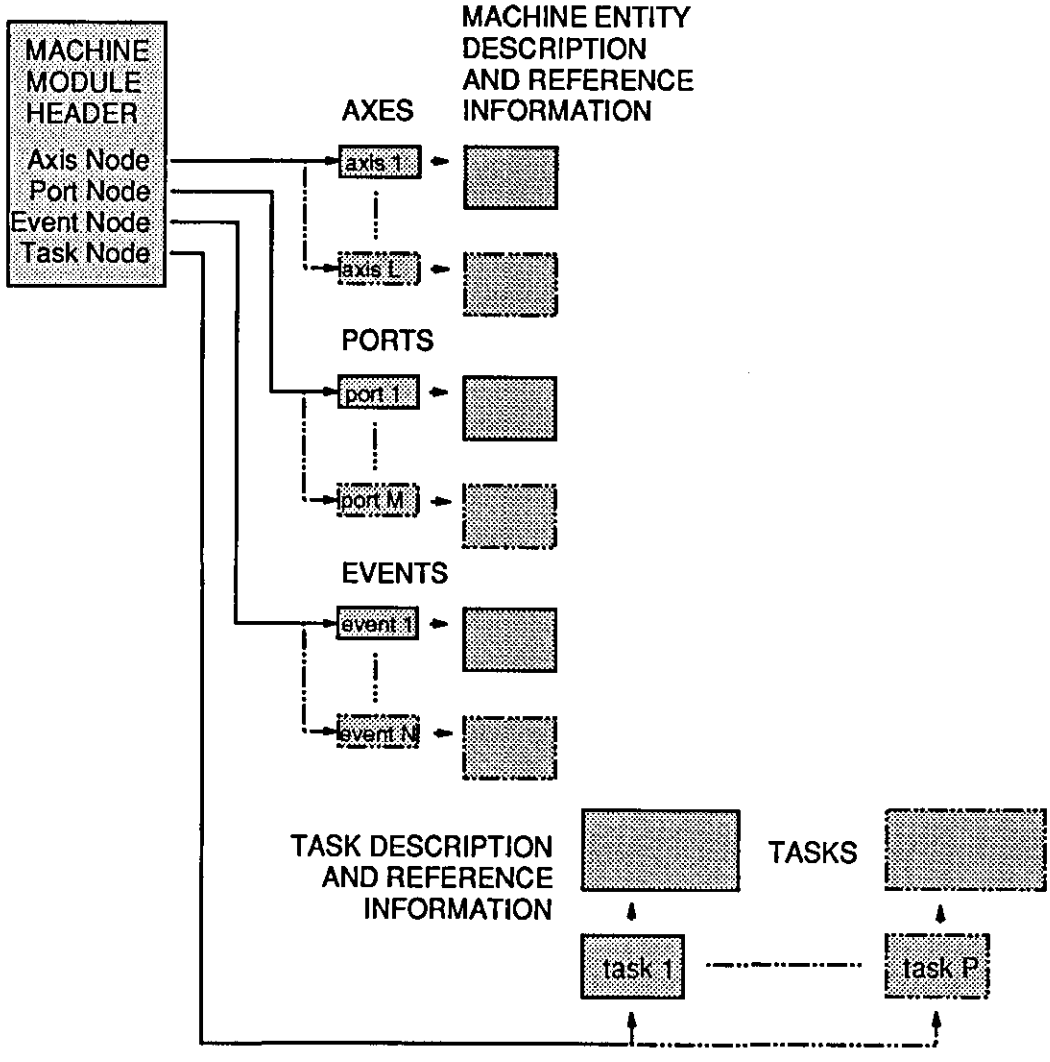


Figure 7.13. Schematic of machine module structure.

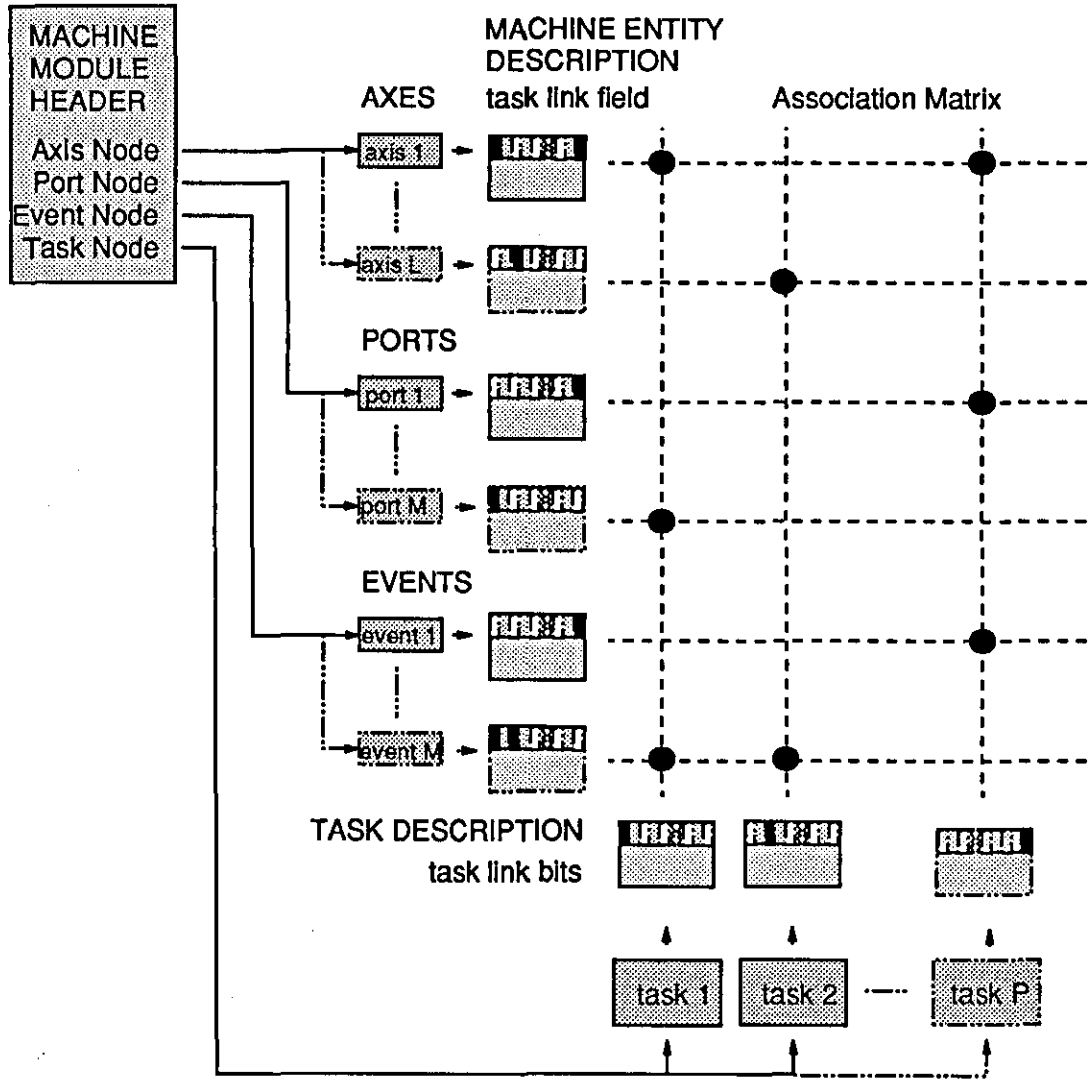


Figure 7.14. Concept of task link bits for associating tasks with UMC entities.

7.6.3 Information Module Editors

Editors allow both the "off line" creation and modification of the disk based edit files and the "on line" modification of information modules contents at machine run time [5]. These editors fulfil the following functions:

Template Editor for templates and sub-structures,

Module Editor for machine, and device handler module files,

Task Location Editor for the on line editing of location module files, and

Parameter Tuning Editor to support the interactive optimisation of control loop gains (in axis information modules) which usually contain values specific to the mechanism being controlled.

7.7 UMC TASK IMPLEMENTATION

7.7.1 Introduction

The *task structure is entirely user defined.* The number and type of tasks are defined by entries in the machine information module.

The most common and obvious purpose for tasks is the control of mechanical and electrical hardware via handlers. Tasks can also control other task programs, extract information from the system and report it, provide facilities for recovery from or control of failure situations etc. The multi-tasking nature of the system allows control, diagnostics and monitoring logic to be naturally separated. This allows the visualisation and programming of complex applications to be simplified. Figure 7.15 illustrates the division of the Loughborough PCB handling application into multiple tasks.

Due to the predominantly event driven nature of UMC, task programs need not be a great overhead on the processing power of the system. Typically most processes will be in a sleeping state when action is not required and will be awakened by events or signals generated by other tasks or handlers when their action is required. See appendix A.6, OS-9 process management.

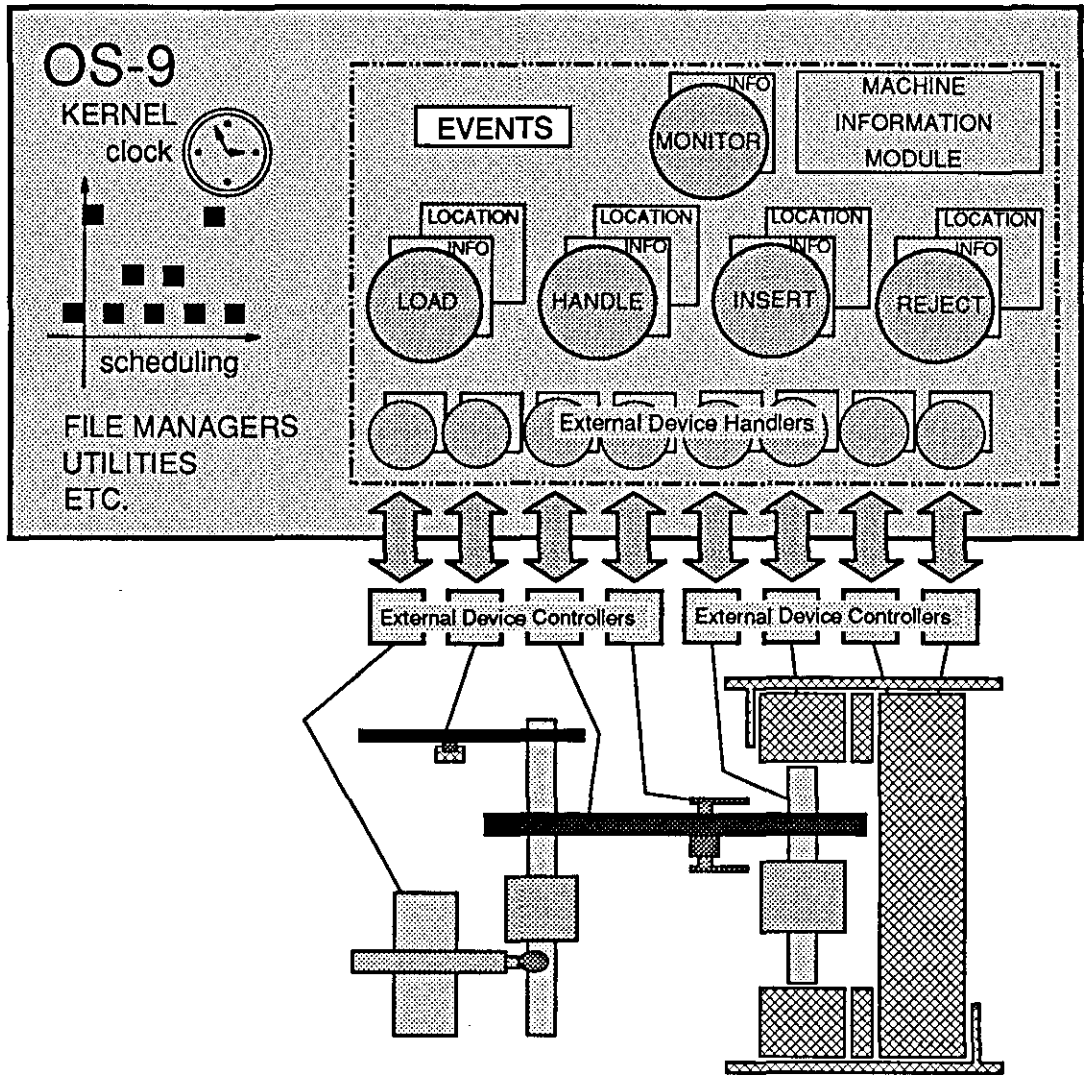


Figure 7.15. Schematic of typical task structure for the PCB handling application.

7.7.2 Task Information Modules

Task information modules provide applications task related information. Currently task information modules are by default created automatically by the machine loader and contain only a small amount of data. An optional *location module* can also be associated with each task (to store position data for axis group motion control) as required by the application.

7.7.3 Task Program Modules

Task programming is currently accomplished in either Microware Basic or C using standard system editors to generate the program code with appropriate compilers and interpreters as required [13], [14]. Appendix C contains the listing of an example task program written in Basic. UMC Release 1 provides extensive examples of C task programs [5].

Task programs can in principle be generated in any high or low level language allowing the most appropriate programming method to be selected for each task. It is however necessary for "hooks" into the UMC system to be provided for a given language to be fully integrated into the UMC system's facilities. These "hooks" are provided in the form of subroutine or function libraries.

The C program function libraries contain relocatable code which can be linked to the task programs. The same C source functions are used to create Basic subroutines modules which can be called from a task program written in Basic. See chapter eight.

7.7.4 UMC Library Functions for Tasks

There are currently several C function libraries and a more limited set of Basic subroutines which may be called by task programs in order to access UMC system facilities. These function libraries cover handler communication, linking functions, information module usage, event usage etc. [5].

7.7.4.1 Linking

UMC tasks need to reference various UMC entities, (events, axes, ports, locations etc.) for control purposes. These entities are initially referenced via mean-

ingful names using link functions. This serves to ensure that the entity referenced does exist and is associated with the particular task. The link functions return numbers which are used thereafter by the task for efficient access or control of that entity. See example program in appendix C.

7.7.4.2 Event Usage

Significant events which occur globally are managed by means of an event system. Events of interest can be given meaningful names and after linking their values are then manipulated from UMC task processes. UMC events are typically used for task coordination and synchronisation [15]. Figure 7.16 illustrates this use of events with extracts from Basic task programs.

Events are also be used to provide the capability for tasks to wait for a change of state in a binary input or for an analogue input to reach a set value. This feature is incorporated within the handler programs and is implemented in a manner which is transparent to the user. It is possible to wait for all inputs to the UMC system in this manner which enables the I/O system to be integrated into task programs in a very natural manner.

7.7.4.3 Handler Usage

Device handlers provide a device independent interface to task processes within UMC via a neutral command set appropriate for each supported class of device. Thus device communication is achieved via a *defined* but *extendible* library of instructions which are common to all implemented handlers of a given type. Axis and binary I/O port handlers have currently been implemented. See section 7.8. The program extracts shown in figure 7.16 include handler motion command calls. "Smove()" is a multi-axis straight line move to a named location.

```
PROCEDURE task_1

  RUN look(group1)
  IF group1.ev_value=HALT THEN
    PRINT "task_1 - Waiting for event 'group1'"
    RUN wait(group1,1,2)
  ELSE
    PRINT "task_1 - Starting SYNC"
    REM sync moves in both tasks using events:
    RUN signal(start2)
    RUN smove(position2,FAST)
    RUN wait(finish2)
    RUN signal(start2)
    RUN smove(position3,FAST)
    RUN wait(finish2)
    RUN signal(start2)
    RUN smove(position2,FAST)
     RUN wait(finish2)
   ENDIF
 ENDLOOP
ENDLOOP
```

```
PROCEDURE task_2

  REM sync moves in both tasks using events:
  RUN wait(start2)
  RUN smove(position3,FAST)
  RUN signal(finish2)
  RUN wait(start2)
  RUN smove(position1,FAST)
  RUN signal(finish2)
  RUN wait(start2)
  RUN smove(position4,FAST)
  RUN signal(finish2)
 ENDIF
ENDLOOP
ENDLOOP
```

Figure 7.16. Task program interaction using events.

7.7.4.4 Information Module Usage

As explained in section 7.5.3, information modules provide repositories for information exchange between different levels in the control architecture. Functions have been written by Booth [12] to allow the reading and writing of individual fields within the information modules. The read and write functions perform data type checking and also contain code to prevent possible information corruption due to the simultaneous reading and writing of data by different processes. This is achieved by the use of binary events which control access to each information module.

7.7.5 Task to Operator Communications

Due to the configurable nature of the UMC control system it is necessary to provide more flexible methods for user interfacing than those normally offered by conventional controllers.

Traditional controllers are largely predefined in their form and usually have user interfaces of fixed configuration. See section 3.4. In the UMC system the user interface configuration is defined in software and has a hardware independent representation.

In common with UNIX each OS-9 process used within UMC has three standard I/O paths, standard input, standard output and standard error [16]. These paths are all by default directed to the interface device on which the parent forking process is running but it is possible to redirect them. The redirection of these interface paths are defined for each UMC task in the machine information module. The *UMC machine loader* program includes code to redirect the standard paths to any appropriate I/O device specified in the machine information module. See chapter eight.

The capabilities of user interface devices have progressively improved during the duration of the research. Initially only simple serial devices, chiefly terminals, were available for user I/O paths. Terminals do not however lend themselves to being shared by multiple processes, particularly with respect to input. It was necessary to employ a terminal for each task during system testing which was not an elegant solution particularly when a large number of tasks were running. Windowing systems now offer a more elegant solution to this problem. Each task or handler can have its own window supporting its own set of standard paths with the operator being able to select the desired window as required. IBM PC and Sun

workstation windowing systems are currently being evaluated for this purpose [17], [18]. The Sun solution is achieved by means of a network bridge between the OS-9 system running the UMC software and the Sun UNIX based environment [19].

7.8 UMC HANDLER IMPLEMENTATION

The implementation of handlers at LUT has been researched extensively by Booth [5] and others [4], [20]. This section seeks to provide only a brief overview of handler implementation.

The handler modules are concerned with the machine specific I/O devices for example, axes of motion, analogue or digital inputs or outputs etc. These may be of virtually any type and degree of intelligence.

The handler programs are device specific but may be shared if multiple devices of the same hardware type are used. A handler program is made up of three main parts, code common to all handlers, code common to handlers of a specific type and, code specific to a particular hardware device. See figure 7.17.

Since OS-9 uses re-entrant code a single handler program and a single device driver can be shared by all handler processes of the same type. Each of these processes will have a unique information module associated with it that will contain the device specific information including a unique device descriptor for each physical hardware interface. See appendix A.4, OS-9 I/O management.

As illustrated in figure 7.18 each handler information module defines the physical parameters of the external device (e.g. length, maximum speed, maximum acceleration for an axis of motion). It also holds transient information to support handler to task communication at run time.

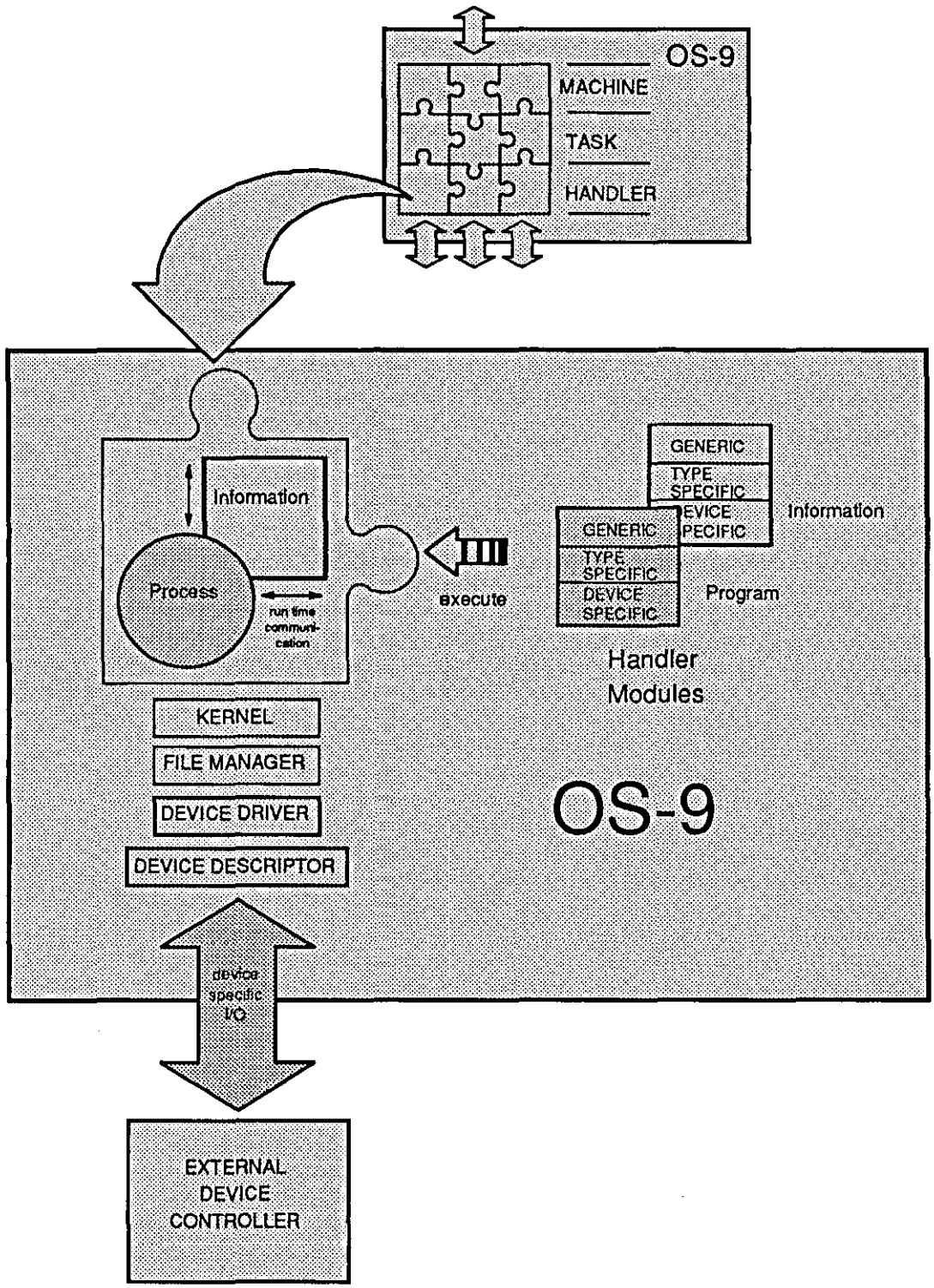


Figure 7.17. Handler structure and implementation within OS-9.

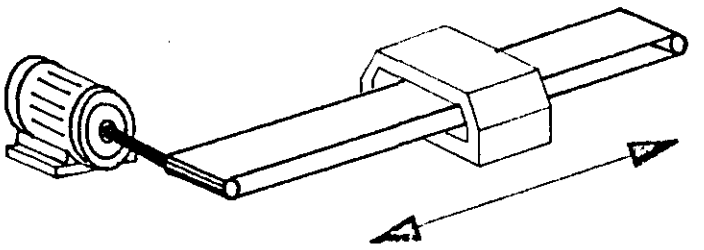
Component Description

– Axis Of Motion

Software Representation

Axis
.....
.....
.....
User Units
Max Positions
Max Velocity
Max Acceleration
.....
.....
.....
.....
Control Parameters
.....
.....

Physical Component



User Mapping

Figure 7.18. Handler information module schematic.

The use of handlers makes UMC applications tasks independent of the device hardware. The hardware thus becomes virtual and can be changed or reconfigured at will. If real hardware is not available during system design it is a simple matter to substitute handlers which simulate the real hardware for system development allowing extensive offline debugging.

7.9 UMC MACHINE RUN TIME

7.9.1 Introduction

The run time system includes software to load, run, modify and unload machines in whole or part. During system development it also allows the teaching of axis locations, the tuning of drive systems, the control of events, and the simulation of axis and port I/O devices [5].

7.9.2 Machine Loading and Unloading

At run time the machine loader must initially be executed as the first UMC process on the target machine controller. This utility program requires a machine information file name as an argument and creates the run-time controller as defined by the machine information module formed. See figure 7.19. The loader creates the specified handler and task modules and associated events and initiates all the required processes. Having done this the loader currently dies. The operation of the loader is detailed in chapter eight.

The task processes link to the UMC entities (currently axes, ports, events and locations) they wish to use and call UMC control functions or subroutines as appropriate. See section 7.7.4. An unloader utility allows the modules associated with a UMC machine to be partially or fully removed in a consistent manner.

In the future the machine loader, modification utilities and unloader may be integrated together to provide a consistent interface for communications with other machines and with high levels in the manufacturing systems architecture. See chapter nine.

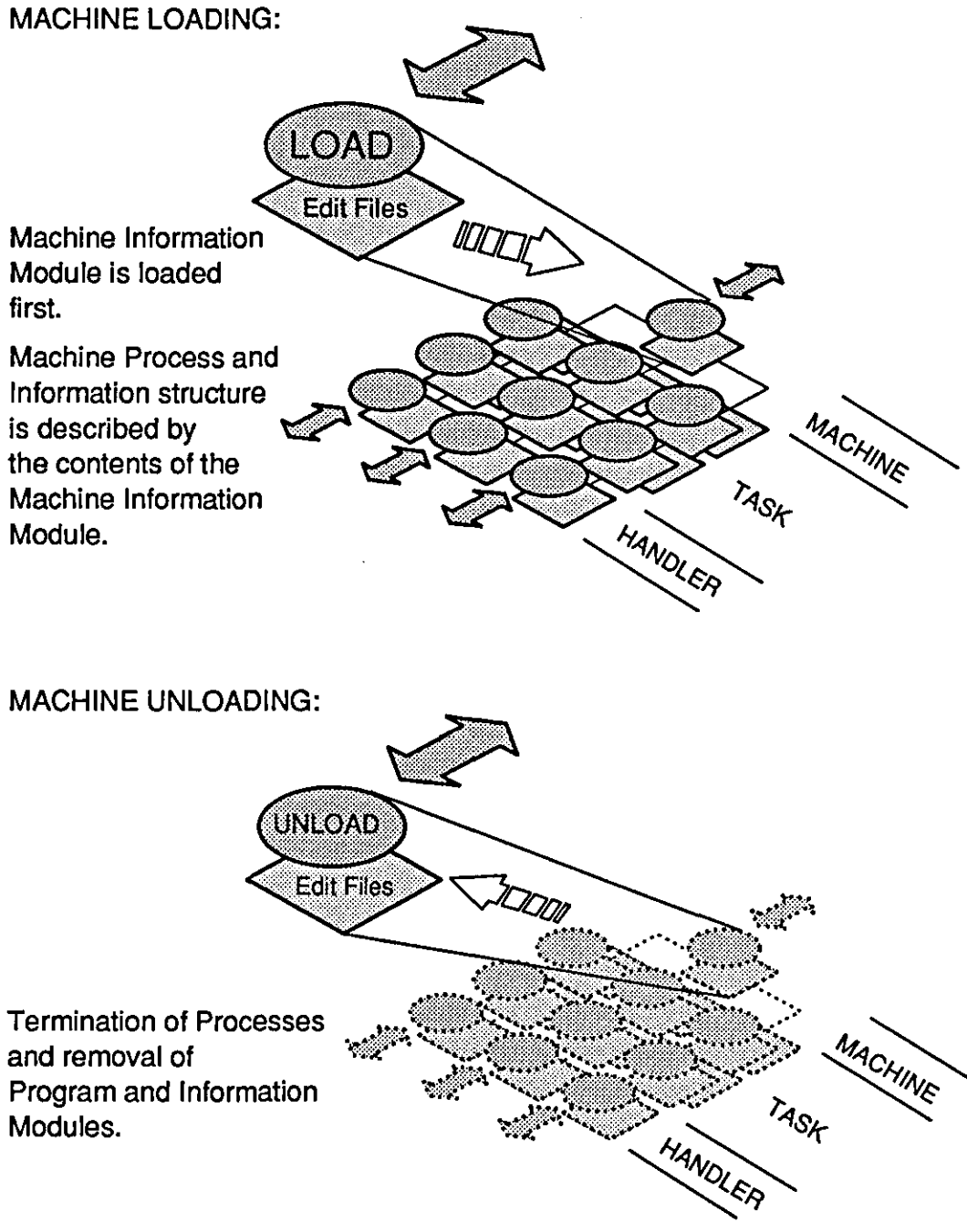


Figure 7.19. Concepts of machine load and unload.

7.10 SUMMARY

This chapter has described an implementation of the UMC reference architecture to enable the construction of machine controllers in a generalised manner. Tools have been created to allow the description of required machines and their run time control. Sufficient UMC components have been produced to control a target application involving the use of distributed manipulators for discrete parts handling. The capabilities of the UMC system are however inherently expandable.

The current system has been implemented using the OS-9 real-time operating system which offers suitable features and performance as detailed in Appendix A. Chapter eight details the design and implementation of selected UMC modules by the author.

Chapter 7: References

1 Lent, B. "Dataflow Architecture for Machine Control", Section 2.3, Machine Embedded Control System, Wiley, 1989.

2 "Motion Controller Survey", PCIM (Power Conversion and Intelligent Motion), USA, August and September 1988.

3 Jasany, L.C., "PLCs Into the 1990s", Automation, USA, April 1989, pp 20-24.

4 Goh, A.S., "A Pneumatic Servo Axis Handler for Universal Machine Control", MSc Thesis, Loughborough University, September 1989.

5 Booth, A. H., "UMC Release Notes", Version 1, Loughborough University, Department of Manufacturing Engineering, January 1990.

6 Schmitz, D. et al., "The CMU Reconfigurable Modular Manipulator System", Technical Report CMU-RI-TR-88-7, Robotics Institute, Carnegie Mellon University, Pittsburgh, USA, May 1988.

7 Weston, R. H. et al., "Universal Machine Control System Primitive for Modular Distributed Manipulator Systems", Int. J. Prod. Res., Vol 27, No. 3, pp 395-410, December 1988.

8 Harrison, R. et al., "A Study of Application Areas for Modular Robots", Robotica, Vol. 5, 1987, pp 217-221.

9 Thatcher T. et al, "Software Design for Modular Robots", Int, J. Prod. Res., Vol 22, No. 1, 1984, pp 71-84.

10 Thatcher T., "Control System Architectures of Distributed Manipulators and Modular Robots", PhD Thesis, Loughborough University, October 1987.

11 Dibble, P., "OS-9 Insights: An Advanced Programmers Guide to OS-9/68000", Microware Systems Corporation, Des Moines, Iowa, November 1988.

12 Booth A. H. "Module Magic Release Notes", (Internal Document), Manufacturing Eng. Loughborough University, 1989.

13 Anon., "OS-9 Basic Programming Manual", Microware Systems Corporation, Des Moines, Iowa, 1990.

14 Anon., "OS-9 C Programming Manual", Microware Systems Corporation, Des Moines, Iowa, 1990.

15 Anon., "OS-9/68000 Operating System Technical Manual", Chapter 15: Events, Microware Systems Corporation, 1990.

16 Anon., "OS-9/68000 Operating System Technical Manual", Section 2 - The Input/Output System, Chapters 5 to 12, Microware Systems Corporation, 1990.

17 Anon., "PC9 User's Manual", Quin Systems, Wokingham, 1990.

18 Anon., "Sun's Software Overview", BE125-0/75K, Sun Microsystems Inc., Mountain View, CA 94043, 1987.

19 Anon., "Unibridge Technical Reference Manual", Microware Systems Corporation, 1989.

20 Wright, A. C., "Universal Machine Control - Development of an Input/Output Software Module", MEng Report, Loughborough University, Department of Manufacturing Eng., 1988.

8
Implementation of Selected UMC Elements

8.1 INTRODUCTION

This chapter provides an in-depth discussion of UMC implementation methods and documents the design of selected elements of the run time system which have been created by the author[1]. The discussions reflect the state of the UMC system up to September 1990. This is obviously subject to change as the scope and capabilities of the UMC software are progressively enhanced [1].

8.2 RUN TIME MACHINE IMPLEMENTATION

8.2.1 Introduction

As discussed briefly in section 7.9 the run time creation of every UMC machine involves a sequence of *load* and *link* operations, before the applications related UMC *control* operations can occur. This section reviews these operations as a precursor to discussion of UMC system programs.

8.2.2 Machine Creation Cycle

Before UMC machine creation, the chosen target system should consist of appropriate control hardware to support the UMC machine to be created and loaded. For example physical device interfaces should exist which are appropriate to the particular device handlers specified in the machine information module.

At system run time the machine loader (*ml*), a UMC machine level utility, is the first program to be executed. See figure 8.1.

---

[1]Complete source code listings are not included in this thesis although they may be available by arrangement with the Modular Systems Group.

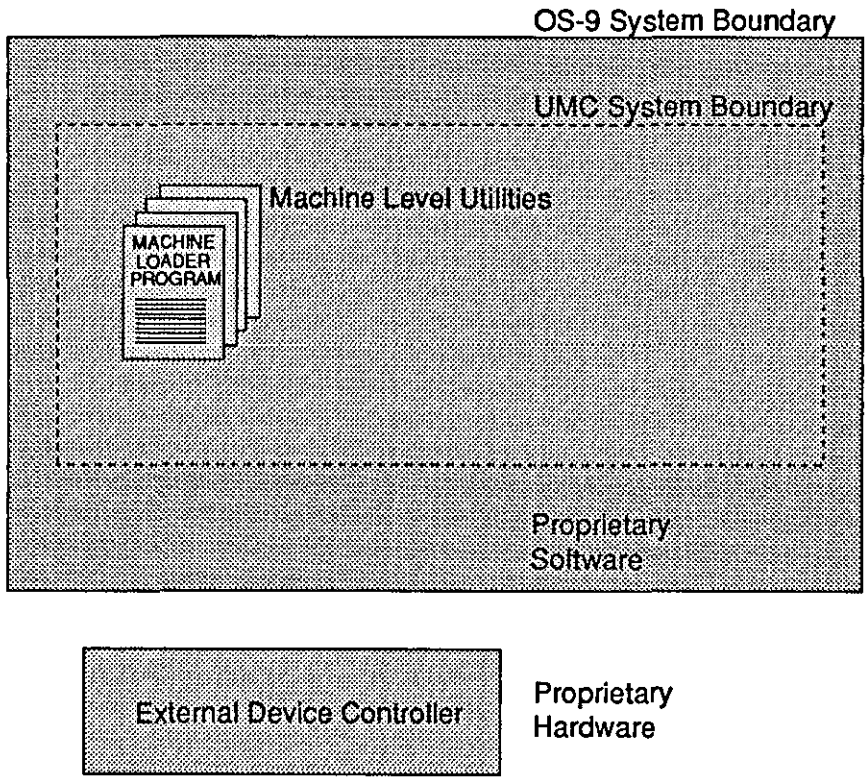


Figure 8.1. Initial state of UMC run time system. Note that for simplicity only one external device is shown.

The loader's initial action is to create a machine information module in memory. A set of specified *task* and device *handler* related program and information modules are then created together with their associated events. These modules form the memory resident software components of the UMC run time machine controller. Module cross referencing is achieved via the writing of suitable pointers into the information module headers. For example each *task* information module contains pointers to its associated *machine* and *location* information modules. See figure 8.2. Task and device handler processes are initiated through execution of appropriate program modules. The current loader then dies.

The processes created by the machine loader execute concurrently. The task processes link to their associated information modules by name and then establish "links" with the UMC entities[2] they wish to control or access i.e. axes, ports, events and locations. See figure 8.3.

[2]The term *entity* is used in this thesis to describe any elements of the UMC system which are *linked* to applications related tasks. The term *component* has been previously used in other UMC documentation for such elements.

After linking, subsequent communication between UMC entities and tasks is achieved via pointer and offset values. These numerical values are returned by the link functions and their use increases UMC system run time efficiency.

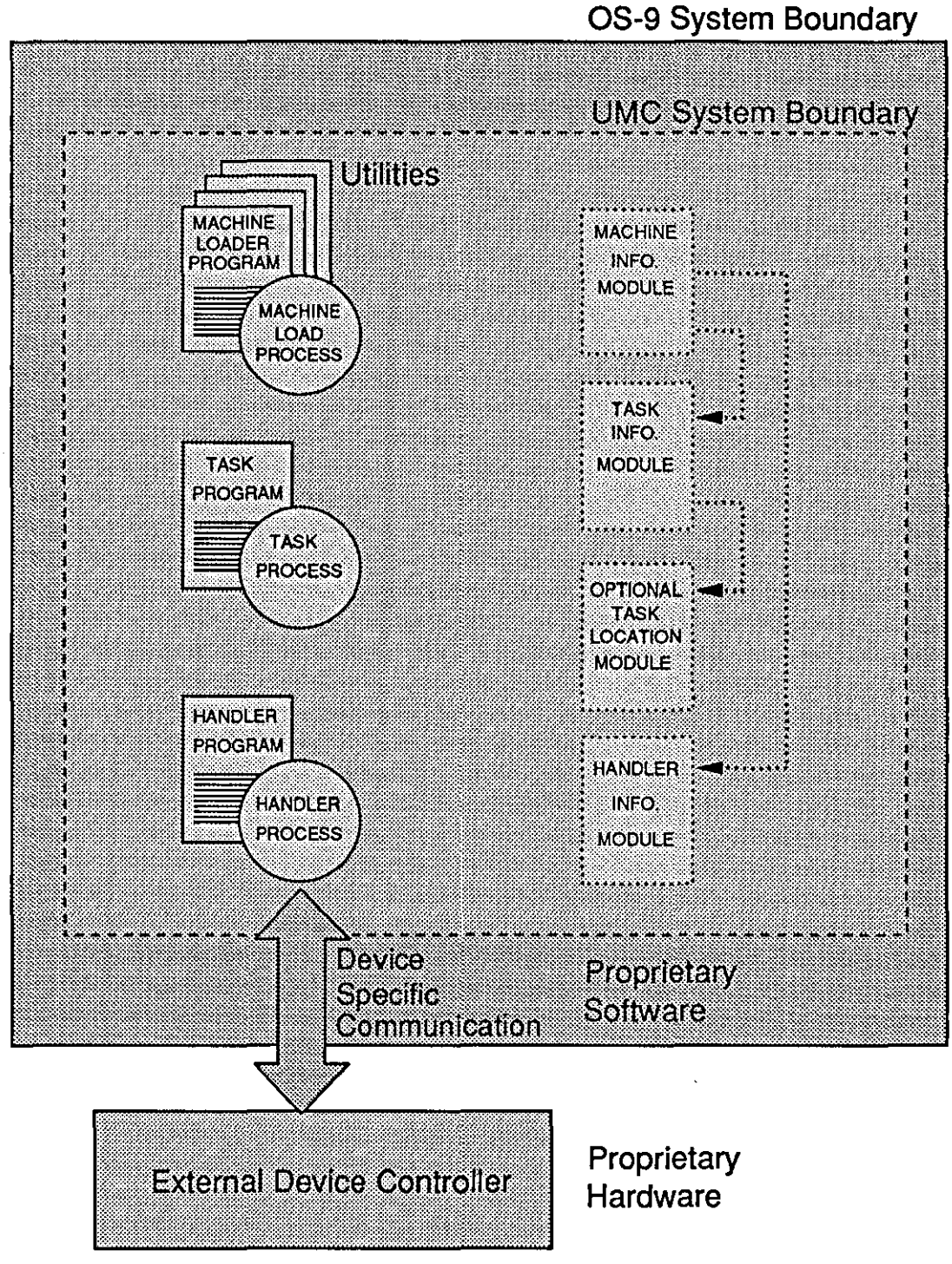


Figure 8.2. State of UMC run time system after loading. Note that for simplicity only one task, one handler and one associated external device are shown.

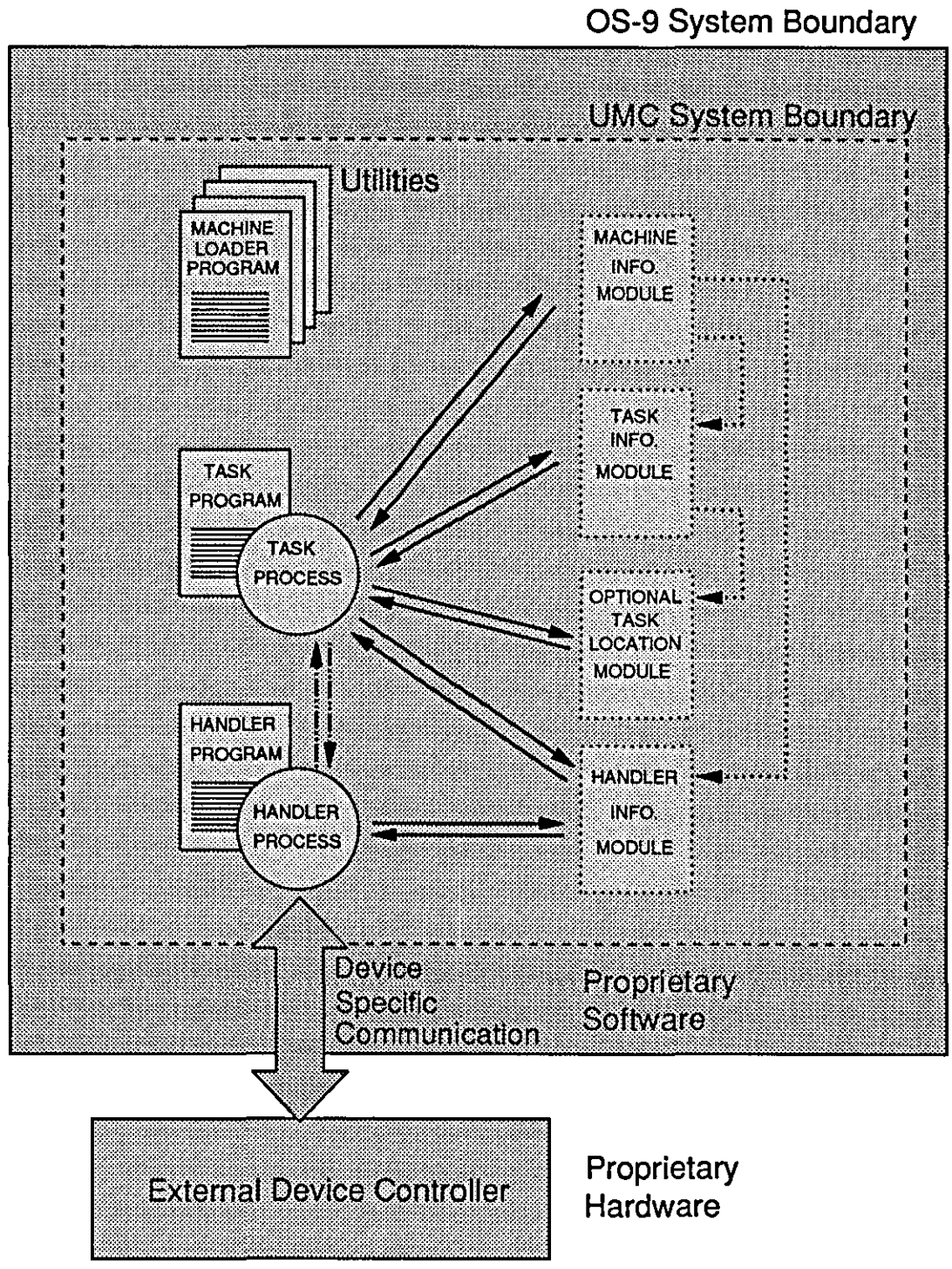


Figure 8.3. State of UMC run time system after linking. Note that inter-process communication and information module access are indicated by arrows.

8.2.3 Information Module Access

Sections 7.5.3 and 7.5.4 provided an overview of the role of information modules within the UMC architecture. This section considers the information access mechanism in more detail.

Although the data structure of the UMC system need not be understood by application programmers, UMC system programming requires extensive data manipulation. System programs including the machine load utility and UMC task linking and control functions must manipulate data within a number of different information modules.

Each UMC information module contains two "classes" of data structure. *Information module headers* which appear only once and repeated groups of data fields termed *information sub-structures*. See figure 8.4.

C functions provided by Booth [2] (within the *mmlib.l* library) enable the individual data fields in information modules to be accessed. The current implementation supports four types of data field, character (one byte), integer (four bytes), float (four bytes) and string (32 bytes).

The location of UMC information within a given module is expressed relative to the beginning of the user data area. This should not be confused with the OS-9 data module header which is located at the top of all OS-9 modules [3]. See figure 8.4.

A library of offset functions are provided (the *offset.l* library) which calculate the required offsets from the beginning of the UMC data area to given data fields. These offset values are then used in conjunction with the data access functions contained in *mmlib.l* [2] to obtain specific information.

In each program which is to access a particular information module type, the names of a corresponding set of *C header files* must be included. These header file names are of the form *tem*.h* for templates which describe the information module header structure and *sub*.h* for information module substructures where * is a character string. In order to provide an example appendix D lists the header files which enable machine module access from UMC programs.

Figures 8.5 and 8.6 list the current machine and task information module contents. These contents are of course subject to change. For complete details refer to the latest UMC release notes [2].

The data field names are indicated in figures 8.5 and 8.6 by hyphenated *italic* text[3]. These names subsequently appear in the same form within the main text of this thesis to aid the reader.

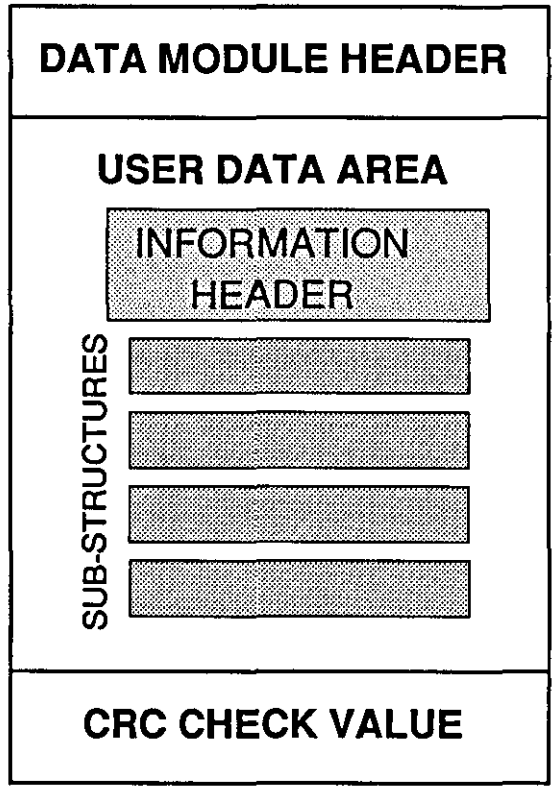


Figure 8.4. Schematic of UMC information module organisation.

[3]Note some of these fields are *not* directly editable by the user.

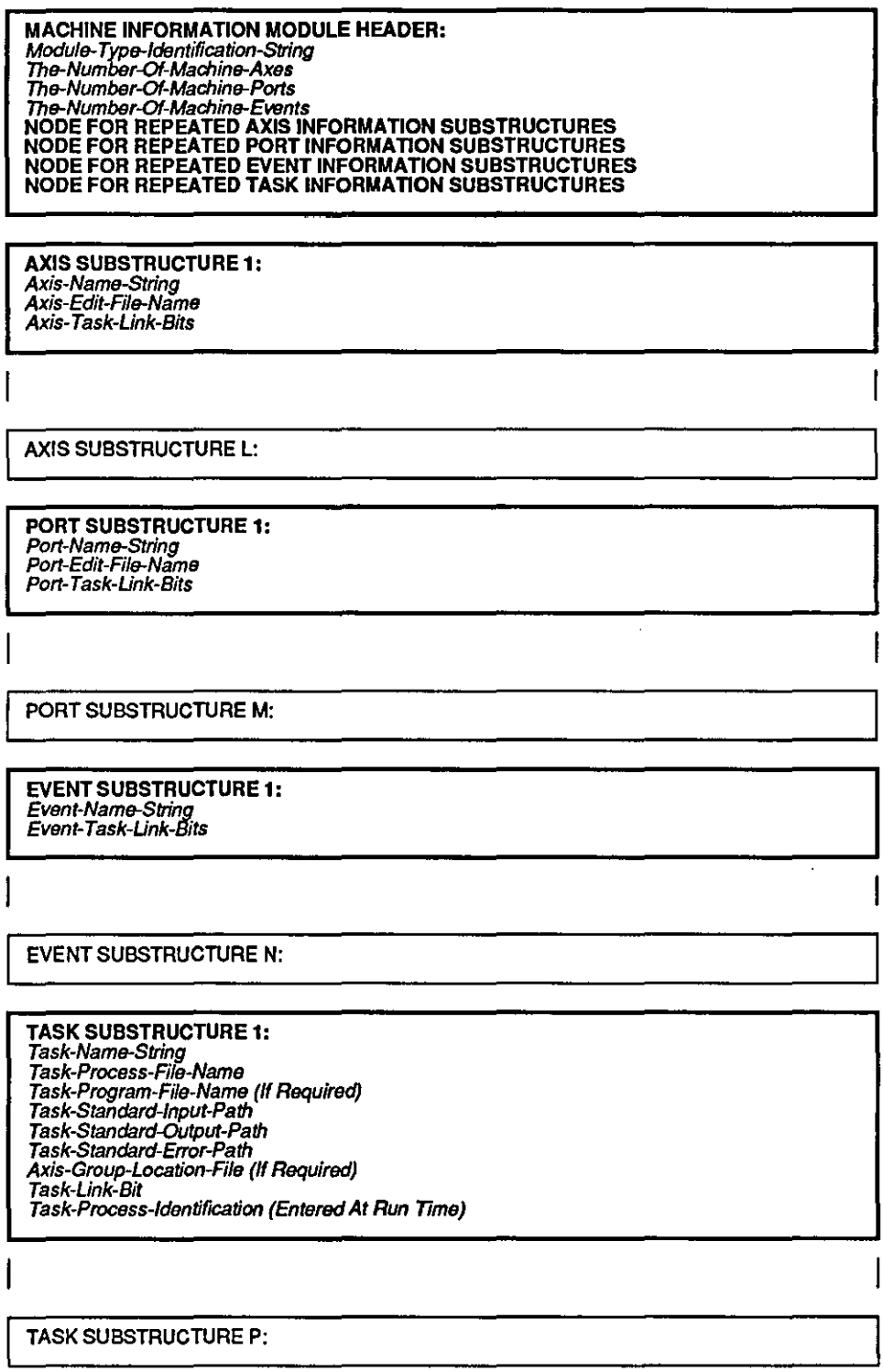


Figure 8.5. Machine information module contents.

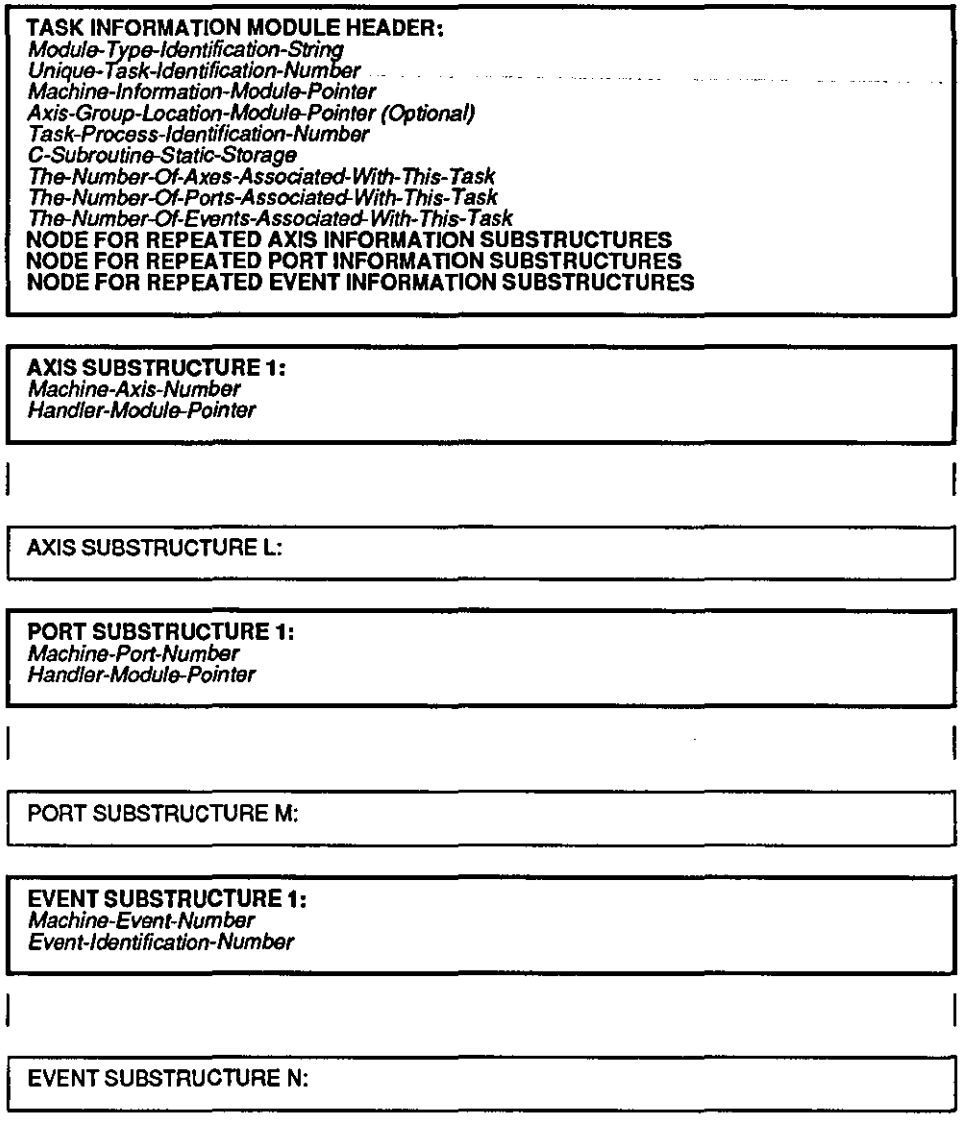


Figure 8.6. Task information module contents.

8.3 MACHINE LOADER

8.3.1 Introduction

This section seeks to provide an overview of the operation of the machine loader program (*ml*). The machine loader described here reflects the author's version (source code ref. *ml89.c* - last modified 19-4-89) plus subsequent modifications by Booth [3], chiefly to allow the loading of multiple machines. A debug version of the machine loader (*ml_d*) has also been created by the author. This version prints messages as it proceeds to provide user diagnostics.

8.3.2 Machine Loader Program Description

The following text provides an outline description of the machine loader's operation. The bracketed numbers reference the structured design charts illustrated in figures 8.7and 8.8.

(1) Process Command Line Arguments:

The machine loader command line has the following syntax:

ml [<opts>] <machine name> [<opts>] {<task names> [<opts>]}

The *machine name* is the name of the machine information module edit file to be searched for. The specification of optional *task names* by the user allows the partial loading of a machine. In such cases only the named tasks, if valid, will be created together with their associated UMC entities.

Options (opts):

-c *Don't Load Entities.* If set, no axes, ports or events will be created

-i *Display Load Information.* This flag causes a table showing machine related information to be displayed.

-l *Load Tasks From Edit Files.* If set, task modules are created from edit files. Task modules are normally created directly from information in the machine module.

-p *Don't Pass Task Arguments.* This flag suppresses the passing of any argument string when a task is created. This string is optionally specified in the machines module's task sub-structure *Task-Program-File-Name* field. If set, program data files associated with tasks written in Microware Basic will not be loaded. See figure 8.5.

-m *Set Machine Number.* Supports the creation of multiple machines. The machine number defaults to zero if the flag is not used [3].

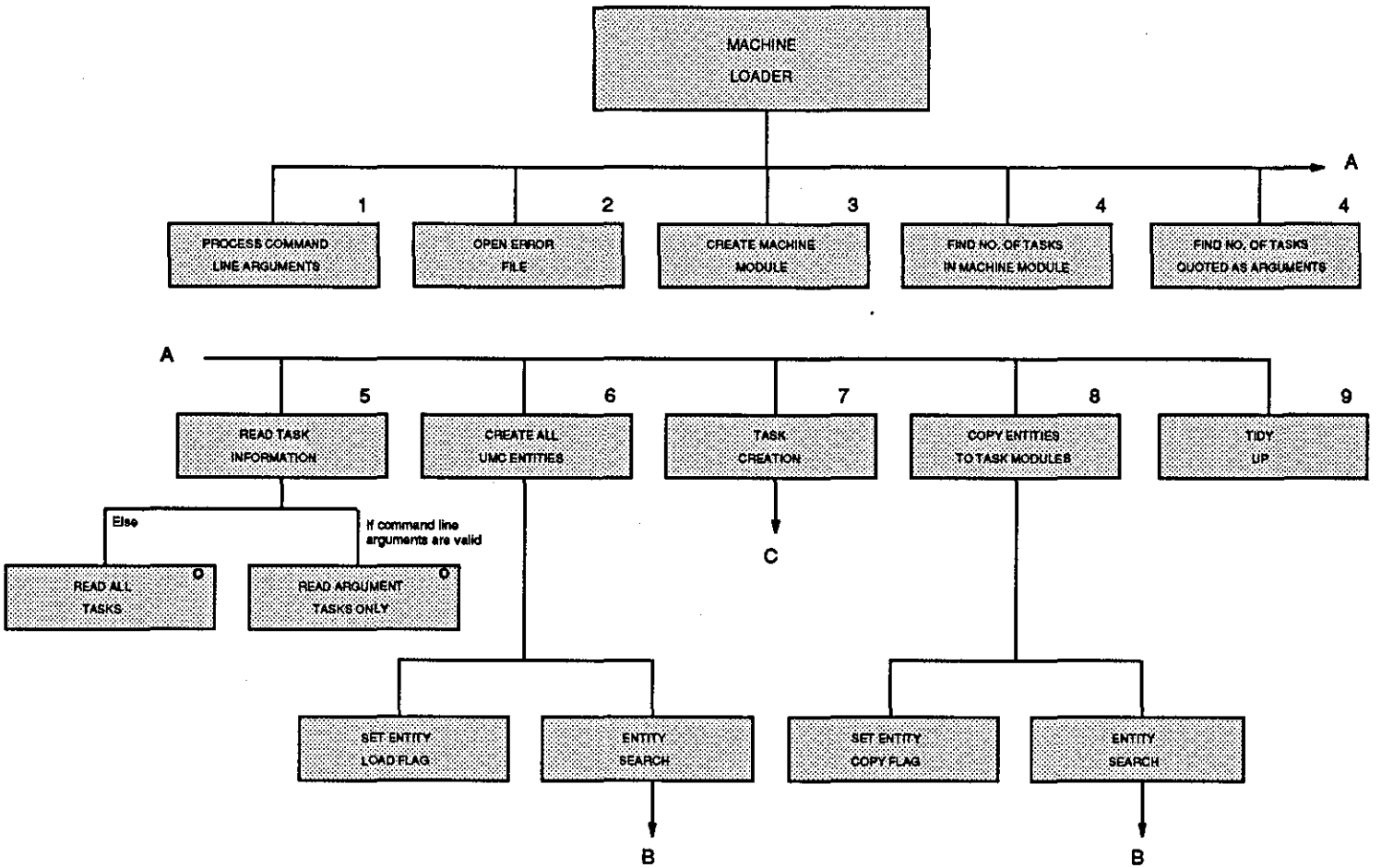


Figure 8.7. Structured design chart for machine loader.

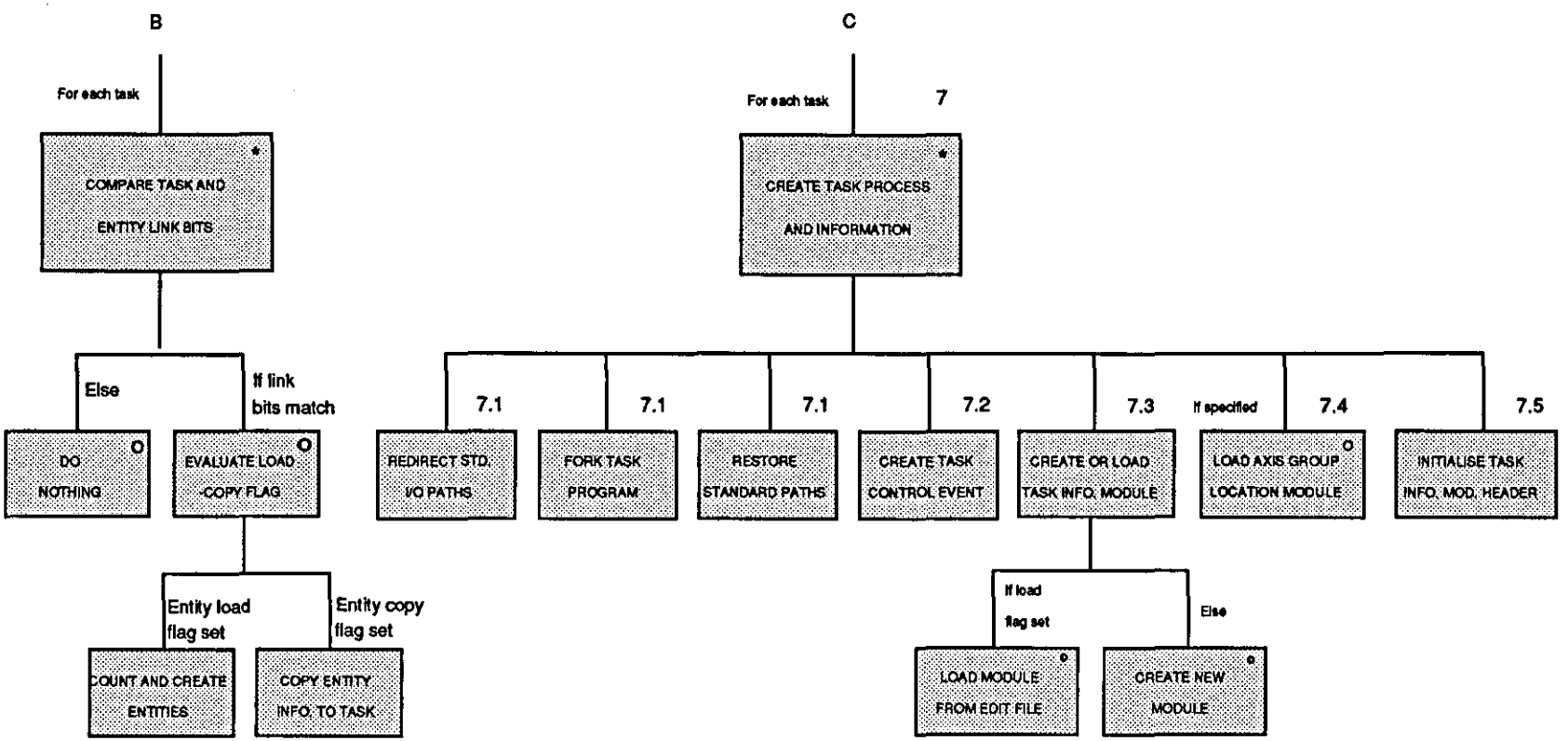


Figure 8.8. Structured design charts for machine loader.

(2) Open Error File:

A UMC specific error message file has been implemented. This has allowed the OS-9 error reporting system to be extended with UMC specific error codes and associated messages.

(3) Create Machine Module:

The machine information module is created from its disk based edit file.

(4) Evaluate Number of Tasks to be Created:

The number of tasks required by the user is evaluated. If no tasks are explicitly stated as arguments on the command line then all the tasks specified in the machine module will be created. If a list of tasks is explicitly stated by the user then only these tasks (and their associated axes, ports and events) will be created.

(5) Task Information to Internal Data Structure:

For each task to be loaded the *Task-Name-String*, *Task-Link-Bit* and the *Task-Substructure-Number* are read and placed in an internal data structure local to *ml*. See figure 8.5. The loader then references this structure during the subsequent stages of machine creation.

(6) Create All UMC Entities:

The number of UMC entities required for each named task are counted for each entity type. i.e. For axes, ports and events.

For each entity type and for each entity node number, the unique *Task-Link-Bit* is cross referenced against the *Task-Link-Bits* of each entity. See section 7.6.2. In each case where a match occurs the entity is loaded or created as appropriate. The events are created using the *Ev$creat* OS-9 system call [17]. The axis and port information modules are created from edit files. The loader then reads the *Handler-Process-Name* from each axis or port information module and forks the handler process.

(7) Task Creation:

The following operations are performed for each task in turn:

(7.1) The user defined standard I/O paths are read from the machine module for each task. See section 7.7.5 and also figure 8.5, *Task-Standard-Input/Output/Error*

fields. These standard paths are redirected as specified typically to terminals or windows for output, and keyboard(s) for input. This involves *duplicating* the standard I/O paths before forking each task process and then subsequently replacing the original paths. See [4] and [5]. Recent additions by Booth provide support for task redirection to an *Internet* host machine [6]. This enables a UNIX based workstation and its associated windowing environment to be used as a remote user interface.

Each task is created using the OS-9 *exec()* function with the redirected paths [15]. An optional argument string may be passed to the forked task process. This allows the referencing of a program file when a Microware Basic process is forked [7].

(7.2) A task control event is created for each task forked. This is a global flag which prevents task execution until machine loading is complete.

(7.3) For each task an information module is created. Its contents are derived from machine module information fields or may optionally be loaded from a disk based edit file.

(7.4) If specified, an associated axis group location module will be loaded for each task which includes axes of motion.

(7.5) The task information module header is accessed and the *Machine* and *Axis-Group-Location-Module-Pointers* and the *Task-Process-Identification-Number* fields are filled in. The task *C-Subroutine-Static-Storage* field is initialised to zero. See figure 8.6.

*(8) **Copy Entity Information to Task Modules:***

At this stage the appropriate UMC modules are resident in memory. The cross referencing between task processes and their associated information modules must now be achieved.

For each entity type and for each required entity, the *<Entity>-Task-Link-Bits* are read and compared with the unique *Task-Link-Bit* of each task. If a bit match occurs this indicates that the task wishes to access or control this UMC entity. *N.B. This algorithm reuses the code described in section (6).* Each time a match exists, the entity number (i.e. the event, axis, port substructure number) is copied from the machine module to the appropriate task module entity substructure number field (e.g. *Machine-Axis-Number*). See figures 8.5 and 8.6.

(9) Tidy Up:

Loading is now complete. The task control event(s) can be cleared and the loader dies.

8.3.3 Naming Conventions

Information module naming conventions have not been discussed in the above description of the machine loaders operation. These conventions were significantly modified by Booth in August 1990 to provide for the creation of multiple UMC machines within a single operating system environment [3]. See figure 7.6.

The current information module naming convention is as follows:

machine module	*mXXmac*
task module	*mXXtakZZ*
location module	*mXXlocZZ*
axis module	*mXXaxWW*
port module	*MXXptYY*
task control event	*mXXctlZZ_c*

Where *XX* is the machine number, *ZZ* is the machine task number, *YY* is the machine port number and *WW* is the machine axis number. *'m'* is currently fixed at zero until multiple machines are fully supported.

8.4 MODULAR APPROACHES TO TASK PROGRAMMING

8.4.1 Introduction

This section reviews modular programming methods as a precursor to discussion of UMC task program implementation.

Modular program design is the subject of exhaustive discussion in the field of computer science [8], [9]. Recent general purpose programming languages for example C, Pascal and Modula-2 permit the writing of programs in a modular man-

ner [10]. Languages such as C++ provide further support for object oriented approaches to programming [11].

Whichever language is adopted conventional program modules are essentially separate pieces of code which are combined by the language or a suitable link editor to form a single program which can be loaded and executed. As modification becomes necessary, only files which contain changes need to be recompiled although all files must be checked before final program creation [3]. If the modules used to build a program are written in a generalised and well defined manner, they can be progressively collected until subsequent programs can be built largely from existing modules. Most current programming systems support static modularity, where a number of separately identifiable source code segments are compiled to form one large executable program. In a multi-processing environment however such an approach often wastes memory and *individual* program structure is fixed with no opportunity for dynamic reconfiguration.

8.4.2 OS-9 Subroutine Modules

As discussed in section 7.5 memory modules are central to the overall design of OS-9. Memory modules are *separate* reserved areas of system memory which may contain either programs or data. A subroutine module is a particular type of program module which contains user defined subroutine code which may be called from main program modules [3].

Puckett and Dibble [12], have advocated the use of subroutine modules as a powerful method to enable software reuse, efficient memory utilisation and dynamic program reconfiguration. The use of OS-9 subroutine modules is now considered in relation to the modular construction of programs.

Subroutine modules offer dynamic modularity for program segments. OS-9 subroutine modules are initiated simply by including appropriate procedure calls in a main program module. If several processes are running at the same time not only can re-entrant subroutine modules be shared between processes but they can be loaded and unloaded dynamically. This saves memory and offers a high degree of operational flexibility. Subroutine modules are bound in memory only while they are actually executing. If code modifications are necessary only the subroutine modules containing changes need to be recompiled and linked. The old modules must of course be replaced with the new versions, in the appropriate executable directory on disk or loaded into system memory, but that is the *entire* extent of the

change [3]. Modules in an executing program can thus be replaced without stopping the program. This is potentially a very powerful method for dynamically modifying application code. Subroutine modules no longer required can be removed, again `while` processes are running.

8.4.3 Applying Subroutine Modules to UMC

Although they have not as yet been widely adopted, as explained in section 8.4.2, there are powerful arguments for the use of subroutine modules, particularly with regard to dynamic configurability, efficient use of memory and potential code reuse. Both Microware Basic I-code and object code subroutines have been created and used by the author. Section 8.5.2 will discuss the implementation of subroutine modules for UMC specific functions.

Microware Basic I-code and 68000 assembler are the only fully supported subroutine languages. Basic I-code subroutines are not adequate for UMC purposes because Basic does not *fully* support the OS-9 system calls which are required to implement UMC specific functions. Unfortunately no programming tools are currently available from Microware to specifically support subroutine program development in C. This makes their current use relatively complex.

8.4.4 OS-9 Trap Library Modules

Another method to enable modular program construction and code sharing between programs at run time is to utilise the TrapLib (Trap Library) module type [3]. Trap modules act as libraries containing globally-accessible collections of services and their principal role is to achieve memory conservation. In contrast each subroutine module usually provides only a single service or function. Traplib modules are distinguished from subroutines by their access method which is incompatible with Microware Basic procedure calls. Typically accessed from C programs, trap modules are easier, although less efficient to use than subroutine modules [3]. They are extensively used by Microware for both maths and I/O functions in order to conserve memory.

8.5 MICROWARE BASIC TASK PROGRAMS

8.5.1 Introduction

It is obviously desirable to simplify the writing of user application tasks as much as possible. One approach to this is to write task programs in an existing, productive, general purpose programming language such as Basic. This section reviews the work of the author to enable the creation of UMC task programs in Microware Basic via embedded UMC specific subroutines.

Microware Basic is an unusual language since it is a hybrid which is claimed to combine the best features of Basic and Pascal [14]. Microware Basic provides its own development environment as illustrated in figure 8.9. This incorporates an interactive compiler which provides extensive symbolic debugging capabilities. The Microware Basic compiler output is an intermediate code (I-code) that is then interpreted at run time. Execution is fast in comparison to most Basic language implementations although of course slower than native machine code execution [7]. A serious limitation of the current version of Microware Basic is that it does not *directly* support OS-9 signal or event system calls which are required for UMC implementation [13]. Complete details of Microware Basic are provided by the user manual [7].

As discussed in section 7.7.3 UMC currently supports task programs written in Microware Basic and C. C is the base level implementation language for UMC and support all its capabilities. UMC Version One release notes provide full documentation of these functions [2]. Basic currently supports a subset of UMC task instructions which could readily be extended if a complete implementation is required. The UMC specific Microware Basic language extensions have been achieved through the use of subroutine modules.

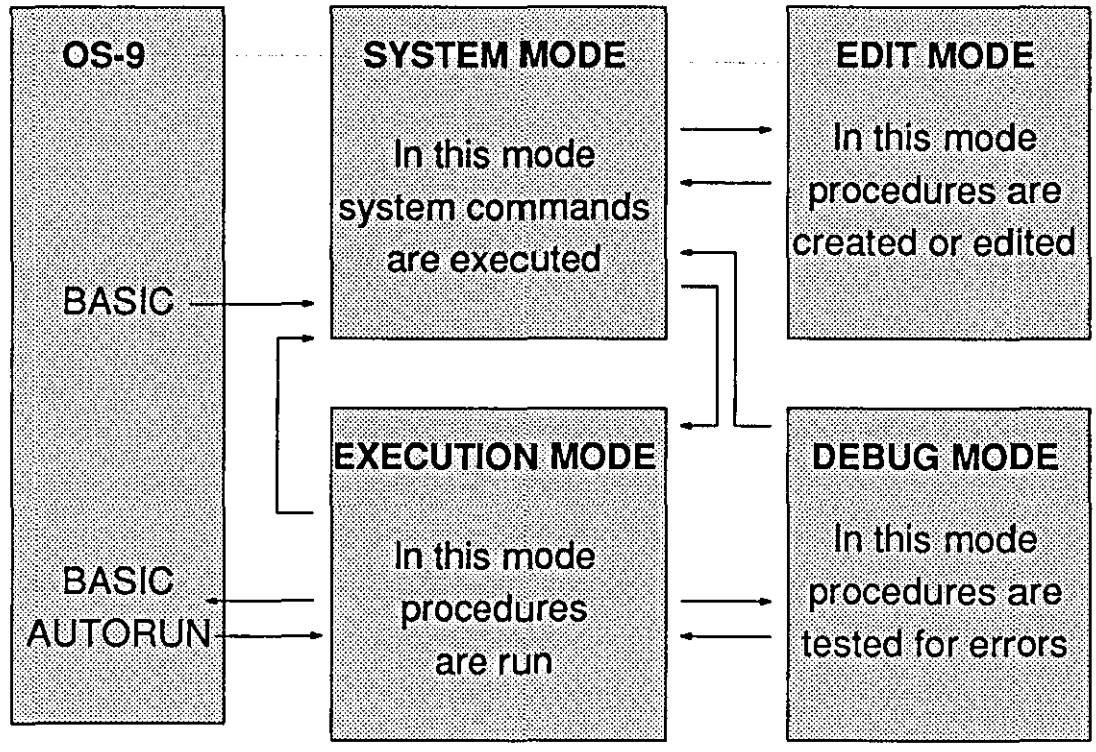


Figure 8.9. Microware basic environment operating modes.

8.5.2 Task Subroutine Module Implementation

8.5.2.1 Introduction

For the reasons mentioned in section 8.4 the use of subroutine modules is a very attractive concept. The major purpose of the subroutine modules implemented by the author has been to provide a method of embedding UMC entity linking and control capabilities into Microware Basic in as transparent a manner as possible.

8.5.2.2 Microware Basic Procedure Calls

A Microware Basic program is composed of a series of procedures which are initiated by *RUN* statements [14]. If the procedure named by a *RUN* statement cannot be found within the workspace, Basic will determine if it has been loaded outside the workspace. If it is not found there Basic will try to find a disk file having the same name in the current execution directory, load it and run it. In each case Basic checks to see if the called procedure is *Microware Basic source code*, a

Microware Basic I-code module or a *68000 machine language module* and executes it accordingly. See figure 8.10. This provides a very flexible approach to program development.

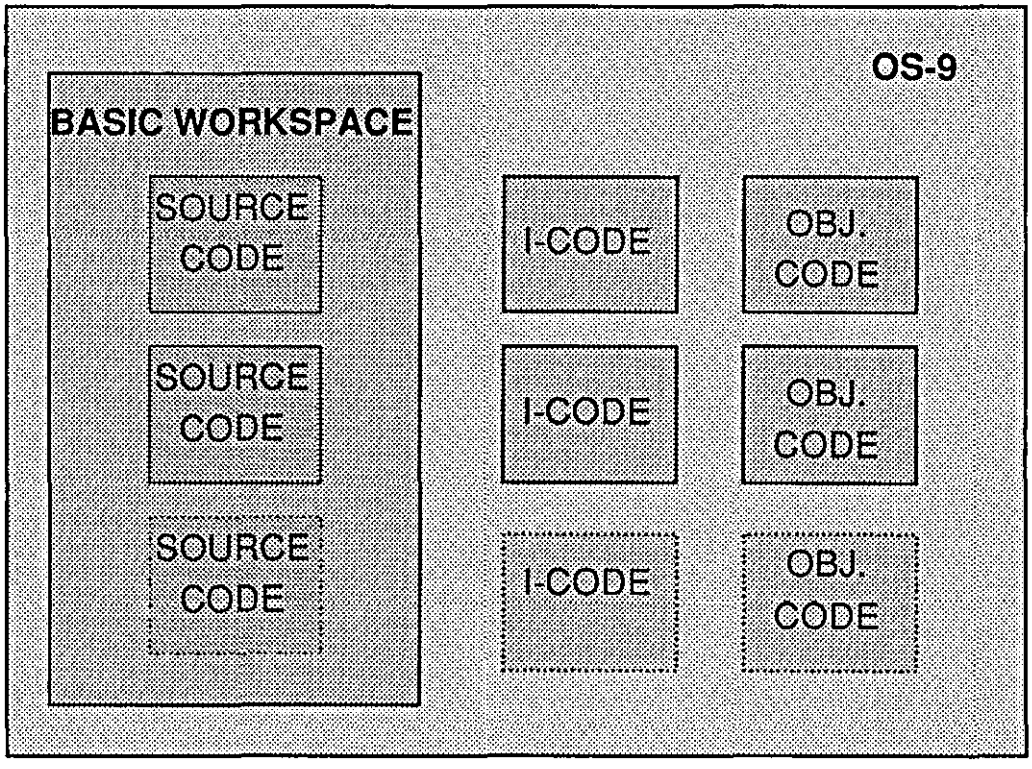


Figure 8.10. Basic program module types.

Machine language subroutines may be called in the same manner as a Microware Basic procedure using a *RUN* statement. Only pointers to each parameter are passed as arguments to a subroutine, all variables being called-by-reference and a function may not return a value as in C [7].

When Microware Basic calls a machine language subroutine the arguments are passed via registers *d0* and *d1* with the remaining arguments pushed onto the stack. Microware Basic then executes a *jsr* instruction to the entry point of the new subroutine module [7]. See figure 8.11.

8.5.2.3 C Subroutine Creation

Assembler subroutines can be readily created as described above. However for UMC the requirement was to utilise subroutines compiled from C source code[4]. Normally a C program is not compiled as a root program section (*psect*), but is linked with the previously assembled file *cstart.r*. *Cstart.a* is a short piece of assembler code which sets up the arguments and some of the global variable space for a C program when it is forked [15]. Unfortunately the creation of subroutine modules is incompatible with the function of the *cstart.a* code. The solution to this problem was to create a new subroutine specific root *psect*. This has been named *umcstart.a* and is listed in appendix E.1. This program was assembled to produce a relocatable object file *umcstart.r* which could then be linked to compiled C programs. The programs created can then be loaded into memory as re-entrant subroutine modules.

It is important to appreciate that *cstart* provides a number of important functions for C programs, especially with regard to global static storage and support for high-level C I/O functions. The C code written within subroutines must therefore be restricted to simple I/O and memory manipulation unless additional support is provided. Assembler code within *umcstart.a* and a simple C output library written by the author provide the required functions to support UMC subroutine implementation. See appendix E.

Within *umcstart.a* assembler code is provided for:

(1) Dummy stack checking for standard library functions (true stack checking is normally provided by *cstart*) and declaring global static storage. E.g. for the variable *errno* which is automatically created as part of any C program to enable error reporting [16].

(2) Changing the static storage provision to a unique area (provided within the header of the task information module) for each task process calling the subroutine and restoring the processor registers to their original state before returning to the Basic calling procedure.

(3) Reporting error codes back to Basic since the methods used for calling functions and reporting errors are not directly compatible between C and Microware Basic.

---

[4]The creation of subroutine modules from C source code and their integration within UMC is largely undocumented elsewhere and is therefore described in detail.

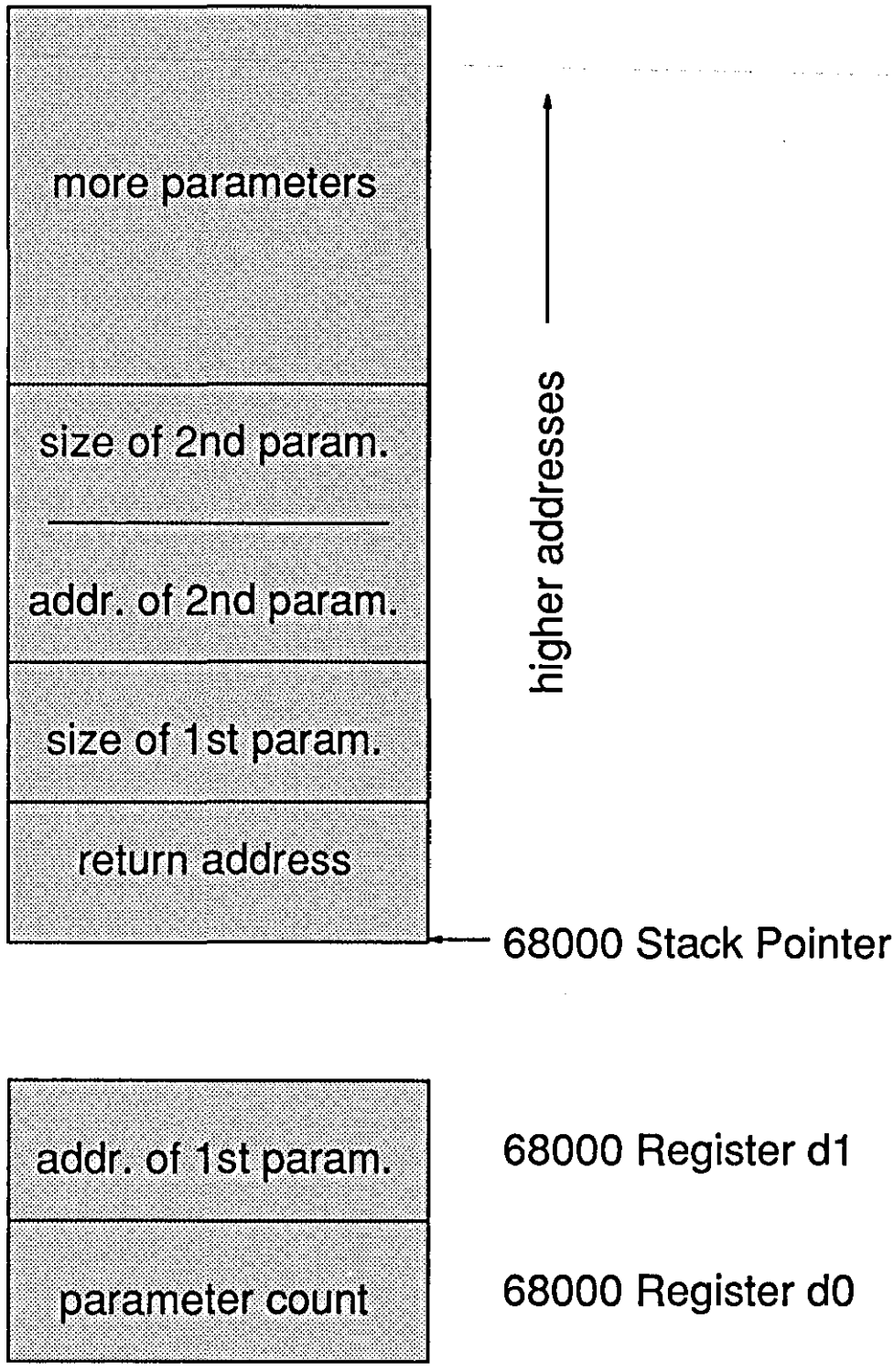


Figure 8.11. Microware basic subroutine call.

8.5.2.4 Provision of Static Storage

The Microware implementation of C on the Motorola 68000 processor uses the *a6* register to store the base address for global and static variables. This base address is passed to a C program when it is forked and *never* changed by C code. Thus when the subroutine is called the contents of the *a6* register must be saved and its value changed to address a specified area for C subroutine program static storage. In order to make the provision for this static storage external to the Basic user task program a 28 byte field for *C-Subroutine-Static-Storage* is reserved in the header of each task information module. See figure 8.6. Note that subroutine program modules must not contain static storage internally in order to allow them to be shared.

8.5.2.5 Returning from C Subroutines

A subroutine returns to the original calling program by executing an *rts* instruction. The called function must restore the registers to the values they contained when the function was called. The *only* exceptions are data registers *d0*, which originally contained the parameter count and *d1*, which is used for returning error code values [7].

Subroutine modules return error status by setting the carry bit of the processor condition code register and by setting the low order word of data register *d1* to the appropriate error code. C uses *errno*, a static variable, to hold error numbers. Therefore if an error occurs the value in *errno* must be coped to *d1* and the carry bit set before performing an *rts* instruction to return to Basic.

8.5.3 UMC Subroutine Specification

8.5.3.1 UMC Data Types

A set of UMC specific data types must be defined at the start of each Microware Basic task program to be used for control purposes. See example programs in appendix C. These are of the following form:

Data Type	Structure
axis structure:	32 character name string, axis module pointer, axis number
port structure:	32 character name string, port module pointer, port number
event structure:	32 character name string, event module pointer, event number event value
location structure:	32 character name string, location module pointer, location number

N.B. All other variables used in UMC subroutine calls are of data type *integer*.

8.5.3.2 Subroutine Call Syntax and Description

The following subroutines have currently been implemented and tested from Microware Basic:

setup()

Creates a communication path for communication to a Microware Basic task from axis and port handlers. Similar to the C function *open_caller()* [2].

link_event(<event structure>)

Links to a specified event. Corresponds to a C function with the same name [2].

link_axis(<axis structure>)

Links to a specified axis. Corresponds to a C function with the same name [2].

link_port(<port structure>)

Links to a specified port. Corresponds to a C function with the same name [2].

link_location(<location structure>)

Links to a specified location. Corresponds to a C function with the same name [2].

smove(<location structure>,<percentage speed>)

Straight line move instruction. Moves with a trapezoidal velocity profile to an absolute location. Velocity and acceleration are specified as percentages of the allowable maxima in the axis data modules. Corresponds to the C function *movelsp()* [2].

look(<event structure>)

Updates the event structure with the current value of the event. Similar to the C function *read_event()* [2].

signal(<event structure>, [<flag>])

Signals the specified event. This subroutine calls the OS-9 standard library function *_ev_signal()* [17]. N.B. The signal increment applied is determined by the machine loader when the event is created. This increment is currently set to one. This subroutine is similar to the C function *signal_event()* [2].

wait(<event structure>, [<min>, <max>])

Waits for a specified event. *Min* and *max* are optional parameters which determine the *ev_min* and *ev_max* values waited for by the OS-9 *_ev_wait()* function called by this subroutine [17]. If not specified these values are both set to one by the subroutine. This default case caters for the most common application of events which is as simple binary semaphores [18]. This subroutine is similar to the C function *wait_event()* [2].

output(<port structure>, <channel no.>, <new state>)

Sets the port channel to a specified output value. Similar to the C function *setoutput()* [2].

setdown()

Closes down task to handler communication. Similar to the C function *closehandler()* [2].

The example Microware Basic programs listed in appendix C provide more complete details of variable definition and procedure call formats. Appendix E.3 provides a source code listing of an example subroutine *wait()*.

8.5.4 Task Program Structure

The example UMC task programs listed in appendix C are of the following general form:

(1) *UMC specific type declarations* to support the current entity types i.e. axes, ports, events and locations.

(2) *UMC variable declarations* for the particular instances of the entities to be accessed or controlled from the task.

(3) *User variable declarations.* These are application related. Microware Basic is a general purpose programming language and any of its inherent features can be used as required by the particular application.

(4) *UMC set-up and link statements.* These are UMC specific. Set-up is associated with handler initialisation. See Booth [2]. Linking establishes a direct numerical reference to each UMC entity after initial referencing by name. Linking is discussed more fully in section 8.6.

(5) *Main program code* then follows with access to UMC entities achieved via procedure calls to subroutine modules. Compiled from C source code, these UMC specific subroutines support event, port, axis and axis location access and control.

8.5.5 Limitations of Microware Basic for UMC Task Programming

With the addition of UMC specific subroutines, Basic provides a productive environment for task programming. The approach is however currently limited in

certain respects and these limitations can only be effectively overcome through modification to the language source code.

Given the availability of source code for Microware Basic it is likely that the following enhancements could be implemented:

- The Microware Basic interrupt handler could be modified to support the receiving of user defined signals. The current lack of this facility restricts the methods used for task to handler communication.

- Subroutine static storage could become internal to the calling process. Currently space in task data modules has to be allocated for this static storage.

- UMC specific functions could be defined within the Microware Basic language. This would allow the provision of syntax checking, reserved words etc.

- The potential exists to fully automate UMC entity linking so that it is unseen by the user.

It is interesting to note that Seiko Instruments have recently obtained a source code licence from Microware and are known to have implemented similar language modifications for robotic applications[5]. Source code for Microware Basic could now be obtained although it is considered prohibitively expensive for research use at $40,000. More importantly however a general purpose language such as Microware Basic is no longer considered to be the preferred task program environment.

Microware Basic is not an industry standard language for control. Although it is a superset of ANSI Basic [7], in common with many similar languages, its best features are non standard and its implementation is operating system dependent [19]. Section 9.3.2 considers further shortcomings which can occur in utilising a general purpose high level programming language in industrial control applications. Due to such factors the author considers that it will be better to devote future effort into integrating emerging industry standard, problem oriented programming environments into UMC. See section 4.5.

[5]For further details contact David West, Microware Systems Corporation, Des Moines, Iowa.

8.6 UMC ENTITY LINKING FROM MICROWARE BASIC

8.6.1 Introduction

This section documents the implementation of UMC entity linking from Microware Basic. The entity linking subroutines described have been coded in C and compiled and linked to form a subroutine module for use with Microware Basic. Similar operations are performed by a set of C functions coded by Booth for use with C task programs [2].

As discussed in section 8.2.2 linking is performed by the task processes once loading is complete. Link calls enable initial reference to be made to a UMC entity via a meaningful name. If the entity is found to be valid a number is returned to enable future reference to the entity.

A single subroutine program module coded in C is used for linking all UMC entities, i.e. for axes ports, events and locations, from Basic task programs. This subroutine is not called *directly* from the Basic task program. The procedure call in the Basic task program initially calls a very short I-code subroutine module specific to the type of entity to be linked i.e. axis, port, event or location. This I-code subroutine sets up the appropriate parameters and in turn calls the common linking subroutine. See figure 8.12.

The calls to the I-code subroutines are described in section 8.5.3.2. (The source code for the common link subroutine described here is *link_cp89.c* - last modified 17-4-89.)

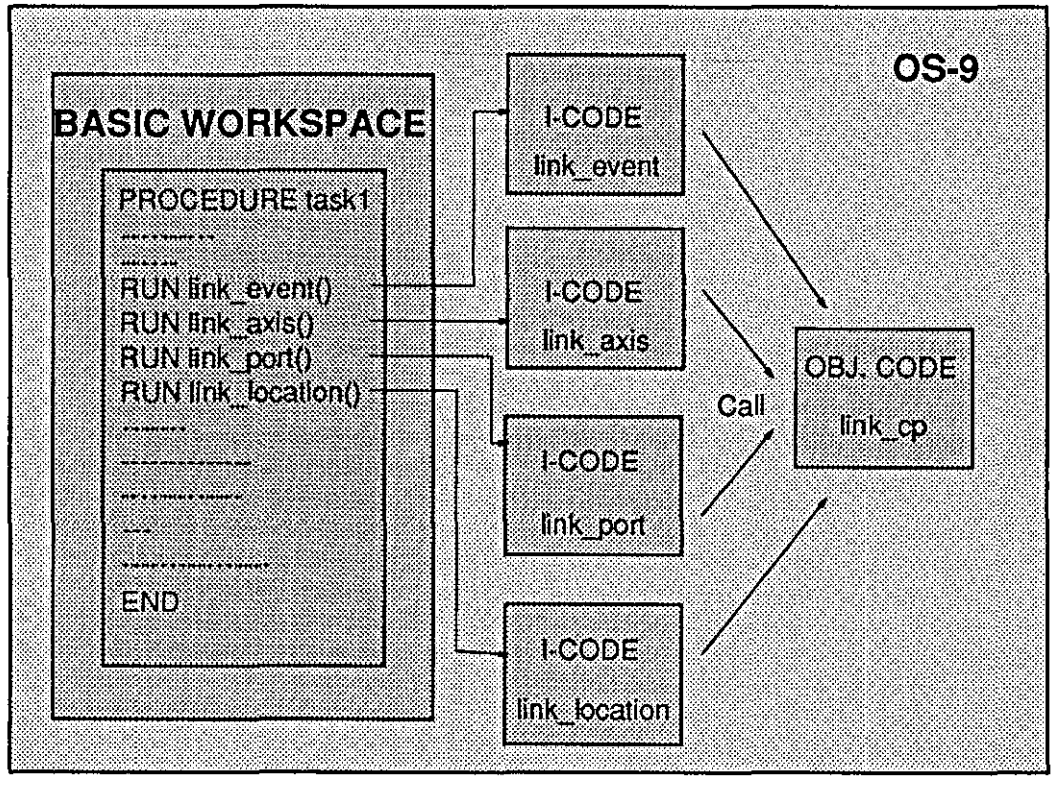


Figure 8.12. Implementation of linking from basic.

8.6.2 Linker Description

The following text provides an outline description of link subroutine operation. The bracketed numbers reference the structured design chart illustrated in figure 8.13. The hyphenated *italic* text references the data field names in the machine and task information modules illustrated in figures 8.5 and 8.6.

(1) Link To Task Information Module:

The linker first links to its own task information module and a pointer to the beginning of the UMC data area is returned.

(2) Change Static Storage:

The linker then immediately changes the program base address to the *C-Subroutine-Static-Storage* area reserved in the task information module header. See figure 8.6.

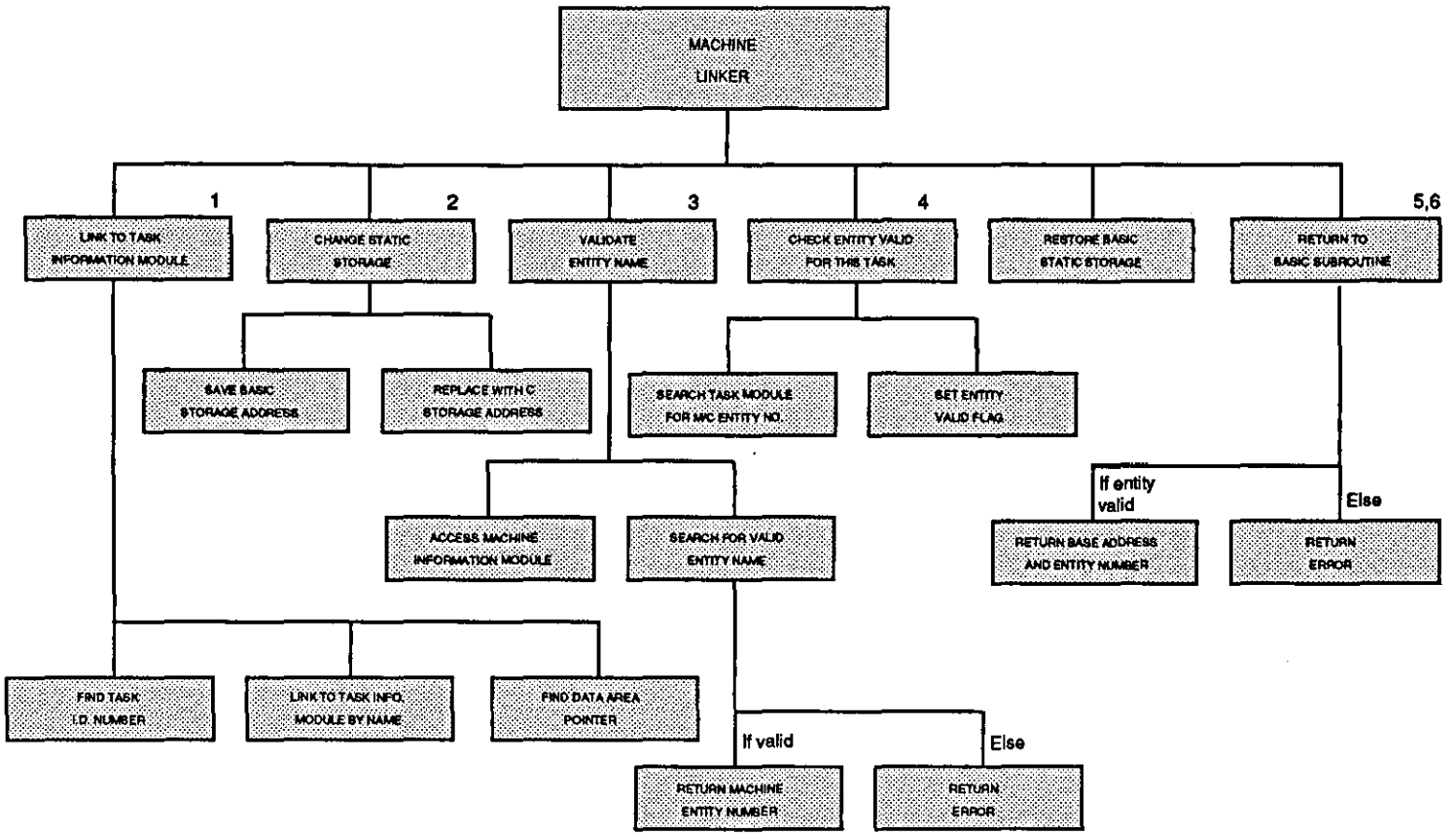


Figure 8.13. Structured design chart for UMC entity linking.

(3) Check Entity Name Is Valid:

For the entity type requesting linking a search is made of the entries in the machine module to check that the entity name is valid. The machine module is referenced via the *Machine-Information-Module-Pointer* in the task information module header.

(4) Check Entity Is Allocated To This Task:

This confirms that the specified entity has been allocated to this task by the machine loader (*ml*). Each task information module references a subset of the total UMC machine entities which are required to support the particular task process.

(5) Link Entity:

If the entity is found to be valid linking will take place. Linking simply involves the return of a UMC module pointer and the entity number to the calling program.

(6) Return Values:

The entity number and UMC information base address are returned to the calling I-code subroutine. This subroutine in turn passes the parameters back to the main task program to enable future entity access.

N.B. Location modules are referenced via an additional *Axis-Group-Location-Module-Pointer* placed in the task module header. See figure 8.6. The method employed is as described above except that during stages *(3)* and *(4)* the location module (not the machine module) is searched and if a matching location name is found this is returned as the entity number along with the UMC location module base address.

8.7 EVENT UTILITIES

As discussed briefly in section 7.7.4.2 events are the main method of synchronising and coordinating concurrent task programs, both with each other and with external changes of state. Events may also be used to pass numerical values between tasks.

Section 7.4.2 has considered the UMC machine development cycle from a user programming perspective. The recommended approach is that the task program

modules forming an application can be written and tested individually and then combined together in a very flexible manner. To assist in task debugging a set of utilities have been created by the author to allow the user to manipulate, create and observe events. The syntax and function of these utilities are briefly described here, more complete details are provided in the UMC release notes [2].

Event Deletion:

Syntax: *edel <eventname>*

Delete any named event. This utility first unlinks the event "eventname" successively until its link count is zero and then deletes the event.

Event Information:

Syntax: *einfo [<options>]*

Displays event information. The display provides the identification number, name, value, wait and signal increments, and use count for user task related events. An optional *-a* flag allows all system events to be optionally displayed.

Event Setting:

Syntax: *eset <eventname> <value>*

Simply sets the value of an event.

Event Pulse:

Syntax: *epulse <eventname> <value>*

Pulses the named event to the stated value. The first process waiting for the event will be activated. This enables a series of processes initially all in a waiting state to be queued and released by a single named event

Event Signal:

Syntax: *esignal <eventname>*

Enables the user to signal that an event has occurred. The named event will have its signal increment applied to its value. The first process waiting for the event will be activated if there is a queue of processes waiting for the event.

Event Wait:

Syntax: *ewait <eventname>*

This utility simply waits for the named event to occur, informs the user and then terminates.

8.8 SUMMARY

This chapter has detailed the design, structure and coding of selected elements of UMC run time software created by the author. The complete proof of concept UMC system, reviewed in chapter seven, was coded predominantly by Mr. Alan Booth and the author with contributions from Goh [20] and Wright [21]. This work culminated in the release of UMC version one in January 1990 [2]. A critical evaluation of the performance and industrial potential of the current UMC implementation is provided in chapter nine.

Chapter 8: References

1 Anon., "Advanced Architecture for Universal Machine Control", SERC Progress Report, No. 2, Modular Systems Group, Manufacturing Engineering, Loughborough University, October 1990.

2 Booth A.H., "UMC Version 1 Release Notes", Modular Systems Group, Manufacturing Engineering, Loughborough University, January 1990.

3 Dibble, P., "OS-9 Insights: An Advanced Programmers Guide to OS-9/68000", Chapter 2: Modules, Microware Systems Corporation, Des Moines, Iowa, November 1988, pp 13-30.

4 Dibble, P., "OS-9 Insights: An Advanced Programmers Guide to OS-9/68000", Chapter 10: The I/O Manager, Microware Systems Corporation, Des Moines, Iowa, November 1988, pp 110-111.

5 Puckett, D.L. and Dibble P., "The Complete Rainbow Guide to OS-9", Chapter 26, Falsoft, Kentucky, 1985, pp 272-273.

6 Anon., "The OS-9 Catalog", System-Level Extensions: Internet Support Package (ESP), Microware Systems Corporation, Des Moines, Iowa, 2nd. Edition, 1989.

7 Anon., "Microware Basic User's Manual", Revision D, Microware Systems Corporation, Des Moines, Iowa, October 1986.

8 Ward P.T. and Mellor S.J., "Structured Development of Real-Time Systems, Volume 1, Yourdon Press, Prentice-Hall, 1985.

9 Orr K. et al., "Methodology: The Experts Speak", Byte, April 1989, pp 221-233.

10 Rock, S.T., "Developing Robot Programming Languages Using an Existing Language as a Base - A Viewpoint", Robotica, Vol. 7, pp 71-77, 1989.

11 Smith, J.D., "Reusability and Software Construction: C and C++", John Wiley, 1990.

12 Puckett, D.L. and Dibble P., "The Complete Rainbow Guide to OS-9", Chapter 29: Modules, Falsoft, Kentucky, 1985, pp 283-297.

13 Anon., "Designing Real-Time Control Systems Using OS-9", Microware Systems Corporation, Des Moines, Iowa 50322, July 1984.

14 Puckett D.L., "The Basic09 Tour Guide", Microware Systems Corporation, Des Moines, Iowa, 1984.

15 Anon, "C Programming Language Manual", Microware Systems Corporation, Des Moines, Iowa.

16 Kernighan, B.W. and Ritchie, D.M., "The C Programming Language", Prentice-Hall, 1978.

17 Anon., "OS-9/68000 Operating System Technical Manual", Chapter 15: Events, Microware Systems Corporation, Des Moines, Iowa.

18 Allworth, S.T., "Introduction to Real-Time Software Design", Chapter 2: The Real-Time Virtual Machine, Macmillan, 1981.

19 Capouch, B., "Basic-09", Byte, April 1984, pp 344-347.

20 Goh, A.S., "A Pneumatic Servo Axis Handler for Universal Machine Control", MSc Thesis, Loughborough University, September 1989.

21 Wright, A. C., "Universal Machine Control - Development of an Input/Output Software Module", MEng Report, Loughborough University, Department of Manufacturing Eng., 1988.

9
UMC Evaluation and Industrial Potential

9.1 INTRODUCTION

The Modular Systems Group at Loughborough have written a considerable amount of software in order to enable the principles of UMC to be tested and demonstrated. Many of the concepts embodied in the UMC approach have been derived in an iterative manner. Initial versions of the software have been created based on the requirements presented in chapter five, tested and subsequently enhanced through utilising the machine hardware installed at the University.

This chapter considers the capabilities of the current implementation of UMC, its perceived advantages, limitations, industrial potential and possible routes to its exploitation.

9.2 MACHINE ARCHITECTURE

9.2.1 Introduction

The current implementation of UMC enables an outline specification of the required system configuration to be entered directly by the user into a simple data base. Machine controller description is therefore largely self documenting.

The current data base package is custom written as described in chapter seven. In view of the need for standardisation and the wider potential of the UMC approach as part of a more comprehensive machine design and life cycle environment the use of standard data base management system is seen as essential in the future. See chapter ten.

9.2.2 Configurablity and Component Reuse

Machine controller functionality is defined in a modular fashion and common software components, for example device handlers, diagnostic tasks, user inter-

face tasks etc. can be reused many times. Changes in machine structure can therefore be made at any point in the design/life cycle with little penalty because the various controller software components are essentially independent.

UMC controllers are thus inherently configurable so that they can be easily customised to provide only the minimum functionality consistent with achieving the required task in an efficient manner. This approach enables wide variations in system size to be accommodated efficiently and allows simple systems to be implemented cost effectively whilst still offering a logical method for expansion.

9.2.3 Staged System Development

UMC control system development can occur in stages with the primary system tasks created and tested first. Each task program can be tested as it is generated by triggering the events that control its operation and utilising dummy device handlers as necessary. In this way a prototype can grow progressively into the final system. As each additional component is introduced its effect can be directly observed.

9.2.4 Dynamic Reconfiguration

Changes in UMC configuration can be carried out dynamically without interrupting a machine's operation. In a limited manner machine components and even complete machines can be loaded and unloaded dynamically while the remainder of the system is running. Future extensions to UMC are expected to progressively increase the facilities for dynamic reconfiguration [1].

9.2.5 Information and Event Visibility

A memory resident information structure is inherent in every UMC controller. This provides a distinct information view of UMC and the extraction of management or diagnostic information from any desired part of the control structure is therefore easily achieved.

In a similar manner the state of significant events can be monitored via the globally visible event system. This facility can provide valuable timing data for trend analysis, machine optimisation etc.

The UMC approach thus inherently allows for the provision of diagnostics, reporting and error checking etc. Reusable task program modules could be specifically developed in the future to fulfil these functions.

9.3 TASK DESCRIPTION

9.3.1 Introduction

UMC is capable of supporting extensive multi-tasking at application program level. The use of multiple tasks allows the simultaneous supervision of different subsets of a machine's motion or binary I/O devices in a very flexible manner. Separate tasks can be employed for diagnostics, data collection, user interfaces etc. This leads to reduced individual task complexity through a division of the total problem into a set of distinct program modules and provides inherent potential for extendability, configurability and reuse.

9.3.2 Language Usage

To date software has been written to support the creation of UMC task program modules in the Microware Basic and C languages. C forms the UMC system implementation language and is thus the base level for applications programming. Microware Basic is a very high level general purpose programming language to which UMC specific C subroutine modules have been added. With its interactive debugging facilities but more restricted language capabilities, Basic provides a potentially more productive, although less versatile, programming environment than C [2].

Following analysis and practical evaluation neither of these general purpose programming languages is seen as a preferred solution for the task programming of UMC. As Bristol suggests [3] general purpose languages such as C and Basic have developed without the type of user interface and the ease of use required for industrial machine control.

Both Basic and C mix quite different computational functions freely. For example declarations can, in principle be mixed with computational statements and I/O functions. Most recognised analysis and design methods however totally separate these functions [4]. A well engineered language ought to visibly separate different functions while integrating their effect in a well-meshed environment.

Microware Basic and C emphasise the use of structured programming building blocks: through the provision of sequence, if-the-else, loop and case statements. However in common with most general purpose programming languages there is no means for making the program structure stand out. It is difficult to separate program structure and function or to view a program interactively at different levels of granularity or hierarchy.

These limitations are compounded in the case of the UMC system by the need to visualise the interaction of concurrent task programs.

9.3.3 User Defined Task Structure

The *UMC task structure is entirely user defined* which although providing many potential benefits requires careful consideration and a disciplined design approach to avoid poor application structure and inefficiency. The very flexibility provided by task concurrency unfortunately often causes problems with application visualisation. Software description, modelling and design tools are however emerging which if properly integrated into the UMC system could significantly improve task program structure and visualisation.

It is likely that a block structured, concurrent function description environment for example Grafcet, may best meet the needs of UMC in a standardised manner. It provides a graphical method for concurrent task description which aids application visualisation and will enable the transparent integration of existing UMC function libraries. See section 4.5.

CASE tools could also significantly aid application development and ensure a disciplined design approach providing consistency through the provision of appropriate modelling techniques and data dictionaries [5], [6]. See section 4.4 and chapter ten.

9.4 INTERFACING

9.4.1 External Device Control

Handlers have currently been implemented to enable the interfacing of motion and binary I/O devices in a unified manner. Different types of motion control hardware employing different power media have been extensively tested on the Uni-

versity demonstrator system. Local parallel bus and remote serial interfaces have so far been implemented [7]. Networked handlers are the subject of current research and will be based on emerging fieldbus standards [8].

Handlers allow local, remote and even simulated devices to be controlled in an interchangeable manner. Controlled devices can thus be distributed as required to cater for the varying physical needs of each application be it chemical plant, a packaging machine, an assembly manipulator etc.

9.4.2 Motion Control

The system provides a novel, flexible method for multi-axis motion description. Any subset of a machine's axes can be associated together as a group. The group can be controlled to move between named multi-axis locations from its associated task program. Once defined groups may be extended simply by editing the associated location information module without the need for task program revision.

The current UMC axis handlers provide point to point motion control with the option of defined velocity profiles when these are supported by the motion control devices used. The UMC system does not currently allow for precise synchronisation of multi-axis movement. Only relatively close co-ordination of the start of moves is available. Thereafter moves are currently executed under the control of the individual motion controllers in an unsynchronised manner. Task level motion instructions are therefore at present restricted to point to point commands [7].

Interpolation, contouring and programmable transmission capabilities are not currently available and need to be incorporated into the system in the future although the provision for the addition of such capabilities is intrinsic in the UMC approach. The addition of programmable transmission capabilities originally developed by Quin Systems [9] is the subject of current research.

9.4.3 Manufacturing Interface

Network communications currently allow machine creation and monitoring from a remote OS-9 or UNIX based development environment. A consistent run time manufacturing systems interface to UMC machine controllers has however yet to be implemented. Current research is expected to enable the machine module to be used as a shared information repository for communications with other machines and with higher levels in the manufacturing systems architecture.

9.5 REAL TIME PERFORMANCE

9.5.1 Introduction

It must be appreciated that the dynamics of the UMC system are not loop/scan time related as in a PLC, but are related to the operating system and control software overhead. The interrupt driven I/O system, process time slicing etc. mean that while it might be theoretically possible to calculate worst case response times [10], in practise system loading is not generally deterministic in nature. This fact affects the manner in which the UMC system should be applied to certain applications.

9.5.2 Application Suitability

In non-time-critical applications, for example control of the LUT assembly demonstrator, the computational time required between move or binary I/O instructions is negligible, typically much less than 0.1 second, relative to the time required for manipulator motion. This type of application which is not subject to strict deadlines consists predominantly of what will be termed *soft real-time tasks*. See figure 9.1. Such tasks are well suited to implementation *within* the UMC system boundary. It is important to appreciate that the input system is interrupt driven and that task priorities can be dynamically adjusted in order to ensure that a rapid response can still be provided to failure conditions etc [11].

Certain applications include tasks where deterministic performance is an absolute requirement. These will be termed *hard real-time tasks*. E.g. control loop closure or failure monitoring. As illustrated in figure 9.1 such tasks *must be periodically scheduled* and *must terminate before a given deadline.*

Provision for hard real time tasks within the UMC system boundary does not currently exist. The introduction of alarm functions in the latest release of OS-9 [12] does however now provide the basis for their implementation. Using the current implementation platform the performance available using such an approach would however be limited. Alarms make use of the standard OS-9 system clock which usually has a period of 10 msec. Processes requiring more frequent scheduling could therefore certainly not be effectively controlled without operating system modification.

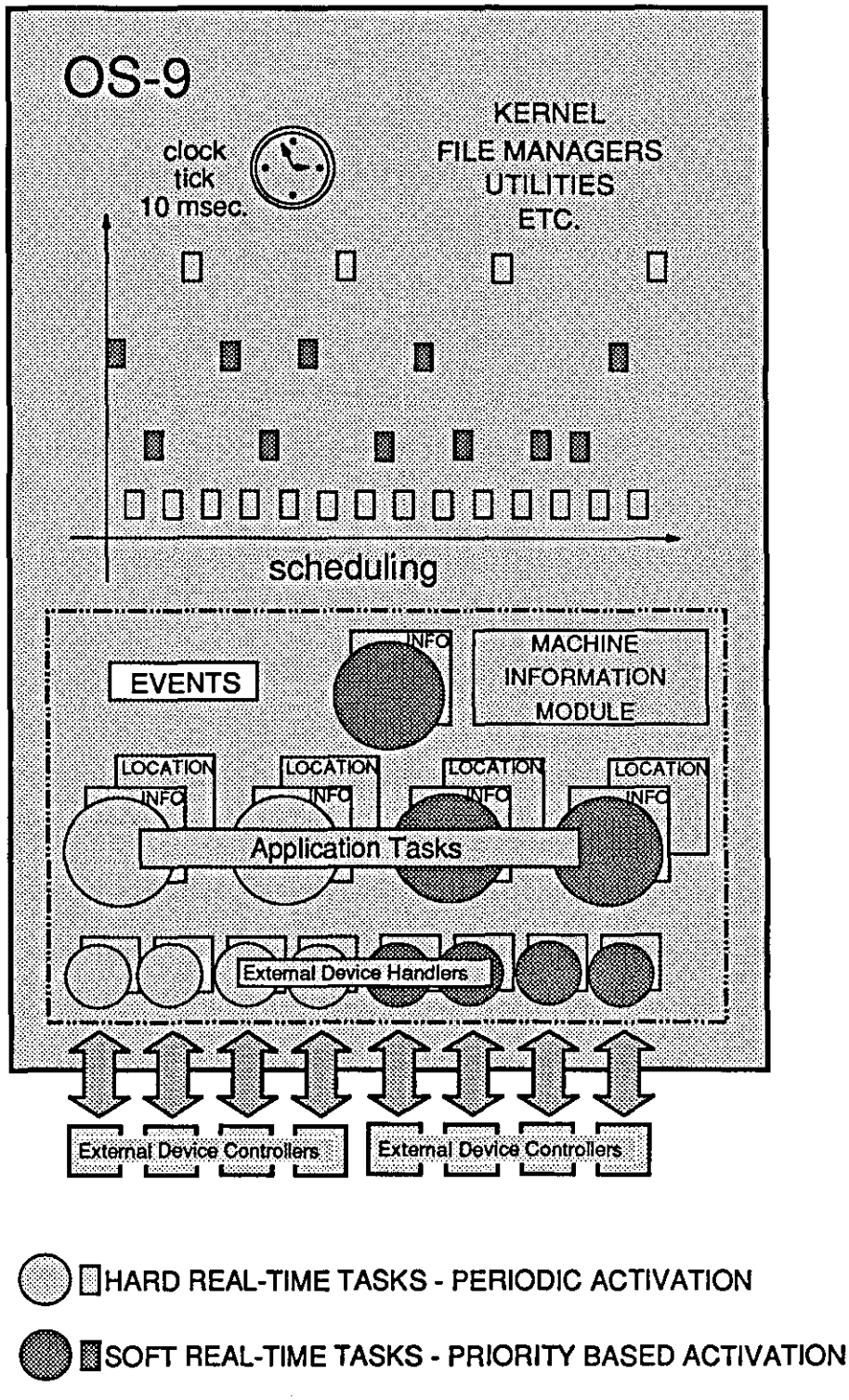


Figure 9.1. Process time slicing for hard and soft real-time tasks.

Performance limitations within the UMC system boundary need not however be a limiting factor on overall control system performance. The use of intelligent I/O devices *supervised* via *the UMC handlers* is the intended philosophy for time critical *hard* real-time processes beyond the performance capabilities of the UMC environment. For example, in the case of high speed synchronised drives, a dedicated motion controller would typically be utilised for each axis of motion. Precalculated position demand profiles may be stored in motion controller local memory and drive synchronisation is achieved heterarchically via a common clock [9]. In this supervisory role the UMC system performs device set-up, monitoring and adjustments via appropriate handler functions. The UMC system's rapid interrupt response can thus be exploited for monitoring and error correction without incurring a large processing overhead. See appendix A.5. The performance limit of external devices interfaced to UMC via handlers is typically due to either their local I/O controller or drive system bandwidth limitations [13]. As advances occur in state of the art external device performance these can be rapidly and consistently interfaced to the UMC system via the implementation of appropriate handlers.

9.5.3 Performance

Wright [14] has performed some limited I/O performance evaluation of the current UMC implementation UMC but a thorough study is yet to be completed. There is obviously a progressive deterioration in system performance as the number of task and handler processes is increased. This need not be a limiting factor provided that a correct assessment is made of where the system boundary should be positioned with appropriate use of internally or externally implemented processes. See section 7.3.

The optimisation of task priorities would enable best use to be made of available processing power within the UMC environment. In order to achieve this the future development of additional software tools is required to enable system performance data to be collected.

The use of a faster supervisory processor and even more significantly a parallel processing environment could be used to up rate UMC system performance as and when it is required [15], [16].

9.6 PORTABILITY

The UMC software has been written to run under the OS-9/68000 operating system. Many features of OS-9 particularly those for interprocess communication have been utilised. See appendix A. Several similar real-time operating systems are now available as discussed in section 6.9. It would be possible to rewrite the UMC software to suit other operating systems but this would obviously be a considerable task. The software has been developed on Syntel computers using the Motorola MC68000 family of processors but could be ported to other computers using the OS-9 or OS-9000 operating systems with little effort. The code has been written in C with the use of OS-9 and UNIX compatible function libraries. See section 6.14.

9.7 INDUSTRIAL POTENTIAL

9.7.1 Introduction

The features so far discussed in this chapter have been demonstrated in a working system within the University. The next step must be critical evaluation to determine both the industrial acceptability of the approach and its run time performance.

9.7.2 Advantages

Several major advantages should accrue from the use of UMC as a result of standardisation allowing a consistent approach to a wide range of problems.

A tried, tested and well documented library of UMC control system components can be used repeatedly and additional modules can be progressively added to this library. As the automation marketplace is vast, the repeated use of these modular components should facilitate low unit costs and high levels of support and reliability. These attributes which are common in well established business software products should be established and utilised in the field of industrial machine control.

The inherent UMC property of extendability should present a clear advantage over most currently available approaches. System functionality can be progressively extended with new application areas being addressed by the addition of appropriate components. Support and diagnostic modules can be incorporated to serve the requirements of system builders, manufacturing engineers, shopfloor workers,

maintenance engineers etc. The appropriate support modules being included to suit the level of user interface required.

The hierarchical nature of the reference architecture can also provide important advantages when compared with the flat architectures typical of piecemeal system implementations [17]. There is a natural hierarchy in machine control problems and a growing need to extend this hierarchy both upwards to encompass factory control and information systems and downwards to include low level machine components such as mechanical modules, sensors and actuators. It is important to allow specific expertise and methods to be brought to bear on the problem at each level in a well defined manner. UMC inherently provides hierarchical problem segmentation in the form of UMC components whilst as the same time providing an architecture for the effective integration of these components. Currently the situation all to often arises where required control technology and methods exist in isolated implementations which cannot be effectively combined.

The use of a hierarchical data model provides the ability to configure, program and monitor control systems with varying degrees of sophistication in a consistent manner. It also allows inherent system wide information visibility which is illustrated schematically in figure 9.2.

It is during the reuse and modification of existing controllers or the development of subsequent machines that the value of the UMC approach is likely to emerge most strongly. These aspects are obviously difficult to quantify effectively until a wide base of control system manufacturers, system builders and users has been established.

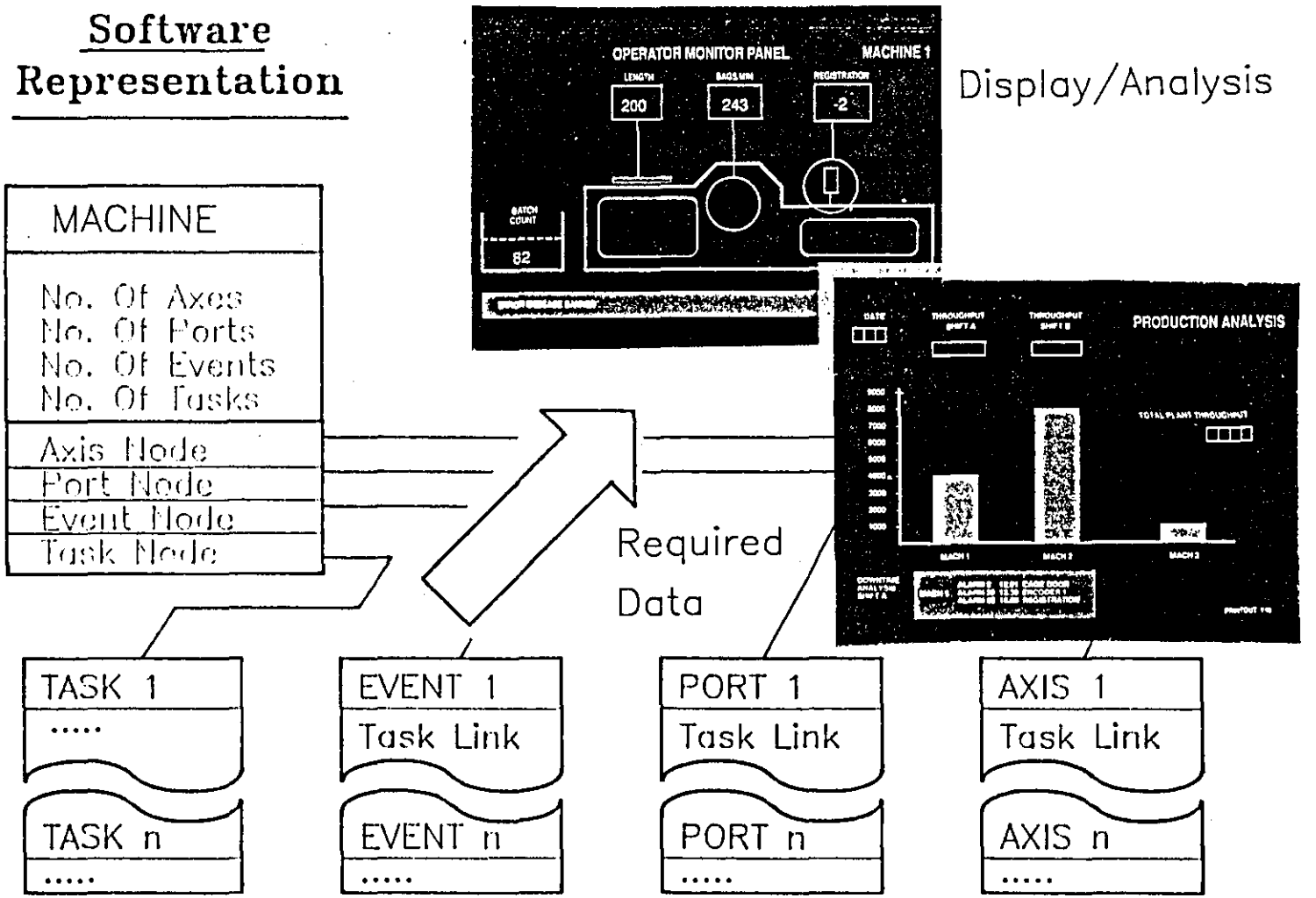


Figure 9.2. Information visibility provided by the UMC system.

9.7.3 Cost Implications

It is important to consider the likely cost implications of the UMC methodology in comparison with more conventional control approaches. Clearly many assumptions must to be made in order to make such a comparison and any findings must be treated with extreme caution until verified by industrial trials.

Peck [18] has looked at the advantages of modularity in robot application software and his evaluation of the effects of a modular approach on the effort expended for each phase of the software development cycle is presented in figure 9.3. The results assume the implementation of all modules from scratch with *no reuse* of existing software modules and the author anticipates that similar cost trends might emerge for initial UMC software development.

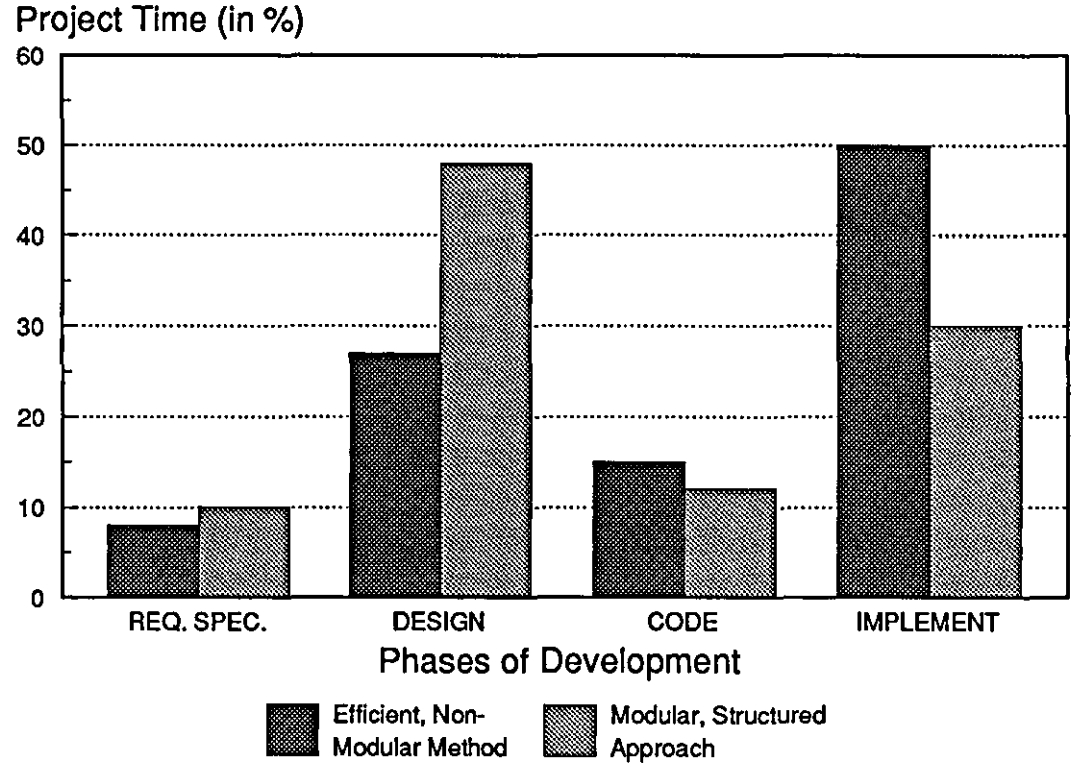


Figure 9.3. Effects of a modular approach on the effort expended for each phase of the software development cycle. Source: Peck.

Peck maintains that the additional cost incurred by using a modular approach during design is more than offset by the reduced manpower cost during the coding and implementation phases. As illustrated in figure 9.3 it is during the implementation/debug phase of the software development cycle that the major immedi-

ate benefits of modularity are realised. This phase represents lost manufacturing output since it involves the use of physical machine hardware and cannot be performed offline. Camp [19] states that a modular approach can reduce implementation costs by between twenty five and fifty per cent in computer systems and similar savings might reasonably be expected in a UMC implemented control system.

In the longer term the potential for the reuse of software components to extend a current machine or create new ones from an established UMC component library is likely to offer even greater potential but is difficult to quantify.

Weston [17] has considered the possible cost implications assuming that widespread acceptance of UMC has been achieved and that consequently a large number of UMC components conforming to the reference architecture are available. When such a situation is reached system builders or manufacturers would be able to select components from various sources to produce the machine control systems they require. Figure 9.4 illustrates the relative cost trends which might then emerge. Figure 9.5 tries to show graphically the potential of UMC to provide greater functionality in comparison with traditional controller types. Functionality in this context means the provision of desired features notably, extendability, reusability, information visibility and reconfigurability. It is considered vital to convey the meaning and potential value of these attributes to control system users.

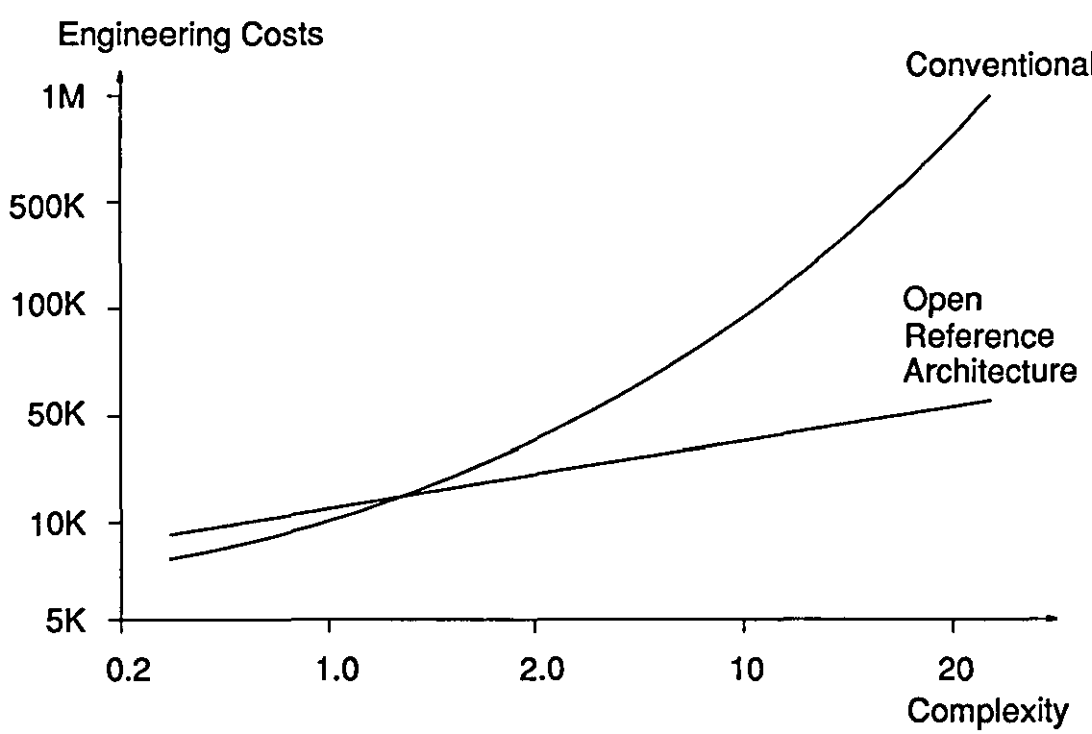


Figure 9.4. Relative engineering costs of conventional and open reference architecture approaches to machine control in a mature environment. Source: Weston.

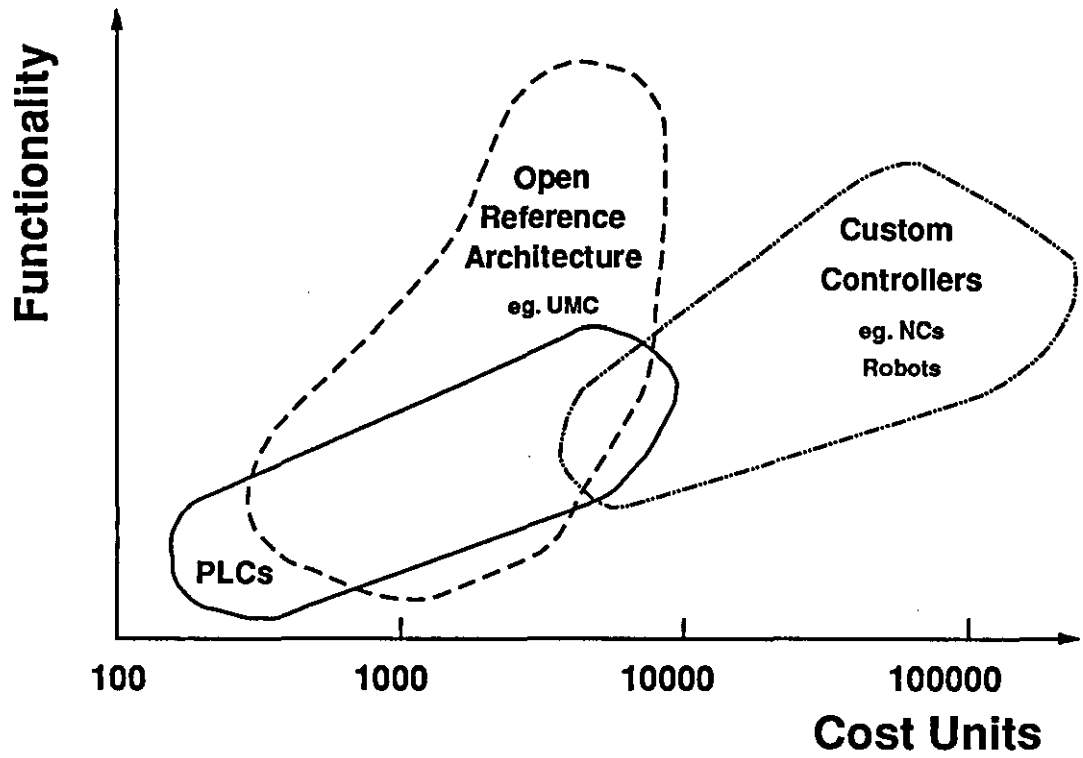


Figure 9.5. Functionality - cost comparison of reference architecture and conventional control approaches. Source: Weston.

9.7.4 Acceptance

The advantages of the generalised approach on which UMC is based seems compelling and indeed a number of groups are adopting similar methodologies in various sectors of the CIM arena. Available information on these systems was presented in chapter four.

Given that such generalised approaches have significant potential it is important to consider why they are not already widely used commercially. This is a complex issue but can perhaps best be answered with reference to implementation issues and the need for an acceptance of the approach by vendors and users.

As described in chapter seven and appendix B the LUT PCB assembly manipulator has provided a practical demonstration of the UMC concepts. It provides a very important tool for the awareness and education of visiting industrialists. We have attempted to use this demonstration system to enable potential users to grasp the underlying concepts and potential of the UMC approach. This is seen as a major

hurdle in what is a very conservative, customer driven sector of the controls market place [20]. See also section 3.4.

The evidence from a number of open days has revealed that the UMC approach is not a readily understood method and needs careful presentation and documentation. Users were found to be generally unaware of the potential for more effective problem oriented approaches to control. See chapter four.

Vendors have a vested interest in their own distinct products. At present closed CIM systems are difficult to customise and maintain [21]. This is inefficient and wasteful but usually ensures never-ending vendor involvement keeping current systems up to date and consistent. An open systems approach may not be viewed to be in the vendors interest if specific vendor involvement is no longer guaranteed. An emerging standard could also be seen as a threat with vendor selection then being made predominantly on a cost/performance basis. This effect has been observed in other software fields most notably the recent attempts at operating system standardisation [22].

9.7.5 Exploitation

Having established a proof of concept implementation of UMC, both the conceptual completeness of the architecture and its general applicability now require evaluation. A number of research and industrial studies are currently being considered to enable the evaluation of UMC in the control of certain classes of industrial machine. It should be possible to generalise the findings of these case studies and revise and/or extend the architecture as necessary.

Categories of machine under consideration include:

(1) Parts handling, assembly, inspection machines involving sequential motions of concurrent axis groups. This type of application could be achieved using the currently implemented UMC components with relatively few additions. A beta-test release of UMC version 1 is being evaluated by Quin Systems for this type of usage [7].

(2) Synchronous machines typically employed in packaging or forming applications. Current research aims to add the functionality provided by the Quin Programmable Transmission System (PTS) to UMC. An SERC-DTI High Speed Machin-

ery research programme has been approved to enable the industrial evaluation of UMC when applied to this category of machine [23].

(3) Applications involving profiling or interpolation, for example machine tools, fettling, cloth cutting. The development of new handlers capable of interpolation is required although this is closely related to the provision for synchronous machine control. See section ii).

(4) Process control systems. This field has not so far been addressed by UMC. Analogue I/O port handlers have yet to be implemented although provision exist for these devices. The implementation of closed loop control *within* the UMC environment using sample data techniques also requires research. The sampling requirements of process control system are however usually relatively modest, with periods typically greater than 100 msec [24]. This means there is potential in this application area for the coding of *hard* real-time tasks within the UMC environment. See section 9.5.

(5) Mixed systems. There are a significant number of applications which require a mix of the above capabilities either within one machine or across adjacent machines. The ability to mix functions in a consistent manner with the opportunity for common monitoring and diagnostic approaches across all machines has great potential in helping to contain the ever increasing complexity of CIM systems. This of course requires the development of an effective manufacturing systems interface and the establishment of a mature set of task and handler components.

To achieve commercially viable UMC products a considerable amount of work remains to be done in order to package the UMC software. The word "package" here refers to the provision of software customised to suit a chosen set of control and mechanical hardware together with the necessary documentation.

Adequate resources are needed to ensure the proper development of a reliable and maintainable software product. As the first stage in the process of commercialisation version 1 of UMC has been released to Quin Systems for their evaluation

[7]. Additions to this package will be required for particular application areas as they emerge. The final packaging chosen must be the result of a commercial assessment of the market carried out by appropriate vendors.

9.8 SUMMARY

The feasibility and value of adopting a generalised approach to machine control has been demonstrated although its industrial utilisation is yet to be achieved.

Based on the requirements presented in sections 2.5 and 5.2, the current implementation of UMC demonstrates a number of significant advantages over conventional control systems; particularly with regard to configurability and reusability. Some limitations exist in the current UMC software due to the need for additional function modules which have not yet been implemented. There are also a number of intrinsic characteristics, notably the user defined task structure, inherent information structure and intelligent device interface handlers, which affect the methods which need to be adopted for the effective utilisation of UMC.

A "standardised" control approach such as UMC can only evolve over a considerable period of time. For the UMC methodology to become industrially established, as the programmable controller is today, will require acceptance of the approach followed by the progressive creation of a considerable base of software conforming to the methodology. It is only at such a mature stage that the major advantages of UMC namely component reuse and system extendability will have most impact. Exploitation is therefore probably best achieved in "targeted" stages, addressing initially the application areas where ease of configurability and integration will have the most impact.

Chapter ten looks at the UMC approach in a broader context as part of a framework for complete machine design and life cycle support.

Chapter 9: References

1 Booth, A.H., "Changes to UMC", Internal release note, Modular Systems Group, Manufacturing Engineering, Loughborough University, 20 July 1990.

2 Anon., "Designing Real-Time Control Systems Using OS-9", Microware Systems Corporation, Des Moines, Iowa 50322, July 1984.

3 Bristol E.H., "How Control Languages Should be Designed with and Integrated Example design", Proc. of the 13th IFAC/IFIP Workshop on Real Time Programming, West Lafayette, Indiana, 1985, pp 93-96.

4 Ward, P.T. and Mellor, S.J., "Structured Development for Real-Time Systems", Vol 1, Yourdon Press, 1985.

5 Gibson, M.L., "The CASE Philosophy", BYTE, April 1989, pp 209-218.

6 Frost, S., "Making the best method mix", Microsystem Design, Frost, S., "Making the best method mix", Microsystem Design, February 1990, pp 14-15.

7 Booth A.H., "UMC Version 1 Release Notes", Modular Systems Group, Manufacturing Engineering, Loughborough University, January 1990.

8 Armstrong, N., "Networked UMC Handler Concepts", Modular Systems Group internal progress document, Manufacturing Engineering, Loughborough University, 1990.

9 Anon., "Programmable Transmission System Reference Manual", Quin Systems Ltd, Wokingham, 1990.

10 Lent, B., "Dataflow Architecture for Machine Control", Chapter Four: Performance Analysis of Computer Architectures Deployed in Machine Embedded Control Systems, Wiley 1989.

11 Dibble, P., "OS-9 Insights: An Advanced Programmers Guide to OS-9/68000", Chapter 6: Process Scheduling, Microware Systems Corporation, Des Moines, Iowa, November 1988.

12 Anon., "The OS-9 Catalog", Microware Systems Corporation, Des Moines, Iowa, 2nd Edition, September 1989, pp 36-38.

13 Pu, J et al., "Architecture/Software Design for Machine Control", Submitted for Publication September 1990.

14 Wright, A.C., "Universal Machine Control - Development of an Input/Output Software Module", MEng Report, Loughborough University, Department of Manufacturing Eng., 1988.

15 Dibble, P., "OS-9/MP", European OS-9 Conference, Luton, 1987.

16 Anon., "OS-9000", Ref. POS-NANA-NASL, Microware Systems, Des Moines, Iowa 50322, September 1989.

17 Weston, R.H. et al., "A New Approach to Machine Control", Computer-Aided Engineering Journal, February 1989, pp 27-32.

18 Peck, R.W., "Advantages of Modularity in Robot Application Software", General Electric Company, Orlando, Florida, 1987.

19 Camp, J.W. and Jensen, E.P., "Cost of Modularity", Proc. of the Symp. on Computer Software Engineering, 1976.

20 Muir K., "Stating the CASE for Industrial Automation", Drives and Controls, June 1990, pp 22-24.

21 Pan, Y.C., "A Framework for Knowledge-Based Computer-Integrated Manufacturing", IEEE Transactions on Semiconductor Manufacturing, Vol. 2, No. 2, May 1989, pp 33-46.

22 Falk, H., "UNIX Rivalries Cloud Developers' Choices", Computer Design, March 1989, pp 49-54.

23 Weston R.H. et al., "Derivation of a Generalised Approach to Designing Control systems for High Speed Machines", SERC Research Grant Application, February 1990.

24 Higham, E.H., "An Assessment of the Priorities for Research and Development in Process Instrumentation and Process Control", Institution of Chemical Engineers, 1988.

10
Future Extensions to the UMC Methodology

10.1 INTRODUCTION

The basic premise of the UMC methodology is that many machine applications have broadly overlapping requirements which can be effectively catered for by a generalised approach. UMC currently enables the implementation of machine *controllers* in a generalised manner. In a broader context the UMC methodology could be naturally extended to create a framework capable of supporting a generalised approach to *total* machine design with proper provision for manufacturing systems integration.

10.2 INTEGRATION OF UMC MACHINES WITHIN A COMPLETE MANUFACTURING SYSTEM

10.2.1 Introduction

The concept of integrating computer based devices and machines to improve the efficiency of manufacturing is widely accepted. What is less generally agreed is what integration means in practice and how it can best be achieved [1].

10.2.2 Device Integration

The integration of *specific* proprietary building elements within the *generalised* UMC machine control architecture has been achieved through the creation of device handlers. These handlers currently enable proprietary *Axis* and binary I/O *Port* devices to have a unified appearance within the UMC environment. As discussed in section 9.4.1 handlers allow local, remote and even simulated devices to be controlled interchangeably. The types of handler could now be extended to encompass other devices for example visions systems, tactile sensors etc. This would enable much greater I/O functionality to be achieved in a consistent manner. The concept of integrating advanced motion, and sensing functions in an open control architecture based on extensions to the UMC methodology has formed the basis

of a recent ESPRIT proposal. This research application, which suggests a generalised approach to processing, manipulation, mobility and perception, has been made by the Modular Systems Group in collaboration with other industrial and academic partners from Britain, France and Germany [2].

10.2.3 Machine Integration

As discused in section 9.4.3 although network communication currently allows the creation and monitoring of UMC machines, a consistent run time manufacturing systems interface has yet to be implemented. Indeed the existence of an interconnection facility is only the first step towards integration. Weston considers that applications control and information support are additionally required before integration is truly achieved [1]. See figure 10.1.

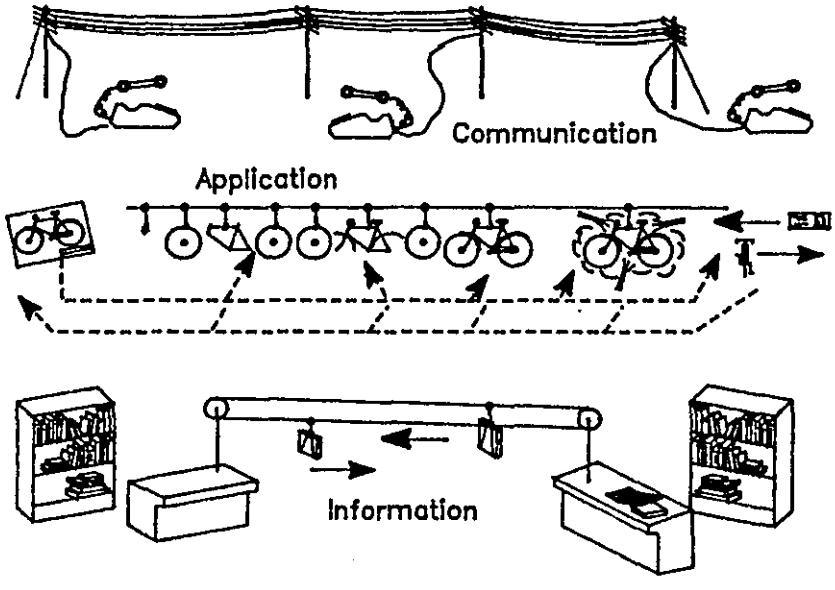


Figure 10.1. Interconnect, control and information support. Source: Weston.

It is anticipated that it will be possible to embody many of the results from research carried out by the Systems Integration Group at Loughborough in the implementation of a UMC manufacturing systems interface. The work of this research group has resulted in the creation of AUTOMAIL (AUTOMAtion Integration Lan-

guage) [3]. This consists of a distributed system application language and a set of configuration, management, and debugging tools specifically targeted at the integration of manufacturing machines. AUTOMAIL is capable of supporting activities such as machine synchronisation, program downloads and management data retrieval. Like UMC which it complements, AUTOMAIL does not seek to impose any particular application structure but provides a supporting set of communication, application and information integration services [1]. It enables "soft" integration between manufacturing entities (typically application processes). Application processes only require knowledge of how to utilise the integration services and do not need to be aware of the other processes in the complete manufacturing system. Association between processes is driven by a data model with enables configurability (ie. responsiveness to change). See figure 10.2.

10.3 A MACHINE LIFE CYCLE ENVIRONMENT

10.3.1 Introduction

Whilst run time control has a major role to play in the creation of advanced industrial machines it must not be treated in isolation. UMC should therefore be viewed as a partially implemented computational framework to support the design and life cycle requirements of machine systems in a manufacturing environment. The capabilities of this framework could be extended to encompass machine design, simulation, work planning, monitoring, diagnostics etc., in addition to the machine control system description and run time support facilities provided by the current UMC implementation. See figure 10.3.

The ability to integrate mechanism and controller design would allow the modelling of their behaviour in a consistent manner in order to determine the optimum combination of the two for each particular application.

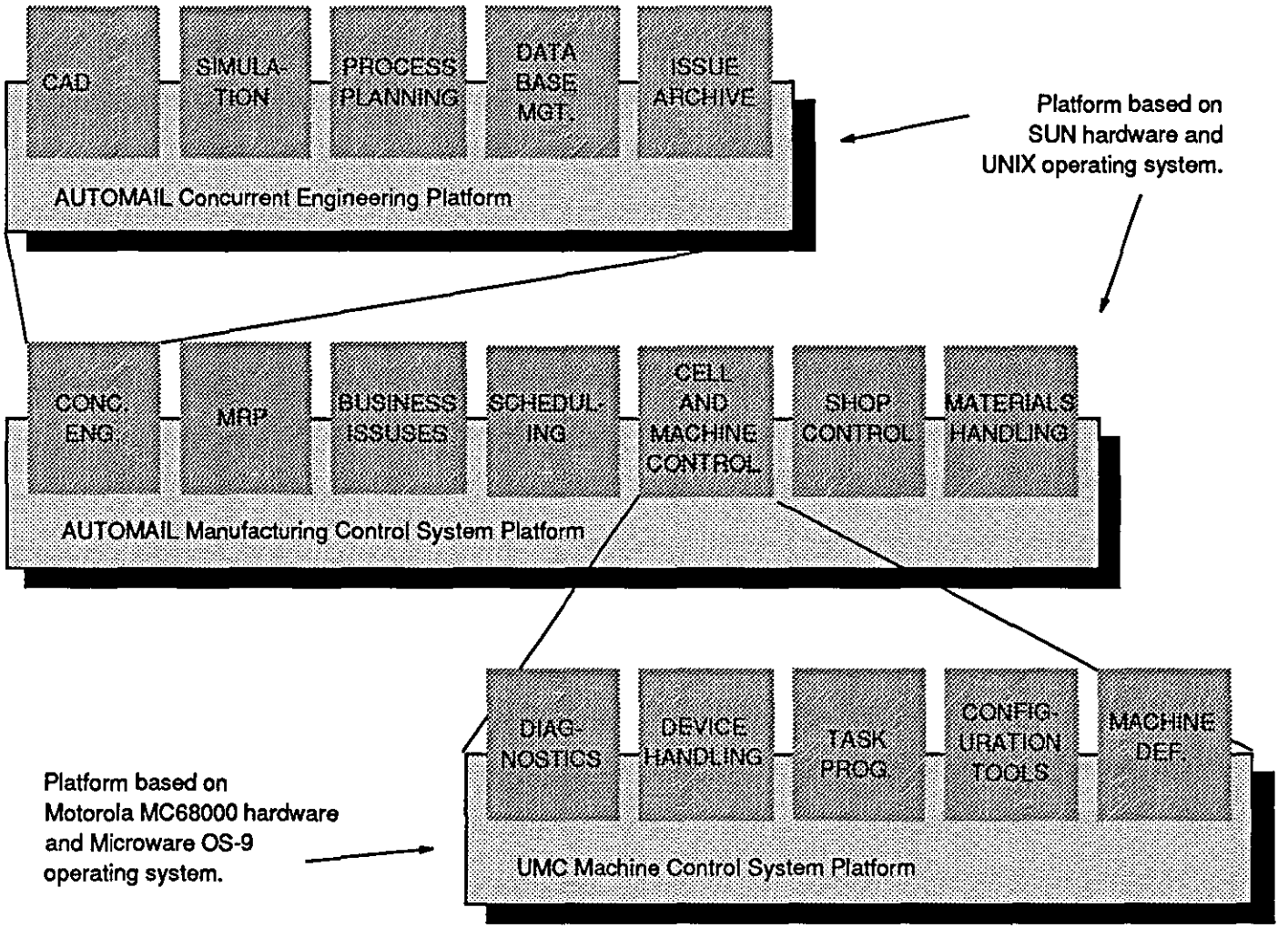


Figure 10.2. Example usage of AUTOMAIL and UMC concepts.

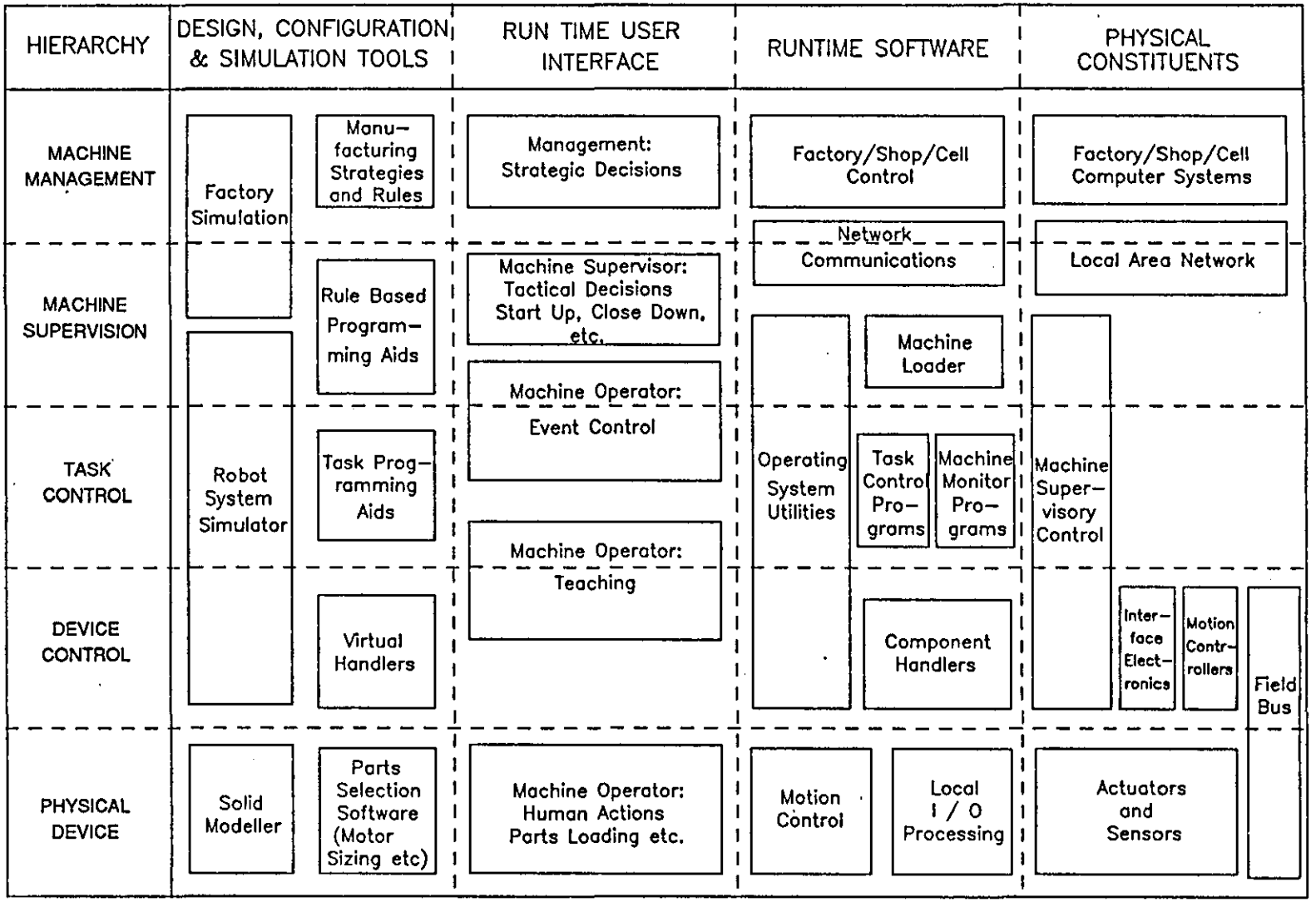


Figure 10.3. The expected features of a mature UMC environment.

10.3.2 Information Requirements

The information required for the different aspects of a machine's life cycle could be combined into one shared unified model for each machine in the form of a centralised knowledge base and machine library. Such a unified machine information base would contain both generalised process knowledge and machine specific information relating to machine and control system design, simulation, monitoring, diagnostics etc. Using structured decomposition (see section 5.4), generic knowledge in any of these spheres might be captured in a suitable form and then utilised to create particular machine instances.

To create a total machine model will require a far more comprehensive information architecture than that currently provided by UMC. The current UMC Reference Architecture describes the control system components and the method of arrangement of these entities to form a hierarchical run time structure. See chapters five and seven. The extended architecture would need to possess the capability to inherit standard design, mechanical, electronic, diagnostic and control components in the creation of new machines.

The design of a given machine should ideally involve as many interested parties as is practical. However these individuals invariably come from different disciplines and often have difficulty in acting together effectively. A common information base which all parties can understand and contribute to in their own terms would have many attractions. The vision is to create a single, common machine model which can be understood by all interested parties and maintained by ordinary plant personnel in a consistent manner. Process engineers, equipment operators, maintenance technicians would utilise a range of interfaces appropriate to their individual requirements. As illustrated in figure 10.4, each user would be provided with a customised view of the information base appropriate to their sphere of knowledge.

A database consisting of appropriate data models, utilising standard information storage and retrieval mechanisms would be required to form the core of the necessary information architecture. Interfaces would need to be provided between the database and the UMC related processes, e.g. modelling packages, machine loaders, editors etc.

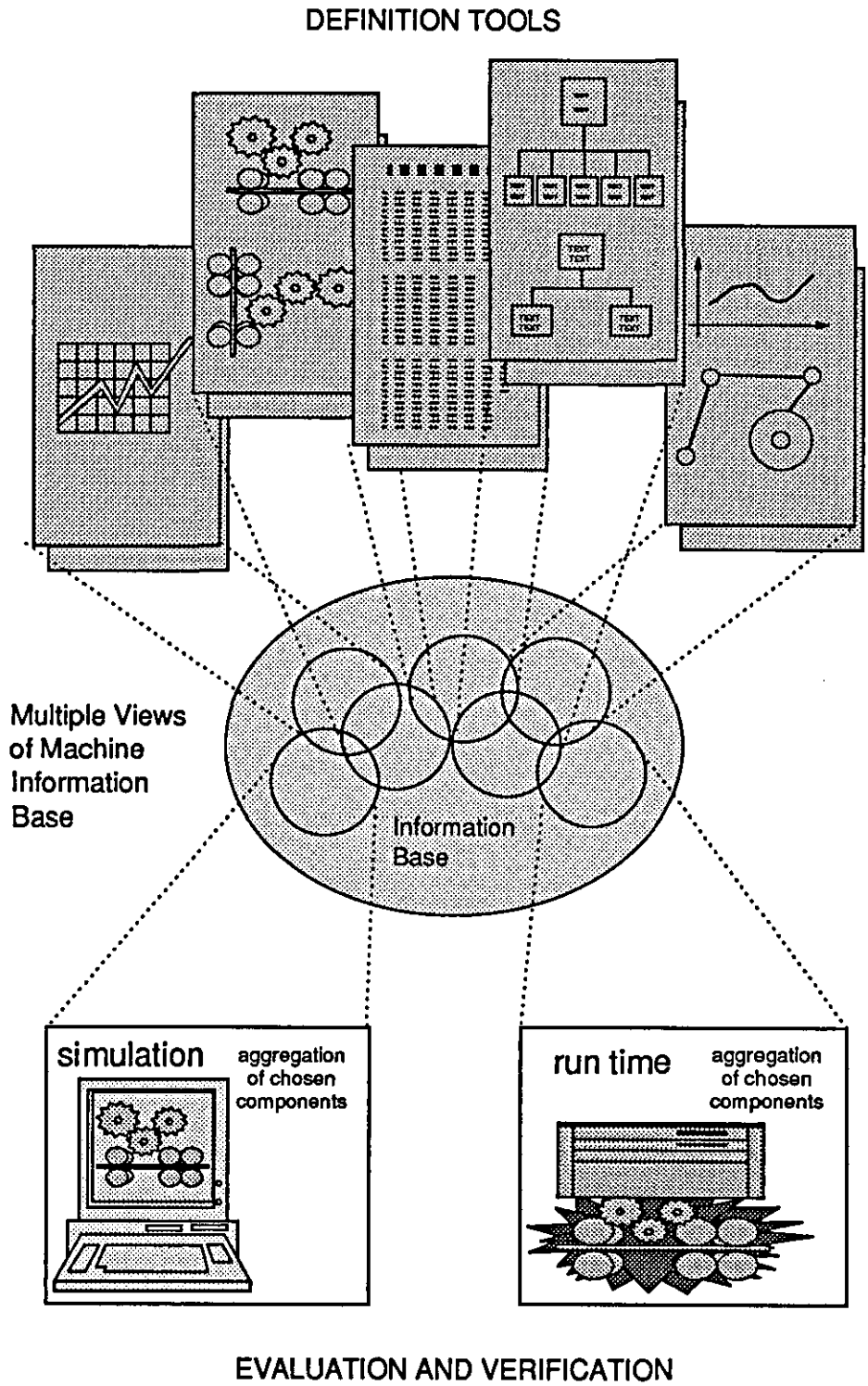


Figure 10.4. A common machine information base.

The provision of an appropriate and maintainable centralised database is thus crucial to the long term success of the UMC approach and is currently under investigation by the Modular Systems Group at Loughborough [4]. As discussed in section 6.12, of the three traditional database structures, hierarchical, network, and relational, the relational approach appears to offer the best fit as an extendible centralised data resource for UMC. The choice will however require careful consideration since this data resource forms the hub of the implementation. It should be noted that the CAD industry has found relational databases to be inadequate in certain respects and they are now turning to object oriented database technology [5].

10.4 MACHINE DESIGN AND LIFE CYCLE SUPPORT TOOLS

The design and realisation of modern computer controlled machines is becoming increasingly demanding and multi-disciplinary in nature. Each aspect of machine design (physical structure, dynamics, kinematics, drive systems, interface systems, control systems etc.) requires a distinct discipline. Mechanical, electrical, electronic and computing skills must however all be combined effectively in order to achieve design success. Typically the performance of the final machine is difficult to predict and design shortcomings are often revealed too late due to misconceptions between members of the design team. This at best leads to a very conservative approach to design both in terms of cost and machine performance.

After installation, maintenance and modification are on-going activities which require consistent support throughout a machine's life cycle. Design requirements may change, modifications and additions may be needed and should be catered for with minimum disruption to the existing system.

UMC aims to address these machine design and life cycle issues through the progressive provision of appropriate software tools. Figure 10.3 illustrates the expected features of a mature UMC environment. Suitable tools which could fulfil *some* aspects of a machine design and life cycle environment are considered in the following sections.

10.5 COMPUTER AIDED SYSTEMS ENGINEERING

If the original system specification or design is wrong, incomplete or ambiguous, then even the most thorough implementation will never overcome these deficiencies. A precise and comprehensive design specification can however provide an agreed target against which achievements can be measured. Most design methodologies are unfortunately extremely laborious to implement manually.

As discussed in section 4.4 CASE tools essentially automate the techniques which all good system developers *should* already use but which invariably prove to be too time consuming.

The potential exists to provide a CASE environment from which the final run time machine could be monitored and its performance compared directly against its original design specification. Virtual Software Factory (VSF) for example is a tool kit which enables the creation of user defined conceptual models, validation rules and graphics/text documents, thus providing automated support for any system design methodology [6]. Products such as the (VSF) could be used for the creation of a UMC CASE environment.

10.6 MACHINE LOGIC DESCRIPTION

The need for improved task programming was discussed in section 9.3. The use of function charts, for example Grafcet, could provide an effective graphical means of concurrent applications task visualisation and description. Grafcet also provides the capability to enable successful functions to be stored and recalled for use in other program modules. A function block typically contains a small number of program statements, sensibly grouped together to initiate an action or to attain a particular program state. See section 4.5.

In the broader context of total machine design there is a need for a common method of task logic description which can be utilised both for run time control of the physical machine and to drive machine models during simulation. The actual program functions called and the executing environments will obviously be different in each case [7].

10.7 MACHINE MODELLING

10.7.1 Introduction

Logical and geometric modelling tools can be employed to create, compare and evaluate machine designs. Studies are currently in progress to add manipulator and mechanism design capabilities to UMC.

The analysis performed by these modelling tools may be numerical as well as visual giving the designer the opportunity to make comparative judgements on alternative designs. Since machine design invariably requires iteration the UMC strategy allows the designers to return to the *aggregation phase* and to specify new component combinations based on modelling and/or run time results until an acceptable degree of optimisation is achieved. See figure 10.5.

10.7.2 Distributed Manipulator Modelling

Distributed manipulators allow the potential for tremendous design freedom but this can in itself cause problems of inadequate design visualisation. The creation of effective kinematic solid modelling tools is therefore of vital importance. See figure 10.6. Constructing a three dimensional solid model and displaying the actions of a machine greatly improves the certainty that a design is correct [7].

Research into the geometric modelling of user defined rotary or linear axis groups is being conducted in collaboration with BYG Systems who produce a successful robotic modelling and simulation package called GRASP [8]. A geometric modeller based on GRASP allows the user to describe each manipulator element from simple shapes. Alternatively machine elements may be extracted from existing libraries. Figures 10.7 and 10.8 illustrate the modelling of selected axes groups from the LUT demonstrator system using GRASP.

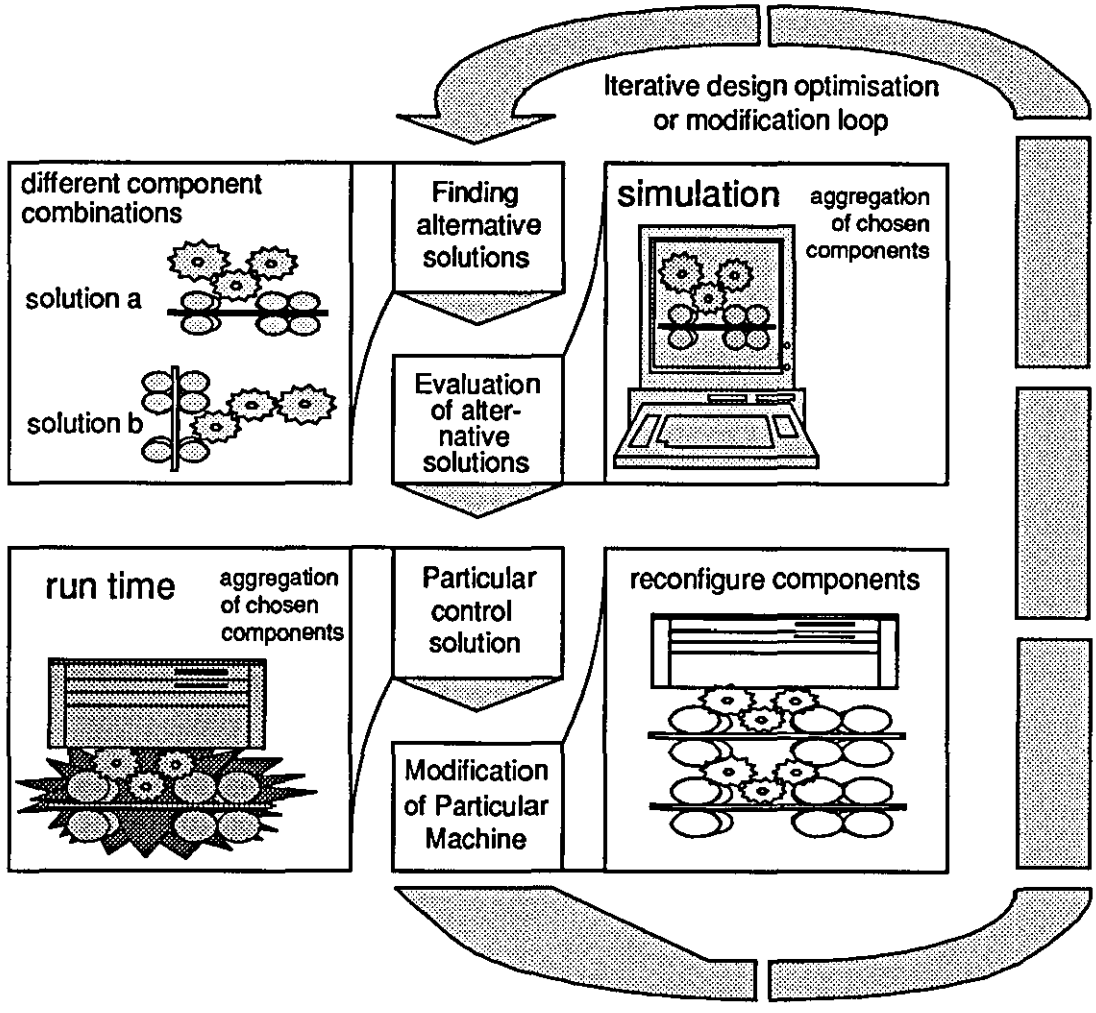


Figure 10.5. Iterative design optimisation.

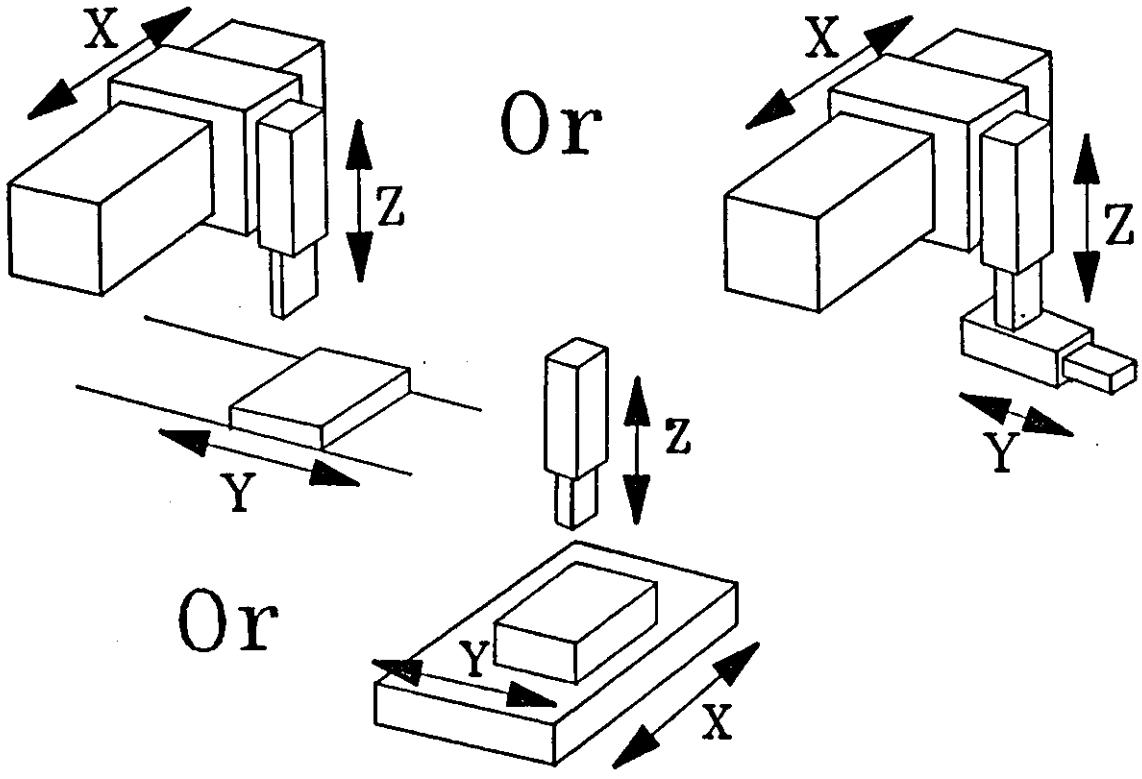


Figure 10.6. Kinematic design tools.

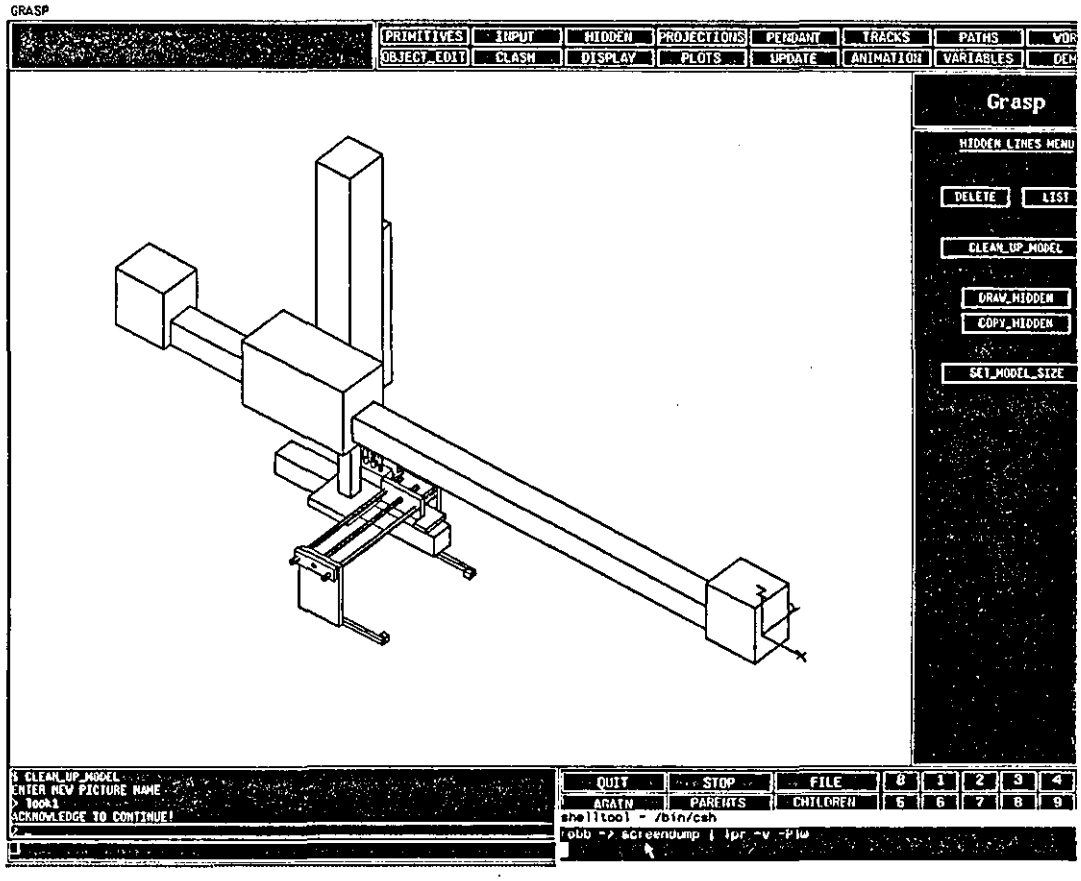


Figure 10.7. PCB handling axis group.

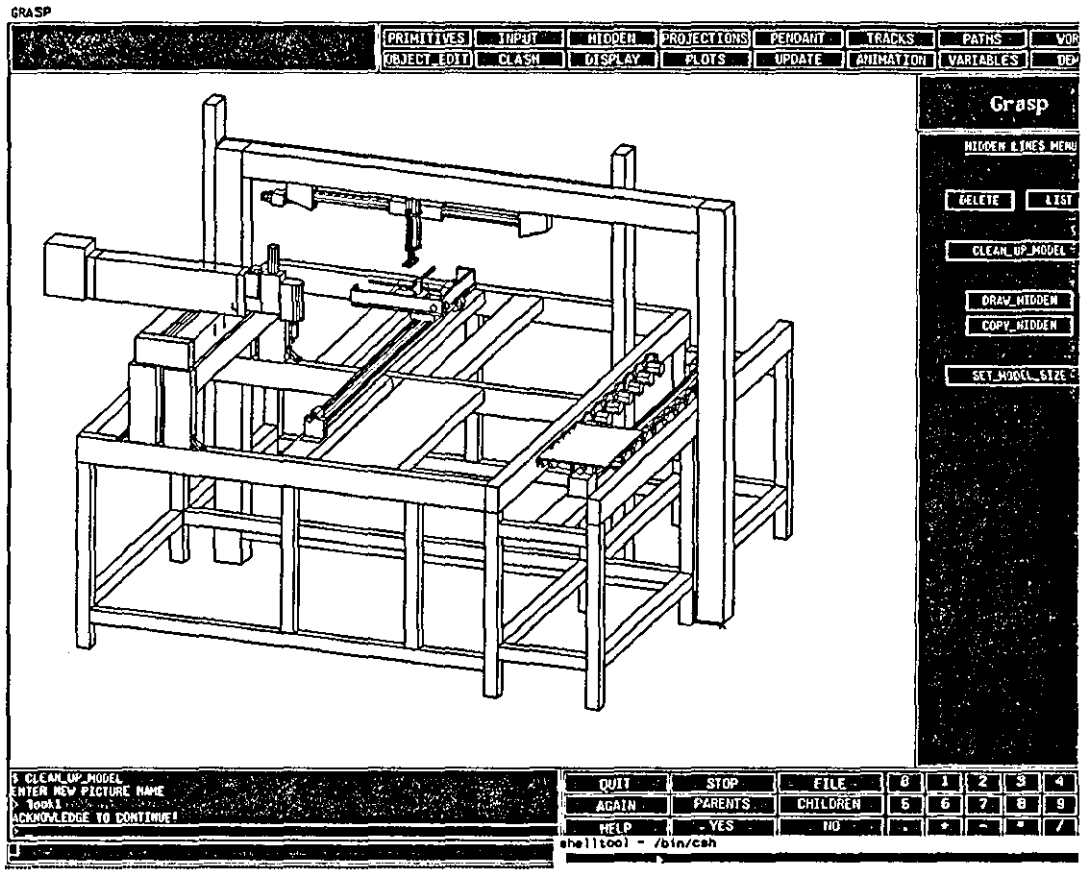


Figure 10.8. Complete LUT manipulator.

10.7.3 Mechanism Modelling

CAMLINKS is a commercial mechanism modelling package which is currently being evaluated for possible inclusion in the UMC environment [9]. It provides two dimensional mechanism design and analysis with facilities for animation. A complementary program called MOTION enables the design of motion profiles in either linear of angular coordinates. MOTION is capable of generating location data for programmable transmission systems. See section 4.5.4.3.

10.8 SUMMARY

The establishment of the UMC Reference Architecture for run time control has been an essential first step in creating a generalised approach to industrial machines. Run time control must not however be treated in isolation. The capabilities of UMC should now be progressively extended to encompass the design and life cycle requirements of machine systems in a broader manner.

Chapter 10: References

1 Weston, R.H. et al., "The Need for a Generic Framework for Systems Integration", in "Advanced Information Technologies for Materials Flow Systems", NATO ASI Series F53, pp 279-309, Springer-Verlag, 1989.

2 Anon., "OSCAR: Open System Control Architecture for Reconfigurable Robotics Machines", ESPRIT II - CIM, Project Area: III.5.1, Submitted to the CEC, December 1989.

3 Weston, R.H. et al., "Integration Tools Based on OSI Networks", AUTOFACT ,89, October 1989, Detroit, Michigan, SME Ref. MS89-708.

4 Goh, A.S., "Minutes of UMC Project Meeting: UMC Design Strategy", Modular Systems Group, Manufacturing Engineering, Loughborough University, May 1990.

5 Edwards, C., "Objects to relate to data", Micro Technology, June 1990, Hanover Press, pp 28-29.

6 Anon., "The Virtual Software Factory: Version VSF 3.4", Software Product Description, Systematica, St. Stephens Road, Bournemouth, BH2 6JL.

7 Doyle R. et al., "The Logical and Geometric Modelling of a Universal Machine Control Architecture", Accepted for Eurotech Direct '91, IMechE Conf.

8 Yong, Y.F. et al., "Off-Line Programming of Robots", in "Handbook of Industrial Robotics", John Wiley, New York, 1985, pp 366-386.

9 Anon., "Camlinks and Motion", Users Manual Version 0.0, Limacon, Meadow Farm, Horton, Malpas, Cheshire, SY14 7EU, 1990.

11
Conclusions

Pursuing the global aim of defining and demonstrating control methods for next generation machines has involved many facets of study. In particular the author has:

- Evaluated the role of machine control systems in a responsive manufacturing environment.
- Advanced the use of problem oriented approaches to machine control system software.
- Reviewed enabling technology in the context of open approaches to machine control, classifying and selecting technological methods and tools.
- Added to the understanding of the role of standards in enabling open approaches to machine control.
- Created a set of prototype programmable modular actuators of novel design together with associated reconfigurable mechanical and electrical interface hardware.
- Assessed and published findings concerning the industrial application of modular machines.

The key conclusions from these studies, which have been considered in detail within the main body of the thesis, are presented in the following sections.

11.1 THE ROLE OF MACHINE CONTROL SYSTEMS

As product life cycles continue to fall, todays manufacturing environment needs to become more responsive to change. An important aspect of improving manufacturing responsiveness is the provision of more flexible production machinery and associated machine control systems. Future machine control systems must embody adequate real-time machine control and integration capabilities. This func-

tionality needs to be provided in a flexible manner through the provision of appropriate design, configuration and run time support environments.

11.2 DEFICIENCIES IN CURRENT CONTROL METHODS

Two factors are now having a major influence on control system requirements:

- Firstly the demands for higher manufacturing quality, greater productivity and responsiveness have in turn demanded more sophisticated automation.

- Secondly hardware costs have dropped dramatically, function for function, while software implementation costs, being labour intensive continue to rise.

Current machine controllers are generally unable to meet adequately the needs of flexible manufacturing, offering limited scope for system reconfiguration or extension and little opportunity for the reuse of software. There is now a need for more efficient engineering methods which can help to coordinate the efforts of the machine or process designer and the control system engineer. Generally speaking however the industrial control environment tends to be more conservative than the data processing and office automation environments. This results in slower rates of equipment replacement with software and hardware developments generally being proven in other fields before they are adopted on the shopfloor.

11.3 PROBLEM ORIENTED APPROACHES TO CONTROL

The exploitation of modern software methods now offers the possibility of maintaining a consistent approach to the control of diverse applications with widely differing machine complexity. The acceptance of problem orientated approaches for real-time control systems has however been much slower than in the case of business systems and the industrial application of these methods has been largely uncoordinated and unstandardised. Reasons for this disparity include:

- Real-time problems are usually more demanding.

- Applications are much more diverse.

- There is generally lower capital investment in new technology in what is considered a very traditional customer driven market sector.

- Users are often unaware of the potential of new methods.

11.4 THE UMC APPROACH

The UMC approach transcends machine control and addresses issues in software engineering. Hierarchical models, multi-tasking and generic software, which are key features of UMC, are not unique concepts. A major contribution of this thesis has been in deriving new methods to combine these tools and to apply them in a usable manner in the context of real-time machine control.

UMC is a generalised approach to machine control which enables software reuse. In order to design effective reusable system components of any type, one must understand how to decompose specific applications into potentially reusable components and how to arrange these components in an architecture to support reuse with minimal changes. The current UMC reference architecture describes control system components and their arrangement to form hierarchical run time structures.

The objectives of the work are to offer not only more cost effective initial control system installations, but moreover to provide a consistent method for both system reconfiguration and the integration of additional functions as new requirements evolve. It was therefore seen as essential to create a methodology which will allow control systems to develop over a period of time and to adapt to evolving requirements and technologies.

11.5 THE ROLE OF STANDARDS

With the emergence of standards for modular hardware and recent advances in real-time software environments, suitable technology currently exists to allow the implementation of the UMC methodology. The ideal of a standard "tool box" which provides an open and consistent approach encompassing all aspects of control system creation is however still largely unrealised. For many aspects of machine control systems effective industry standards for enabling technology have yet to fully emerge. There is also a severe lack of effective applications related standards for machine control which the UMC reference model described in this thesis may help to stimulate.

11.6 IMPLEMENTATION OF UMC

The implementation of the UMC reference architecture at Loughborough has enabled the construction of machine controllers in a generalised manner. Tools have been created to allow the description of required machines and their run time control. Sufficient UMC components have been produced to control target applications involving the use of distributed manipulators for discrete parts handling. The capabilities of the UMC system are however inherently extendable.

The Modular Systems Group at Loughborough has written a considerable amount of software in order to enable the principles of UMC to be tested and demonstrated. Many of the concepts embodied in the UMC approach have been derived in an iterative manner. Initial versions of the software have been created, tested and subsequently enhanced through utilising the industrially representative machine hardware installed at the University.

The UMC software development process is potentially more efficient than more conventional methods, particularly in the long term since there is the opportunity for extensive software reuse across a wide spectrum of control system developers and users. Libraries of machine components can be utilised by machine designers and programmers to reduce the great inefficiency of having to "re-invent the wheel" for each application, saving resources during machine design, control program coding and debug. The configurability inherent in the UMC approach means that installed systems can be very responsive to change, an essential attribute in the modern manufacturing environment. Frequent reconfiguration of the University based UMC machine demonstrator has partially substantiated this claim although such attributes can only be fully assessed through industrial evaluation.

Adoption of the UMC approach has led to the formation of an excellent environment for control system research at Loughborough University and has enabled further research activities to be identified. The Modular Systems Group are currently engaged in SERC-DTI funded research aimed at significantly extending UMC functionality to encompass programmable transmission elements (e.g. software cams, gears and clutches) and more user friendly system configuration programming and diagnostic tools. These University based research programmes are complemented by initiatives aimed at establishing the commercial acceptance of the approach and its industrial utilisation. In the first quarter of 1990 UMC version 1, a software toolbox for control system builders, became available and is currently under evaluation by Quin Systems. The author's role in this work is as an SERC prin-

cipal investigator with special interest in extending and enhancing the areas of knowledge described in this thesis.

11.7 THE FUTURE OF UMC

The potential for the UMC methodology is seen as very broad and consideration has been given to the achievement of adequate real-time performance for a wide range of possible applications. The ability to accommodate change is key to the potential of UMC as a generalised approach to control. Although currently offering relatively modest features it is important to recognise the inherently extendable manner in which the methodology has been implemented. This represents a significant step towards proving the concept of using an "open" control architecture for manufacturing machines.

The basic premise of the UMC methodology is that many machine applications have broadly overlapping requirements which can be effectively catered for by a generalised approach. UMC currently enables the implementation of machine control systems in a generalised manner. In a broader context the UMC methodology could be naturally extended to create a framework capable of supporting a generalised approach to total machine design with proper provision for manufacturing systems integration.

It is expected that the scope of the UMC architecture will be progressively widened to encompass all aspects of the machine's life cycle. UMC should therefore be viewed as a partially implemented computational framework to support the design and life cycle requirements of machine systems in a manufacturing environment

The author firmly believes that the concepts described in this thesis, if not their specific implementation at Loughborough University, will ultimately revolutionise machine control methods.

Appendix A
OS-9: The UMC Implementation Environment

A.1 INTRODUCTION

The chosen operating system OS-9 provides the real-time environment for UMC implementation at a level above the physical computer hardware. See figure 6.6. OS-9 is a system of re-entrant, position independent modules that allow easy system configuration and customising [1]. These features enable it to support control applications of widely differing complexity in an efficient manner, an essential attribute for the implementation of UMC.

OS-9 has enabled the creation of a standardised layered structure of modules which provide the functionality specified by the UMC Reference Architecture. See section 5.6. The real-time operating system allows the dynamic configuration of the UMC system and supports its run time operation.

A.2 MEMORY MODULES

Memory modules are the foundation of OS-9 and key to the implementation of the UMC system. OS-9 enables the structural division of both programs and data into multiple modules. These modules form a consistent memory resident control system structure at run time. Each module whether it contains the operating system kernel, a file manager, UMC programs or UMC data uses a specific memory module format [2].

One of the fundamental principles of OS-9 is that it is a dynamic system. Modules can be added or deleted while the operating system is running, they are both re-entrant and position independent. Re-entrant modules optimise memory usage since only one copy of a program needs to be loaded regardless of the number of processes using it.

The implemented UMC structure is thus inherently reconfigurable. All UMC system components are constructed from sets of modules which are memory resident in the run time controller.

A.3 OPERATING SYSTEM SERVICES

The nucleus of OS-9 is a position independent kernel which serves as a system administrator, supervisor and resource manager. The current UMC software development environment consists of the OS-9 kernel together with its associated "shell" user interface, other utilities and appropriate file managers. A UMC target system controller typically contains a subset of the software modules used in the development environment with only the required services retained.

The OS-9 kernel manages memory and processor time and offers a large number of system services. OS-9 system calls eliminate the need for users to write system management routines for memory, process I/O and file management and thus vastly reduce the effort required to implement the UMC system.

A.4 I/O MANAGEMENT

OS-9 features a modular, unified, hardware independent I/O system that can be expanded or customised. The I/O system utilises a UNIX-like path list notation to define I/O paths. These features enable the physical configuration of UMC I/O devices to be easily changed while retaining a consistent software structure. This enables a variety of external control devices, user interface devices and windowing environments both locally and remotely located to be used interchangeably.

I/O requests perform various I/O functions and are processed in OS-9 file managers and device drivers for a particular device. UMC handlers make use of standard OS-9 device drivers for device specific communication. UMC I/O can thus be easily customised by the addition of a new driver for a new class of I/O hardware (say a new motion or I/O controller), with accompanying descriptors for each instance of that device.

I/O device drivers (in common with all other executable program modules) are re-entrant so one copy of a driver module can simultaneously support multiple devices that use identical I/O controller hardware. For example multiple motion controllers.

Device descriptors are small non-executable modules that provide information that associates a specific I/O device with a logical name, a hardware controller, a device driver, a file manager and a set of initialisation parameters. One device descriptor module must exist for each I/O device in the system. For example each

physical motion controller or I/O device interface with the UMC supervisory processor system.

A.5 INTERRUPT HANDLING

The OS-9 I/O system is interrupt driven and does not require polling [3]. To provide rapid response to events during operation of a real-system it is necessary to pre-empt the processor to service a time critical process. OS-9 provides extremely fast interrupt response. Sample system response times are shown in table A.1.

OS-9 SYSTEM RESPONSE TIMES		
Task Switch	Change active tasks	55.1+ 1.5t* usec.
Interrupt	Time to reach first instruction of interrupt service routine	11.1** usec.

* - t represents the number of tasks in the active process queue, and "t">1

** - Assumes only one driver on a particular interrupt vector

All timings measured on a 20 MHz Motorola MC68020 microprocessor.

Table A.1. Sample OS-9 system response times. Source: Microware.

The UMC interfaces to I/O hardware are implemented in an entirely interrupt driven manner and thus make efficient use of the processing hardware.

A.6 PROCESS MANAGEMENT

OS-9 allows multiple processes to execute simultaneously through task switching facilities supplied by the kernel. Programs run as processes with each process having access to system resources by issuing appropriate service requests to

OS/9. All processes within the UMC system boundary are able to run concurrently in this manner.

OS-9 features a UNIX-like task model that supervises the loading, initialisation and execution of each process while still allowing user configurable timeslicing. A prioritised round-robin scheduler is used to allocate CPU time for each process [4]. As illustrated in figure A.1, at any instant a process can be in one of three states :

Active: The process is active and ready for execution.

Waiting: The process is inactive until a child process terminates or a signal is received.

Sleeping: The process is inactive for a specific period of time or until a signal is received.

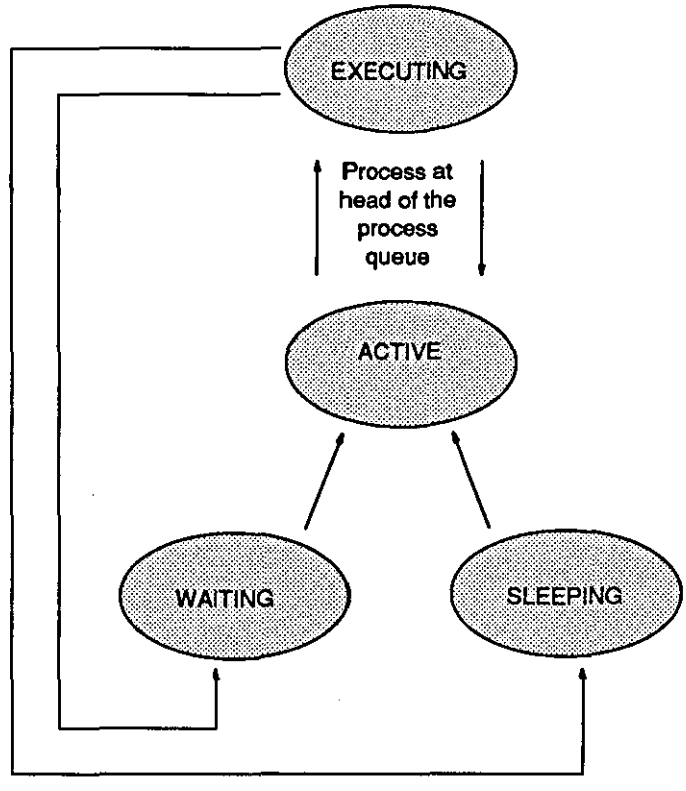


Figure A.1. Process state diagram.

Both the process state and the priority of a process can be defined by the system software from within a UMC process or by user utilities at run time. This provides a method of "tuning" the response of a control system to improve its overall

efficiency. For example handler processes might be executed typically at a higher priority than application task processes.

A typical UMC machine may involve many applications and system related processes. It is important to recognise however that activities within the UMC system boundary will be predominantly event driven and that often the majority of these tasks may be waiting for an appropriate event and thus consuming no CPU time.

A.7 INTERPROCESS COMMUNICATIONS

OS-9 provides four mechanisms for synchronisation and interprocess communication which have all been utilised for the implementation of UMC. They are:

- Data Modules,
- Pipes and Named Pipes,
- Events and Semaphores, and
- Signals.

A.7.1 Data Modules

Data modules are essentially areas of global memory. They are position independent and globally available to all processes that have permission to access them [4]. They are used to create the run time memory resident data structures of the UMC system which are termed UMC information modules.

The information view in the UMC system is constructed from a series of information modules. As discussed in section 5.5.3 this information structure largely mirrors the UMC process structure. UMC information modules contain structured data and access mechanisms have been implemented to provide security and error checking for UMC processes wishing to access this information [5].

A.7.2 Pipes and Named Pipes

Pipes enable concurrently executing processes to communicate data via a first in first out (FIFO) serial device [6]. The output of one process is read as the input by the second process. Named pipes are utilised for buffered communications in

the UMC system for example between task and handler processes. They also provide a versatile debugging aid since they can readily be "emptied" to a user display device during code development.

A.7.3 Signals

Signals are another mechanism provided by OS-9 to control the execution of multiple, asynchronous processes. Signals can only be sent between two processes at any one time. A process may sequentially signal to multiple destinations and a single process may receive signals from more than one process. Signals provide a very narrow buffered channel for processes to communicate. Signals are chiefly utilised in the implementation of UMC for task to handler communications in a similar manner to pipes.

A.7.4 Events and Semaphores

The OS-9 event system differs significantly from data modules, pipes and signals since events only pass control information as opposed to sharing or transferring data between processes. Events do not transmit any information, although the processes using the event system obtain information about events. Events are data structures maintained by the system. The OS-9 event system is a mechanism for permitting processes to share common resources without interference, or for providing sequencing and execution control of asynchronous processes [7].

They are used in by the UMC system processes to control information module access and by the UMC applications processes for general purpose task coordination via UMC function calls and utilities.

Appendix A: References

1 Anon., "The OS-9 Catalog", Chapter 3: A System of Modules, Microware Systems Corporation, Des Moines, Iowa, 2nd Edition, September 1989.

2 Dibble, P., "OS-9 Insights: An Advanced Programmers Guide to OS-9/68000", Chapter 3: Memory, Microware Systems Corporation, Des Moines, Iowa, November 1988.

3 Dibble, P., "OS-9 Insights: An Advanced Programmers Guide to OS-9/68000", Chapter 12: Interrupts, Microware Systems Corporation, Des Moines, Iowa, November 1988.

4 Johnson S. L. et al., "Unix and OS-9 for Distributed and Stand Alone Systems", Proc. of Buscon East, 1988.

5 Booth A. H. "Module Magic Release Notes", (Internal Document), Manufacturing Eng. Loughborough University, 1989.

6 Dibble, P., "OS-9 Insights: An Advanced Programmers Guide to OS-9/68000", Chapter 11: Pipes, Microware Systems Corporation, Des Moines, Iowa, November 1988.

7 Dibble, P., "OS-9 Insights: An Advanced Programmers Guide to OS-9/68000", Chapter 7: Events, Microware Systems Corporation, Des Moines, Iowa, November 1988.

Appendix B
Applications for Distributed Manipulators

Contents

APPENDIX B.1

A Study of the Application Areas for Modular Robots

APPENDIX B.2

The Use of Distributed Programmable Actuators for Flexible Assembly

APPENDIX B.1

Robotica (1987) volume 5, pp 217–221

A study of application areas for modular robots

R. Harrison, R.H. Weston, P.R. Moore and T.W. Thatcher

Department of Engineering Production, Loughborough University of Technology, Loughborough, Leicestershire, LE11 3TU (U.K.)

(Received: March 18, 1986)

SUMMARY

The paper describes a study of potential industrial applications where modular robots can be used in preference to conventional pedestal mounted robots and/or specially designed automatic machines. The study was conducted with reference to a particular family of pneumatically actuated robot modules.

The paper also describes a modular robotic system which was designed to automate assembly processes in the manufacture of a family of egg coddlers. The manipulator comprises multi-axis groups of modules which operate concurrently to perform the assembly operations. Flexibility is incorporated through the use of a hierarchical control system architecture.

INTRODUCTION

The Department of Engineering Production at Loughborough University of Technology (LUT) is engaged in research to design and produce robots of modular design which are pneumatically or electrically actuated.[1,2] This work is sponsored by the U.K. Science & Engineering Research Council.

Since late 1983, Martonair Ltd of the U.K. have produced a range of single degree of freedom modular handling units which provide pneumatically actuated motion to preset mechanical end-stops. The units can be combined to demonstrate user defined kinematics. A three year programme of collaborative research between LUT and Martonair has evolved a complementary range of low cost servo controlled pneumatic modules based on Martonair mechanical hardware.[3] Each servo module utilizes a microprocessor based single axis controller (SAC) to regulate its motion and provide point to point positioning capabilities which are comparable to those offered by electric and hydraulic servo drives (see Figure 1). The SAC may be controlled digitally via either serial or parallel data lines. Commercial versions of the servo controlled positioning system became available in April 1985. The servo modules are used wherever programmable point to point positioning is required and the end stop units where positioning to two preset positions is adequate.

Servo controlled and end stop single degree of freedom modules can be combined together in custom built configurations to suit a wide variety of applications. Combinations of these modular elements can form axis groups which function in a similar manner to conventional multi-axis robots or form the building elements of distributed manipulators, i.e. special flexibly automated equipment in which the axis modules are not necessarily coupled together mechanically but function in an integrated manner to perform given manufacturing operations.

Preliminary investigations have been conducted into the necessary form and features of supervisory controls for multi-axis groups of modules to provide sequencing and user programming. Both at LUT and Martonair microcomputer and PLC based prototype supervisory controls have been constructed to assess the problems of providing appropriate flexibility and performance/cost ratio over the range of potential industrial applications.

Significant research and development work will continue at LUT but successful approaches must be based on adequate applications experience. This paper outlines the results of studies into the potential application areas for modular robotic systems. For chosen industrial applications it illustrates how pneumatic modular elements can be used to offer an automation tool to the manufacturing engineer representing an intermediate approach between the use of dedicated specially designed non-programmable machines (hard automation) and conventional general purpose robots.

OBJECTIVES

The application studies have involved both LUT and Martonair personnel and have been directed towards accomplishing the following objectives:

(i) to categorise potential application areas for the existing family of Martonair/LUT modules
(ii) to evaluate the systems engineering requirements in applying modular elements to industrial applications
(iii) to assess the behaviour and capabilities of servo controlled pneumatic modules under representative conditions
(iv) to investigate, for a range of applications, the requirements for sequencing and programming facilities
(v) to improve the effectiveness of future research into modular systems by more accurately establishing the requirements of industrial applications.

APPLICATIONS STUDY

Here we concentrate on the findings of the application studies conducted by LUT personnel. The study began with a survey of possible modular robot users. Personal visits were then made to companies expressing applications interest and a number of tasks were

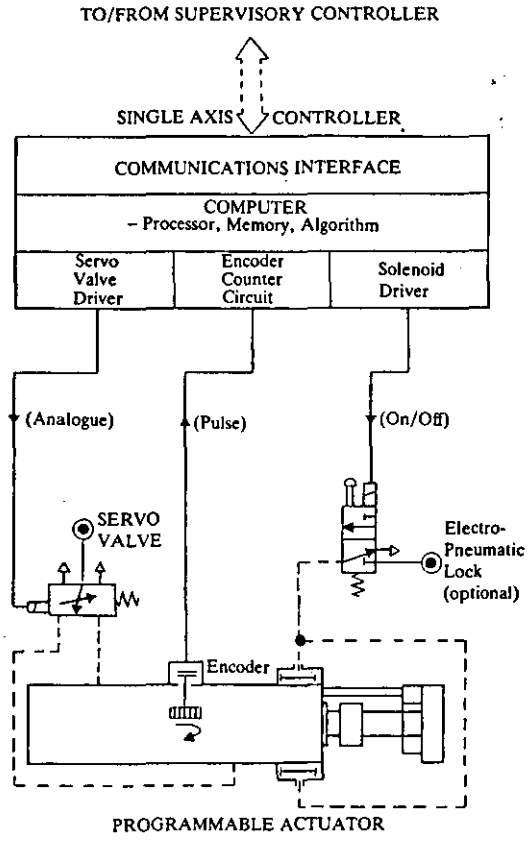


Fig. 1. Hardware elements of a programmable single axis.

identified as being well matched to the capabilities of the LUT/Martonair system.

1. User survey

A preliminary list of possible modular robot users was established (including existing and potential robot end users and also original equipment manufacturers who might be interested in incorporating robot modules into their products or processing systems).

Some fifty companies were contacted of which twelve were willing to take part in a survey and applications study. These companies were sent literature to explain the general capabilities of the present modular system and questionnaires to assess their requirements for flexible automated equipment. Personal visits were made to all the companies expressing applications interest.

Time and resources limited the scope of the applications survey which could be undertaken at this stage. It is intended to distribute this questionnaire to all potential robot users with whom the Department of Engineering Production comes into contact in future. This will form a progressively expanding data base on user requirements.

2 Questionnaire format

The questions asked obtained information from the companies about

(a) their products
(b) the materials handled
(c) the operational requirements of their equipment
(d) how they felt flexible automated machinery/robots could help them now and in the future

Questions were carefully phrased in order not to lead to a specific answer and were in one of two forms, viz:

(i) easy to answer multiple choice questions covering each major aspect of flexible automated equipment usage, design and future requirements
(ii) more general questions requiring brief written answers.

3 Findings from questionnaire

Of the fifty companies contacted, thirteen returned completed questionnaires and all but one of these had direct applications interest in the modular hardware.

The response was poor from those companies without robotic experience, (the very companies who really need more facts about robot technology). Only one of the companies returning questionnaires had not considered the use of industrial robots in the past. Three quarters of the firms were already involved "in house" with robots in some form and all were actively considering their use in the future.

Although the scale of the survey is limited, the results provide an interesting insight into potential user companies and their current production problems.

The most frequently stated reasons for considering or purchasing robots were

(a) improved quality
(b) reduced tedious/repetitive work
(c) increased productivity
(d) ensuring continuous production
(e) to gain flexibility as opposed to a special purpose machine tool
(f) remove men from hostile environments

Major areas of applications interest were quality control, machine loading/unloading and assembly. These were the major labour intensive repetitive tasks in the companies visited. Firms whose manufacturing processes involved significant paint spraying and/or welding requirements had generally already automated these tasks.

Most of the firms used special purpose machinery for production. A third of these companies manufactured some of the equipment themselves and the rest usually bought in machines designed to their specification. It was common to have commissioning and maintenance problems with special purpose machinery and about half the firms experienced long changeover times between product variants. Clearly there is a strong need for more flexible special purpose machinery.

SELECTION OF APPLICATION

For any proposed application of a robotic system, a matching process must be performed between the

functional capabilities of the modular elements and the immediate and potential tasks to be performed.[4–6] This process was carried out in two stages, viz:

(a) assessment of whether the present LUT/Martonair modular system could technically perform the necessary task
(b) a cost analysis to consider how the installation could be justified

The range of possible applications for the present LUT/Martonair family of modules was obviously limited by the specification of the available modules e.g. stroke lengths, load bearing capabilities, positioning accuracy, dynamic characteristics, etc. and the control system facilities. To assist the reader, typical performance figures for a single degree of freedom servo controlled module are listed below.

(i) Positioning repeatability of 0.1 mm.
(ii) Total positioning times of between 1 and 2 seconds for moves between 100 mm and 500 mm.
(iii) Peak speeds approaching 1 m/s.
(iv) Payload capabilities comparable with the equivalent end stop modular elements.[7]

In each application the number of axes required and their configuration, either with or without mechanical coupling, can be optimised to suit the task. The redundant capabilities, often seen where conventional pedestal robots are employed, can be avoided.

POTENTIAL APPLICATIONS FOR MODULAR SYSTEMS

Possible application areas for modular robotic systems in the companies visited by LUT personnel ranged from simple support tasks for existing machinery or conventional robots (i.e. for parts feeding, inspection or as a form of flexible tooling) to complete specially configured systems (with a large number of distributed modular elements operating concurrently). There was a very wide variation in both the payloads and cycle times required.

The LUT applications studies have been complemented by a parallel Martonair activity which has seen the introduction of module/SAC combinations in a range of relatively simple industrial applications involving either one or two servo controlled motions usually supervised by a programmable logic controller. Martonair applications include for example a programmable bar stock feeder for a computer controlled saw, which utilises a single servo axis and a general purpose palletiser with two servo axes for the selection and replacement of items in an array.[8,9]

Applications requiring more complex systems of distributed manipulators were selected for research and development at the university and two early application examples selected were an egg coddler assembly system for the Worcester Porcelain Company and a lawn mower bottom blade machine unload/assembly system for Qualcast Lawn Mowers Ltd. Axis groups to perform these tasks have been configured at the university and are currently being evaluated to assess the system capabilities and supervisory control requirements in each case.

The coddler assembly consists of a small ceramic pot with a threaded metal ring bonded to its lip. Approximately 350,000 coddlers are manufactured per year in three sizes. The required cycle time is approximately 12 seconds per coddler.

The mower bottom blade machine offload and assembly consists of unloading the blade from a CNC milling machine and then assembling it with a pair of end brackets. There are two types of end bracket and four variants of blade. Approximately 300,000 blade assemblies are manufactured per year and a cycle time of 21 seconds is required to match that obtained using a conventional CNC milling machine.

Both of these chosen applications demonstrate concurrent operation of mechanically decoupled axis groups.

Due to space limitations this paper will concentrate on describing the simpler coddler assembly system.

CODDLER ASSEMBLY SYSTEM

A schematic of the manipulator design is shown in Figure 2. The major mechanical elements of the system are three servo axes, three end stop axes, two grippers, the ring dispensers and the adhesive applicator. It was proposed that the final industrial system would include the automatic conveyor loading and unloading of pallets.

The system exploits programmable point to point positioning to load and unload the coddlers from their pallets at variable pitch to suit the three coddler sizes. Variable intermediate positions on the linear module axis C enables collection and adhesive application onto each of the three sizes of ring.

The tasks carried out by the manipulator are listed below, the necessary servoed motions being indicated in each case.

Task 1 Remove the coddler from pallet (Servo axes A and B)
Task 2 Collect the appropriate metal ring (Servo axis C)
Task 3 Apply an adhesive bead to the groove in the metal ring (Servo axis C)
Task 4 Assemble the ring and coddler body (Servo axes B and C)
Task 5 Replace the coddler assembly in the position from which it was removed (Servo axes A and B)

Tasks 1 and 5 occur concurrently with tasks 2 and 3 with the restriction that the movements of axes B and C are co-ordinated at one point in their respective cycles to achieve task 4.

System architecture and control requirements

Figure 3 shows the control architecture of the coddler assembly system.

The control system has a modular multiprocessor structure with the control functions being distributed at two levels:

(1) the upper supervisory level (LUT applications software runs on a SYNTEL 680 microcomputer and operating under OS-9/68000) carries out the general management of the system as well as co-ordinating the tasks defined in the applications programme.[10]

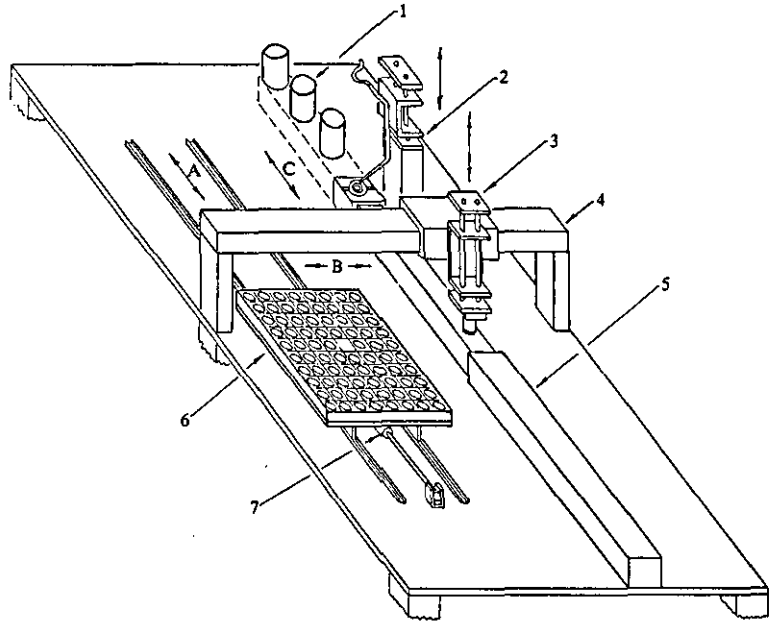


Fig. 2. Schematic of the major mechanical elements in the coddler assembly system. *Key*: A, B and C. Programmably positioned axes; 1. Ring dispensers; 2. End stop unit for adhesive applicator; 3. End stop unit and coddler gripper; 4. Servo controlled gantry unit; 5. Servo controlled linear module with rotary wrist and ring gripper; 6. Pallet of coddlers; 7. Servo controlled cylinder.

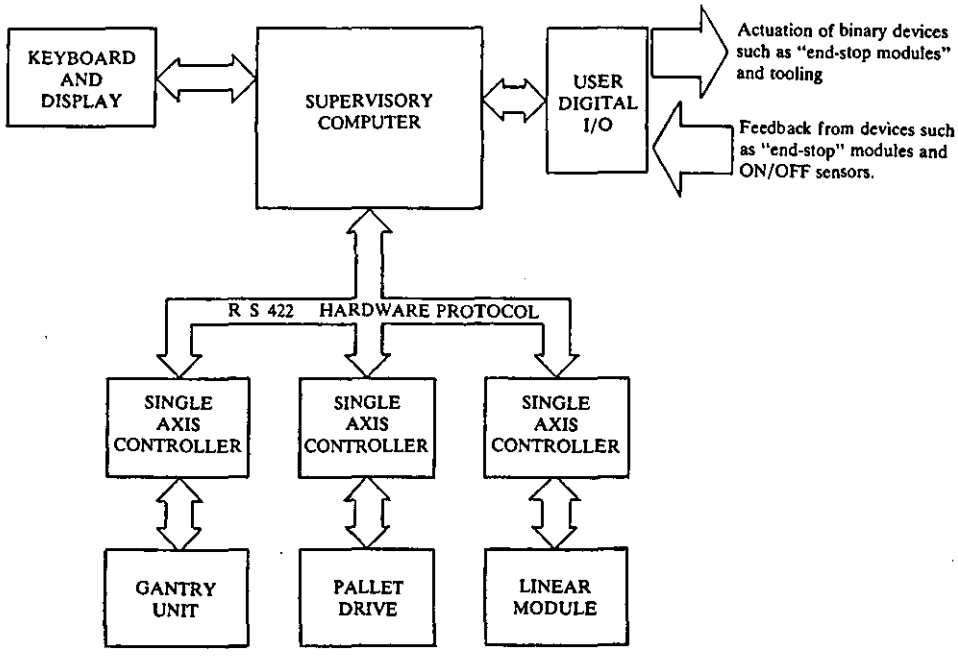


Fig. 3. Control architecture of the coddler assembly system.

(2) the lower, SAC level controls positioning of the servo axes to set-points defined by the supervisory computer. Information exchange between levels occurs through a standard hardware serial data link utilising software based communication protocols which were evolved for the commercially available SACs.[11]

Each servo controlled axis is based on standard end stop mechanical hardware. The addition of a position sensor and a proportional valve interfaced to a SAC allows closure of the position loop to be achieved, thereby forming a "programmable positioner" of the type which is now commercially available from Martonair (see Figure 1). The end stop axes are controlled by binary valves sequenced directly from the supervisor.

CONCLUSIONS

The existing family of Martonair/LUT pneumatic servo controlled modules provide low cost point to point positioning. They have adequate performance to satisfy a wide variety of tasks in systems ranging in configuration from single axes through axis groups to large distributed manipulators.

There is tremendous potential for the use of distributed manipulators in industry, i.e. special purpose flexible automated machines composed of modular elements. In the design and development of any manipulator system it is of great importance to minimise the systems engineering content since this usually accounts for a very large part of the total installed cost. It is therefore vitally important that the elements in a modular system are as easy to integrate and reconfigure as possible both during initial build and for post installation modifications and extensions.

In producing an easily reconfigurable modular system the design of the supervisory controller is of great importance since it must have the ability to produce and execute concurrent programs for each axis group and provide adequate teach facilities in systems of variable configuration.[12] The evolution of supervisory control hardware and software to satisfy these requirements is a major aim of the current research.

Faster cycle times and the ability to provide contour following would greatly extend the applications areas for modular systems. Research at LUT into electric servo drives aims to produce modular elements with these capabilities in the future.

References

1. R.H. Weston, P.R. Moore, T.W. Thatcher and J.D. Gascoigne, "Design of a Family of Modular Robots" *Procs. of 7th Int. Conf. ICPR* (Windsor, Canada 1983) pp. 235–267.
2. P.R. Moore, R.H. Weston, T.W. Thatcher and J.D. Gascoigne, "Modular Robot Systems" *Procs. of 2nd IASTED Int. Sym. on Robotics and Automation* (Lugano, Switzerland, June 22–24, 1983).
3. R.H. Weston, P.R. Moore, T.W. Thatcher and G. Morgan, "Computer-controlled Pneumatic Servo-drives" *Procs. of Inst. of Mech. Engs.* **198B**, No. 14, 275–281 (1984).
4. F. Riley, "The Use of Modular Flexible Assembly Systems as a Halfway Path between Special Design and Robots" *3rd Int. Conf. on Assembly Automation* (May, 1982).
5. Y.G. Kozyrev, "Constructing a Standard Series of Industrial Robots" *Machines Tooling* **49**, No. 7, 3–10 (1978).
6. B.N. Surnin *et al.*, "Design Features of Modular Type Robots" *Machines Tooling* **49**, No. 7, 17–20 (1978).
7. Martonair Modular Handling Units 6.2.11, Publication S/1/138, Issue 3, October 1984 (Martonair Ltd., Twickenham, England).
8. R.H. Weston and G. Morgan, "A New Family of Robot Modules and Their Industrial Application" *Procs. of IMechE. Conf. on UK Robotics Research* (4–5 December, 1984).
9. G. Morgan, "Programmable positioning of Pneumatic actuators" *J. Applied Pneumatics* **11**, No. 82 (May, 1985).
10. T.W. Thatcher, R.H. Weston, P.R. Moore and R. Harrison, "Supervisory Control of Single Axis Controllers for Modular Robotic Systems using a Serial Interface" National Conference on Production Research, Nottingham (September, 1985).
11. T.W. Thatcher, R.H. Weston, P.R. Moore and R. Harrison, "Single Axis Controllers for Modular Work-handling Systems" *Procs. of 4th IASTED Int. Symp. on Robotics and Automation* (Lugano, Switzerland, June, 1985).
12. T.W. Thatcher, R.H. Weston, P.R. Moore and J.D. Gascoigne, "Software Design for Modular Robots" *Int. J. Prod. Res.* **22**, No. 1, 71–84 (1984).

APPENDIX B.2

THE USE OF DISTRIBUTED PROGRAMMABLE ACTUATORS FOR FLEXIBLE ASSEMBLY

R. Harrison, J. Pu, R. H. Weston, P. R. Moore and A. H. Booth
Department of Manufacturing Engineering,
Loughborough University of Technology, UK.

ABSTRACT

This paper considers the use of distributed programmable actuators for assembly automation. These devices bridge the gap in industrial machine systems between dedicated "hard" automation and "general purpose" industrial robots. For example, they offer the opportunity for optimising the machine configuration including the choice of drive system elements for each individual operation. Distributed machine systems, which have been studied at Loughborough University for several years are used to illustrate the basic principles. The functionalities of "multi-purpose" motion and supervisory controllers designed for electric and pneumatic drive systems are described together with a discussion of the performance characteristics of these drive types.

INTRODUCTION

Industrial automation is becoming progressively more widespread with the need to reduce product costs through improved efficiency and better quality control. The automated machinery used in industry is extremely diverse both in form and function. The tasks performed can be simple or complex and may either semi or fully automate manufacturing processes. These diverse requirements have led to an equally wide range of manufacturing machines and associated control systems with very little standardisation between them. Typical industrial systems are difficult to maintain and modify and almost impossible to integrate together at any chosen level [1]. If CIM is to be effectively achieved individual machine elements can no longer be regarded and evaluated as discrete items of plant but must be seen as vital pieces in the "automation jigsaw".

This paper is promoting the concept of modular machine systems. The concept of modularity is in itself not new and it has been applied in disciplines such as mechanical, electronic and software engineering for many years. However, no consistent approach across these disciplines has been apparent in the design of contemporary machinery [2]. A methodology is needed to integrate machine elements in a structured manner. The use of a modular design framework for machine configuration and control is developed with reference to the existing methods it could replace and the potential improvements it can provide. A modular demonstrator system has been developed which uses industrially representative hardware in a printed circuit board assembly application.

CURRENT ASSEMBLY METHODS

By nature assembly processes are extremely diverse including for example insertion, glue laying, press fits and spot welding. There are also wide variations in product variability and volume, typically ranging from high volume products with low variation to small batch production of frequently modified items. A number of common elements can however be identified within most assembly systems. Generally they incorporate manipulators for component handling, feeder devices to support component flow and a control system to integrate its operation. The manipulators used in assembly systems can generally be classified in one of three categories [3]:

(a) dedicated special purpose machines where the manipulator is designed and built specifically for one task,
(b) modular manipulators built as standard units and configured together in different ways to suit a wide range of tasks, and
(c) general purpose manipulators of fixed configuration, i.e. conventional robots.

A particular assembly system will usually include a number of manipulators which can be the same or a mixture of manipulator types.

Hard automated machinery plays an essential role in the automated assembly of high volume products. Major companies specialise in the design and manufacture of such systems for example the Bodine Corporation (who produce cam operated long line assembly machines) and Bosch (who produce pneumatically actuated assembly machines).

Where more flexible automation is needed in assembly then software controlled machines are required, typically SCARA and Prismatic robots of fixed configuration or modular elements combined in a user defined manner.

Feeder devices come in numerous forms, they cover the functions of component storage, orientation and placement. These devices are usually dedicated to a single task.

Control systems vary widely depending on the degree of flexibility required in the assembly system. They vary from a simple sequencer for co-ordinating the elements of a small dedicated machine to a complex mix of programmable logic controllers, robot controllers and supervisory computers on larger more flexible systems.

GENERAL PURPOSE MANIPULATORS

The major advantage of general purpose robotic manipulators over task-specific hardware for automation is their flexibility. In theory, a robot's task can be changed simply by loading a new program into its controller. However in practice this is rarely the case. Robot arms can be purchased in a variety of mechanical forms and each manipulator will demonstrate specific kinematic and dynamic properties sufficient to provide mechanical flexibility for a range of manufacturing tasks. However, having provided mechanical flexibility, mechanical optimisation is seldom achieved when accomplishing a given task. The manipulators potential flexibility is also seldom properly utilised since once tooled up with conventional feeding, fixing and processing equipment, the robot becomes part of a relatively inflexible machine which cannot deal with significant process or product variations.

A fundamental problem exists, that of achieving a sufficiently high level of optimisation while maintaining acceptable flexibility in a cost effective manner. These problems have only been overcome when using conventional robot arms in a limited number of manufacturing application areas [4].

DISTRIBUTED MODULAR MANIPULATORS

The major factors that define a robots configuration are the link lengths, joint actuators and geometry of joint-link connections. Using manipulators with different configurations for each task is possible when the task requirements are known beforehand [5]. Distributed manipulators utilise their degrees of freedom (ie. their axes of motion) along the production line where they are actually needed and provide as much - but no more - complexity than the various production tasks demand.

Distributed robotics is now viewed by many engineers as the structural foundation of flexible manufacturing, whereas the idea of realising a machine to replace the man - i.e. the general purpose robot, has been largely abandoned; firstly due to cost and technical snags but also because the automated factory based on this anthropomorphous view is not the most efficient [6].

The Modular System Group at Loughborough have worked for some eight years on deriving modular distributed manipulators and investigating their application in flexible manufacturing [7,8]. The reconfigurable modular manipulator system currently under research extends the concept of modularity throughout the entire machine to include not only the mechanical hardware, but also the electrical hardware and most importantly the machine control system.

CURRENT PRACTICES IN MACHINE CONTROL

Currently two distinctly different approaches are commonly used when creating a computerised machine control system as depicted in Fig. 1.

The first of these approaches (see Fig. 1i) involves creating a custom designed control system to accomplish a specific task. An array of conventional control systems of this type exist including robot controllers and computer controllers for various semi-dedicated and dedicated machines [4,9]. These custom controllers typically offer a wealth of features but are of fixed configuration and their realisation ties them to specific mechanical hardware.

The second approach is embodied in various forms of "industrial controller" (including programmable logic controllers) which provide a degree of modularity and configurability [10]. Industrial controllers of this type (see Fig. 1ii) are available "off the shelf" and can be customised to control specific machines through selecting hardware modules and creating control software using either a symbolic (such as relay ladder diagrams) or high level programming language. Many industrial controllers have evolved as replacements for relay and hard wired electronic controllers with new

functions added over the years in a bottom-up manner. They typically exhibit a confused software structure and are difficult to reconfigure or integrate together.

The philosophy embodied in industrial controllers is an excellent one, recognising that there are many common problems when controlling manufacturing machines and providing modules and configuration tools for creating specific machine control systems. Unfortunately current industrial controllers can exhibit major limitations where high levels of functionality are required (e.g. multi-axis motion control) or where products or process changes are frequent. With advances in computer technology since their conception in the early seventies, the time is now ripe for problem dominated approaches to the engineering of real-time process and machine control systems. Study of current control practices reveals the following clear trends [11]:

(a) the current ad-hoc development practices used for the implementation of machine controllers should be superceded;
(b) the scope of development practices should be widened from focusing purely on implementation to encompass problem formulation and definition.

A NEW MODULAR APPROACH TO MACHINE CONTROL

The Modular Systems Group at Loughborough has devised a family of software based machine control modules which can be arranged in a hierarchical manner (see Fig. 3). The philosophy behind the approach is one of decomposing the control functions required for many types of manufacturing machines. This decomposition has led to the specification and creation of control system modules to form a software library of building elements. For a particular machine, a controller can be configured based on an appropriate selection and aggregation of the library modules (see Fig. 4). The architecture illustrated can thus be a useful step towards creating modular and hence more generally applicable control systems. A key factor in enabling this development is the application of a real time operating system , Microware OS9/68000, to provide a "virtual machine" at a level above the physical computer hardware [12]. This virtual machine supports a structured device independent real time environment for control system design [13].

Software tools have been implemented, using the programming language C, to support the creation and manipulation of system modules during controller development and at machine run time. System calls supported by OS9 are used to provide the required "bindings" between modules. To ensure the correct "fit" of the modules with each other (and with external devices) "templates" are used to enforce a standard data format. In its current implementation (see Fig. 3) the system consists of three levels:

(a) the machine module which contains the data relevent to the overall machine;
(b) the task modules (whose number and type are defined in the machine module) support the concurrent use of multiple "applications programs" in the system. These application programs may be concerned with mechanism control or serve a general purpose programming function;
(c) the component modules are concerned with machine specific I/O devices (e.g. axes of motion, analogue or digital inputs or outputs).

Fig. 2 illustrates this modular approach where essentially a simple two-level physical control hierarchy is employed. The use of separate Single Axis Controllers (SACs) associated with each axis of motion, with their interactions organised and managed by the supervisory controller leads to a rational decoupling of control functionality and enables axis controllers to be produced as "standard" components. This heirarchically organised segmentation of the control problem can be more generally applied if the SACs embody sufficient functionality to control a variety of drive types.

DRIVE SYSTEMS FOR ASSEMBLY AUTOMATION

There are often wide variations in the force and positioning requirements for each axis of motion in a given assembly machine. At Loughborough University we have been looking at the use of "mixed" drive technology on modular manipulator systems. The use of a

reference architecture concept, with seperate SACs for the major degrees of freedom, allows the easy mixing or substitution of different drive types. This gives the opportunity to optimise the cost/performance ratio for each axis of motion on a given machine. Machines are typically powered either hydraulically, electrically or pneumatically or by some combination of these methods.

Broadly speaking, hydraulic drives provide a durable, reliable and safe form of high force or power actuator which possesses high stiffness and is backlash free. However, hydraulic systems are often bulky, noisy, expensive and are liable to oil leakage. Hydraulic drives are normally employed in application areas with power requirements higher than 10 kw.

Conventionally pneumatic drives are widely employed for "bang-bang" control of repetitive operations such as pick and place movements in special purpose assembly machines and for other non-programmable motions for example simple robot end effectors. Pneumatic systems are comparatively cleaner and lighter than hydraulics, but are rather more difficult to apply with closed loop control because of their tendency to oscillate under high-inertia load and because of the compressibility of the gas medium. The severe inherent non-linearities introduced by the fluidic control medium can now be effectively suppressed by advanced digital control techniques [14,15]. Recently pneumatic servo drives have shown cost advantages in low to medium power application areas, but tuning problems still remain with present pneumatic control systems.

Electric motors exhibit the best linearity and simplicity in system dynamics and are well understood theoretically. They hence offer currently the most mature and usable drive technology and are generally considered to be the most suitable for programmable positioning and velocity control in assembly applications.

The majority of assembly operations require varying degrees of either motion or force control. Point-to-point positioning is a typical requirement for many assembly operations e.g. palletising, parts fitting, spot welding. All three drive technologies can now offer usable point to point positioning in appropriate application areas.

Good velocity control can be relatively easily achieved with both electric and hydraulic systems and has been well documented [16]. Velocity control is more difficult to achieve with pneumatic drives although pneumatics have a long standing history of being used to accomplish simple open loop speed control tasks [17].

The authors would like to emphasise that servoed pneumatic drives can exhibit some unique performance properties (not possessed by electric or hydraulic drives) which may be usefully employed in assembly applications. Due to the existence of choked flow in the gas medium, the velocity response of pneumatic drives can be insensitive to load and/or supply pressure variations. The compressibility of air does not always create control difficulties, it provides compliance which can be highly desirable in many force or hybrid control tasks which are common place in assembly.

DISTRIBUTED MANIPULATOR DEMONSTRATOR

A demonstrator system has been commissioned for materials handling applications in Printed Circuit Board (PCB) manufacture and assembly. The facility which uses industrialised hardware wherever possible, has been configured to provide a realistic environment for practical control system evaluation. The system, shown in Fig. 5, consists of modular actuators and tooling to transport PCBs between pallets and work piece fixtures. The demonstrator extends the concept of modularity throughout the machine mechanical hardware, electrical, electronic and control systems [2]. The mechanical modules are mounted on a modular extruded section aluminium framework which is also reconfigurable. The system implementation consists of up to twelve programmable axes of motion plus additional I/O controllers. Based on functional analysis of the application these elements are organised and controlled as five distinct groups (see Fig. 6).

Group 1 Component Insertion: Four electric servo modules with interchangable tooling.

Group 2 PCB Registration: Electric servo actuated slideway and two axis stepper motor positioned PCB registration fixture.
Group 3 PCB Handling: Pneumatic servo actuated gantry module with electric servo actuated vertical module and programmable pcb gripper.
Group 4 Pallet Transportation: Pneumatic servo actuated rodless cylinder with pallet/carrier location tooling.
Group 5 Pick and Place: Two electric servo actuated modules and tooling for odd form components.

This rig makes use of "mixed" drive technology. Pneumatic actuators are used for simple binary devices such as end effectors and clamps. Servo pneumatic drives are used for the lower accuracy point to point PCB handling (for position accuracy of up to 0.1 mm). Stepper motors are used for programmable fixtures where open loop control is acceptable. DC servo actuators are employed where velocity control is required and for component insertion where accuracies better than 0.1mm are needed.

CONCLUSIONS

This paper highlights some of the limitations of conventional manipulators and control systems which are currently used for assembly automation. The use of distributed manipulators in conjunction with modular control systems offer a potentially more efficient solution and could significantly extend the manufacturing areas in which automation can be justified.

The modular systems group firmly believe that the concepts described in this paper if not the specific implementation of those concepts evolved at Loughborough University will ultimately revolutionise machine control methods.

ACKNOWLEDGEMENTS

This work has resulted from the availablity of funding from the ACME Directorate of SERC, UK.

REFERENCES

[1] Ruckman, R.P. "When I Can Talk to the Outside World, What Will I Say?", Proc. 6th Annual Control Eng. Conf., Reed Publishing USA, (May 1987).
[2] Jackson, M.R., Moore, P.R., Weston, R.H., Harrison, R. and Booth, A.H. "Advanced Concepts in Machine Design through the adoption of Mechatronics Technology in Modular Machines", Mechatronics in Products and Manufacturing, Conf. Lancaster University, England (September 1989).
[3] Weston, R.H., Harrison, R., Moore, P.R. and Booth A.H. "The State of the Art In Flexible Assembly", SERC, ACME Grantees' Study Tour Report, (1986).
[4] Redford, A.H. and Lo, E. Robotics in Assembly, Open Univ. Press, England (1986).
[5] Schmitz, D. "The CMU Reconfigurable Modular Manipulator System", Technical Report CMU-RI-TR-88-7, Carnegie Mellon University, Pittsburgh, USA, (May 1988).
[6] Challis, H. "Where is Robotics Going?", Automation, UK, pp. 23-29 (May 1989).
[7] Weston, R.H., Harrison, R., Booth, A.H. and Moore, P.R. "Universal Machine Control System Primitives for Modular Distributed Manipulator Systems", Int. J. Prod. Res., Vol. 27, No. 3, pp. 395-410 (1989).
[8] Harrison, R., Weston R.H. and Moore P.R. "A Study of Application Areas for Modular Robots", Robotica, Vol. 5, pp. 217-221 (1987).
[9] Boothroyd, G., Poli, C. and Muich, L.E. Automatic Assembly, Marcel Dekker, Monticello, NY, USA (1982).
[10] Machiewicz, R.W. "Programmable Controllers". Industrial Automation, Chapman and Hall, London, UK (1986).
[11] Jasany, L.C. "PLCs into the 1990s", Automation, USA pp 20-24 (April 1989).
[12] "The OS-9 Catalog", Ed. Davis D.F., Microware Systems Corporation, Des Moines, Iowa, USA (January 1989).
[13] Weston R.H., Harrison, R., Booth, A.H. and Moore, P.R. "A New Concept in Machine Control", Comp.-Int. Manf. Systems, Vol. 2, No. 2, pp. 115-122 (May 1989).

[14] Pu, J. Weston, R.H. and Liu, Y.M. "Regional Motion Control Models for Single-Axis Pneumatic Robots", Robotics: Applied Mathematics and Computational Aspects, IMAA Conf. Loughborough University, England (July 1989).
[15] Saffe, P. "Nonlinear Control Concepts for Servo Hydraulic Drives", Fluid Power 8, Elsevier Applied Science Publishers Ltd. pp. 478-479 (1988).
[16] Critchlow, A.J. Introduction to Robotics, MacMillan, New York, (1985).
[17] Yeaple, F. Fluid Power Design Handbook, Cahners, Boston, Mass. USA (1984).

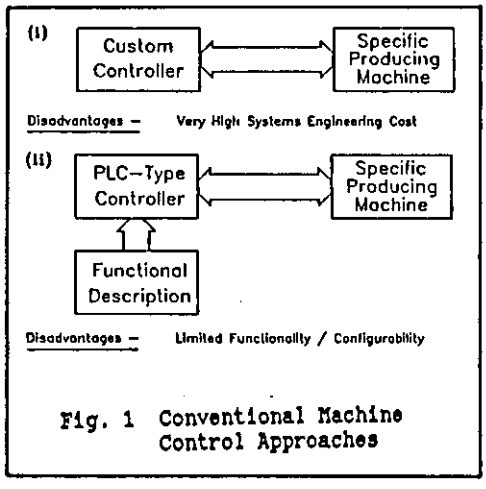


Fig. 1 Conventional Machine Control Approaches

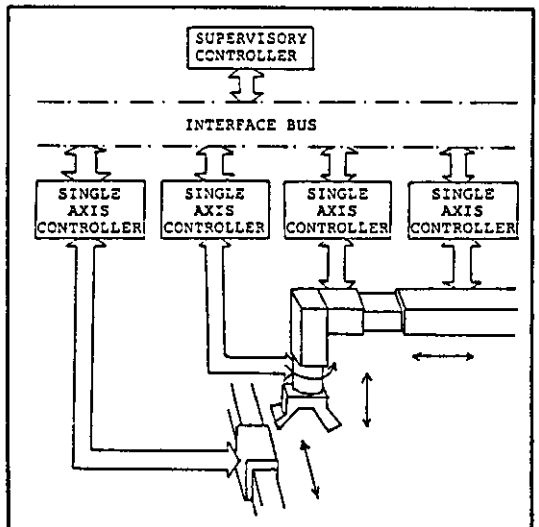


Fig. 2 Two-Level Machine Control Hardware Hierarchy

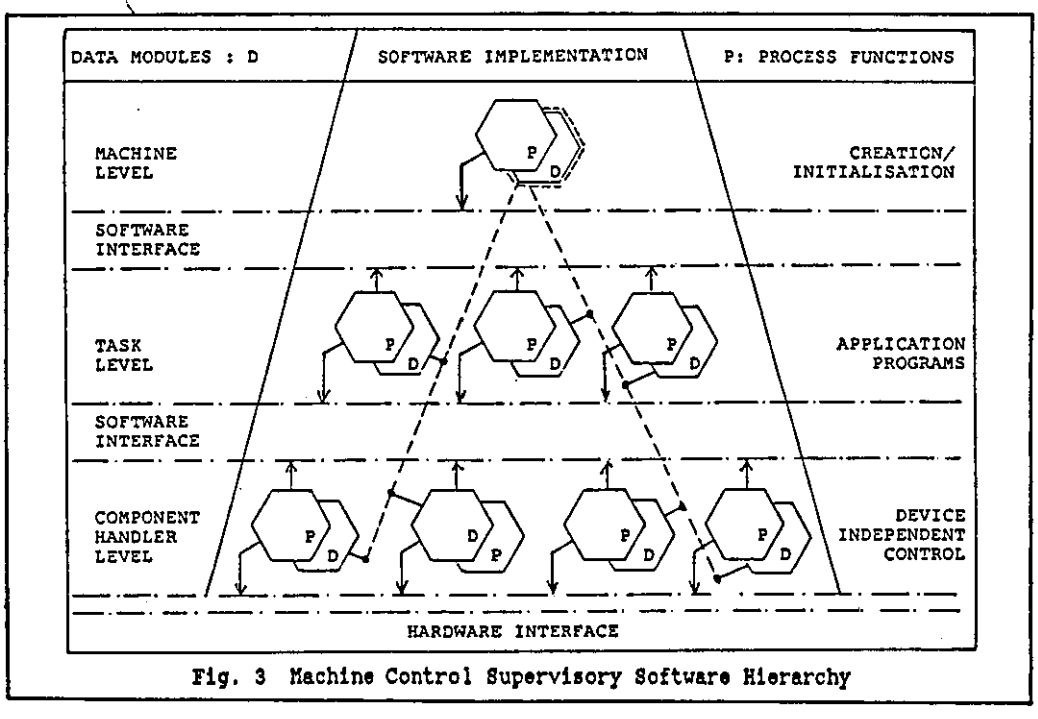


Fig. 3 Machine Control Supervisory Software Hierarchy

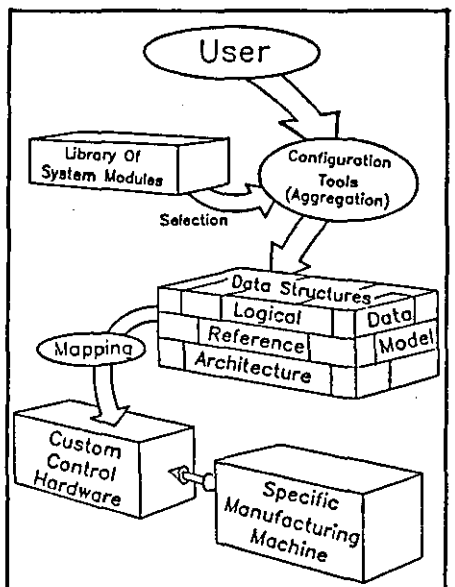


Fig. 4 Reference Architecture Concept

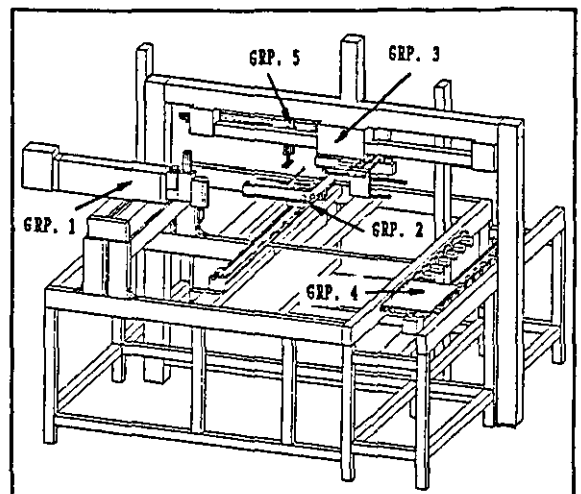


Fig. 5 Demonstrator Mechanical Modules

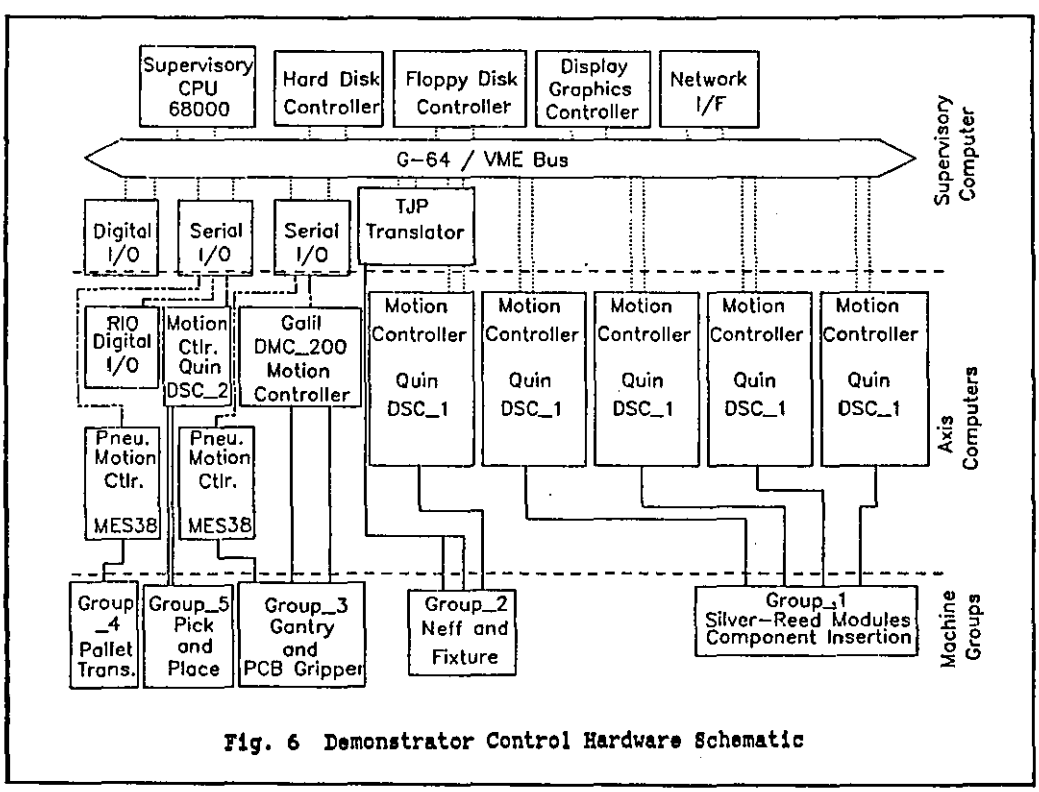


Fig. 6 Demonstrator Control Hardware Schematic

Appendix C
Microware Basic Programs

Contents

APPENDIX C.1

Example Task Program: insert

APPENDIX C.2

Example Basic Link Subroutine: link_port

APPENDIX C.1 EXAMPLE TASK PROGRAM: INSERT

```
PROCEDURE insert
 0000       REM
 0006       REM UMC DATA TYPE DECLARATIONS.   DO NOT EDIT.
 0036       REM ------------------------------------------------
 0070       BASE 0
 0074       TYPE AXIS=ax_name:STRING[32]; ax_modptr,ax_no:INTEGER
 00A0       TYPE PORT=pt_name:STRING[32]; pt_modptr,pt_no:INTEGER
 00CC       TYPE EVENT=ev_name:STRING[32];
                                  ev_modptr,ev_no,ev_value:INTEGER
 00FE       TYPE LOCATION=loc_name:STRING[32];
                                      loc_modptr,loc_no:INTEGER
 012A       REM ------------------------------------------------
 0164       REM
 016A       REM
 0170       REM
 0176       REM UMC VARIABLE DECLERATIONS.
 0196       REM ------------------------------------------------
 01D2       DIM axes:INTEGER
 01DE       DIM axis_x,axis_w:AXIS
 01F2       DIM port1:PORT
 0200       DIM event1,event2,finish2,group1:EVENT
 0220       DIM pickup_U,pickup_D:LOCATION
 0234       DIM above_location(20):LOCATION
 024C       DIM insert_location(20):LOCATION
 0264       REM ------------------------------------------------
 02A0       REM
 02A6       REM
 02AC       REM USER VARIABLE DECLERATIONS.
 02CC       REM ------------------------------------------------
 0308       DIM MIRROR,ALTERNATE,SYNC,EXIT:INTEGER
 0326       DIM SLOW,MEDIUM,FAST:INTEGER
 033E       DIM HIGH,LOW:INTEGER
 0350       DIM count:INTEGER
 035C       LET MIRROR=0
 036C       LET ALTERNATE=1
 037C       LET SYNC=2
 038C       LET EXIT=3
 039C       LET SLOW=5
 03AC       LET MEDIUM=40
 03BC       LET FAST=80
 03CC       LET HALT=0
 03DE       LET HIGH=1
 03EE       LET LOW=0
 03FE       REM
 0404       REM ------------------------------------------------
 0440       PAUSE "insert: setup next - (TRON if required)"
 046E       REM
 0474       REM
 047A       REM UMC COMPONENT SETUP AND LINK STATEMENTS.
 04A8       REM ------------------------------------------------
 04E4       REM
 04EA       PRINT "insert: setup and link"
 0508       RUN setup(axes)
 0518       REM
 051E       LET event1.ev_name="event1"
 0538       RUN link_event(event1)
```

```
0548      REM
054E      LET event2.ev_name="event2"
0568      RUN link_event(event2)
0578      REM
057E      LET axis_x.ax_name="axis1"
0596      RUN link_axis(axis_x)
05A6      REM
05AC      LET port1.pt_name="port1"
05C4      RUN link_port(port1)
05D4      REM
05DA      LET pickup_U.loc_name="pickup_U"
05F6      RUN link_location(pickup_U)
0606      REM
060C      LET pickup_D.loc_name="pickup_D"
0628      RUN link_location(pickup_D)
0638      REM
063E      LET insert_location(0).loc_name="place_D0"
0660      RUN link_location(insert_location(0))
0676      REM
067C      LET insert_location(1).loc_name="place_D1"
069E      RUN link_location(insert_location(1))
06B4      REM
06BA      LET insert_location(2).loc_name="place_D2"
06DC      RUN link_location(insert_location(2))
06F2      REM
06F8      LET insert_location(3).loc_name="place_D3"
071A      RUN link_location(insert_location(3))
0730      REM
0736      LET insert_location(4).loc_name="place_D4"
0758      RUN link_location(insert_location(4))
076E      REM
0774      LET insert_location(5).loc_name="place_D5"
0796      RUN link_location(insert_location(5))
07AC      REM
07B2      LET insert_location(6).loc_name="place_D6"
07D4      RUN link_location(insert_location(6))
07EA      REM
07F0      LET insert_location(7).loc_name="place_D7"
0812      RUN link_location(insert_location(7))
0828      REM
082E      LET insert_location(8).loc_name="place_D8"
0850      RUN link_location(insert_location(8))
0866      REM
086C      LET insert_location(9).loc_name="place_D9"
088E      RUN link_location(insert_location(9))
08A4      REM
08AA      REM
08B0      LET above_location(0).loc_name="place_U0"
08D2      RUN link_location(above_location(0))
08E8      REM
08EE      LET above_location(1).loc_name="place_U1"
0910      RUN link_location(above_location(1))
0926      REM
092C      LET above_location(2).loc_name="place_U2"
094E      RUN link_location(above_location(2))
0964      REM
096A      LET above_location(3).loc_name="place_U3"
098C      RUN link_location(above_location(3))
```

```
09A2      REM
09A8      LET above_location(4).loc_name="place_U4"
09CA      RUN link_location(above_location(4))
09E0      REM
09E6      LET above_location(5).loc_name="place_U5"
0A08      RUN link_location(above_location(5))
0A1E      REM
0A24      LET above_location(6).loc_name="place_U6"
0A46      RUN link_location(above_location(6))
0A5C      REM
0A62      LET above_location(7).loc_name="place_U7"
0A84      RUN link_location(above_location(7))
0A9A      REM
0AA0      LET above_location(8).loc_name="place_U8"
0AC2      RUN link_location(above_location(8))
0AD8      REM
0ADE      LET above_location(9).loc_name="place_U9"
0B00      RUN link_location(above_location(9))
0B16      REM
0B1C      REM
0B22      REM ------------------------------------------------
0B60      REM
0B66      REM
0B6C      REM START OF MAIN PROGRAM CODE.
0B8C      REM ------------------------------------------------
0BCA      PRINT "axis.ax_no = ",axis_x.ax_no
0BE8      PRINT "event.ev_no = ",event1.ev_no
0C08      PRINT "port.pt_no = ",port1.pt_no
0C26      FOR count=1 TO 10
0C46        RUN smove(location1,FAST)
0C5E        PAUSE
0C62        RUN smove(location2,FAST)
0C7A      NEXT count
0C8C      REM
0C92      PAUSE
0C96      RUN look(event1)
0CA6      PRINT "value of event1 = ",event1.ev_value
0CCA      PAUSE
0CCE      RUN signal(event1)
0CDE      PAUSE
0CE2      RUN wait(event1)
0CF2      PAUSE
0CF6      RUN output(port1,1,HIGH)
0D16      PAUSE
0D1A      RUN output(port1,1,LOW)
0D3A      PAUSE "pick and place loop next"
0D5A      FOR n=0 TO 9
0D7E        PAUSE "in pick and place loop"
0D9C        RUN smove(above_pickup,FAST)
0DB4        RUN smove(pickup,FAST)
0DCC        RUN smove(above_location(n),FAST)
0DEA        RUN smove(insert_location(n),FAST)
0E08        RUN smove(above_location(n),FAST)
0E26      NEXT n
0E48      RUN setdown(axes)
0E58      END
```

APPENDIX C.2 EXAMPLE BASIC LINK SUBROUTINE: LINK_PORT

```
PROCEDURE link_port
 0000       TYPE PORT=p_name:STRING[32]; p_modptr,p_no:INTEGER
 002C       PARAM p:PORT
 003A       DIM cp_name:STRING[32]
 0050       DIM cp_type,task_ptr,cp_no,cp_ival:INTEGER
 006E       DIM cp_fval:REAL
 007A       REM initialise params
 0090       cp_name=p.p_name
 00A0       cp_type=1
 00AE       task_ptr=0
 00BC       cp_no=0
 00CA       cp_ival=0
 00D8       cp_fval=0.
 00EA       RUN
LINK_CP(cp_type,cp_name,task_ptr,cp_no,cp_ival,cp_fval)
 0122       p.p_modptr=task_ptr
 0134       p.p_no=cp_no
 0146       END
```

Appendix D
Headers for Machine Module Access

Contents

Module Template Header:

```
/* OFFSET TABLE: temmc.h */
 /* 32 char string: generic machine name */
#define  MC_MODTYPE  6
 /* integer: number of axes */
#define  MC_AXES  40
 /* integer: number of i/o ports */
#define  MC_PORTS  46
 /* integer: number of events */
#define  MC_EVENTS  52
 /* integer: number of tasks */
#define  MC_TASKS  58
 /* node: machine axes: */
 /* node: machine ports: */
 /* node: machine events: */
 /* node: machine tasks: */

#define MODSIZE_temmc  62  /* module size required */

#define SUBSIZE_temmc  58  /* sub structure size */
```

Axis Sub-Structure Header:

```
/* OFFSET TABLE: submcax.h */
 /* node: axis parameter: */
     /* 32 char string: axis name */
#    define  AX_NAME  6
     /* 32 char string: axis edit file name */
#    define  AX_EDFILE  40
     /* integer: set axis task link bits */
#    define  AX_TLINK  74

#define MODSIZE_submcax  78  /* module size required */

#define SUBSIZE_submcax  74  /* sub structure size */
```

Port Sub-Structure Header:

```
/* OFFSET TABLE: submcpt.h */
 /* node: i/o port parameter: */
     /* 32 char string: port name */
#    define  PT_NAME  6
     /* 32 char string: port edit file name */
#    define  PT_EDFILE  40
     /* integer: set port task link bits */
#    define  PT_TLINK  74

#define MODSIZE_submcpt  78  /* module size required */

#define SUBSIZE_submcpt  74  /* sub structure size */
```

Event Sub-Structure Header:

```
/* OFFSET TABLE: submcev.h */
 /* node: event parameter: */
     /* 32 char string: event name */
#    define  EV_NAME  6
     /* integer: set event task link bits */
#    define  EV_TLINK  40

#define MODSIZE_submcev  44  /* module size required */

#define SUBSIZE_submcev  40  /* sub structure size */
```

Task Sub-Structure Header:

```
/* OFFSET TABLE: submctsk.h */
 /* node: task parameter: */
     /* 32 char string: task name */
#    define  TSK_NAME  6
     /* 32 char string: process file name */
#    define  TSK_PROCESS  40
     /* 32 char string: program file name */
#    define  TSK_PROGFILE  74
     /* 32 char string: std. input path */
#    define  TSK_IN  108
     /* 32 char string: std. output path */
#    define  TSK_OUT  142
     /* 32 char string: std. error path */
#    define  TSK_ERROR  176
     /* 32 char string: axis group locations file */
#    define  TSK_POSFILE  210
     /* integer: set task link bits */
#    define  TSK_LBIT  244
     /* integer: task process id */
#    define  TSK_PID  250

#define MODSIZE_submctsk  254  /* module size required */

#define SUBSIZE_submctsk  250  /* sub structure size */
```

Appendix E
C Subroutine Support Programs

Contents

APPENDIX E.1

C Subroutine Root Psect Program: umcstart.a

APPENDIX E.2

C Library: slib_s.c

APPENDIX E.3

Example C Subroutine: wait_s.c

APPENDIX E.1 C SUBROUTINE ROOT PSECT PROGRAM: UMCSTART.A

```
************************************
*
* umcstart.a -  Startup routine for a C subroutine program.
*               Subroutine module to be called from Basic.
*
* Contents -    Dummy stack check.
*               Declare errno.
*               Changing static storage.
*               Returning static storage.
*               Reporting error codes back to basic.
*
* Note -        Currently 28 bytes of uninitialised static storage
*               are available in UMC task modules for general use.
*               It should NOT be assumed that these initially
*               contain zeros.
*
*
* RH Apr'88
*

    use <oskdefs.d>


    psect bstart_a,(Sbrtn<<8)!Objct,(ReEnt<<8)!0,0,0,bstart

    vsect
errno: ds.l 1 global error holder
    ends

bstart: bra main

*
* dummy stack checking
*
_stkcheck:
_stkchec:
rts

*
* _pro(<cmemory_ptr>) cmemory_ptr is in d0, changes static storage
*
_pro: move.l a6,d1  save b_mem_add in d1
 move.l d0,a6       put new address in a6
 move.l d1,d0       transfer b_mem_add to d0 for return
 rts

*
* _epi(<bmemory_address>) bmemory_address is in d0, restores old
storage
*
_epi: move.l d0,a6  restore old value of a6
 rts

*
```

```
* _error_epi(<bmemory_address>), returns error + restores old
storage
*
_error_epi: move.l errno(a6),d1 errno returned as error value
 move.l d0,a6                   restore old value of a6
 ori    #Carry,ccr              set carry to indicate error to basic
 rts

*
* _error(init_errno), returns error + NO change to static storage
*
_error: move.l d0,d1    errno returned as error value
 ori    #Carry,ccr              set carry to indicate error to basic
 rts

 ends
```

APPENDIX E.2 C LIBRARY: SLIB_S.C

```
/*

C Functions to support the subroutine modules written in C
and called from microware basic only.

These functions are for:
    Converting integers to ascii characters.
    Writing simple error path messages.

RH Jun'88

*/

/* _itoa() does int to ascii conversion with no static storage.
calls lmod() */
char *lmod(str,arg,modn)
char *str;
int arg;
int modn;
{
    int temp2;
    int temp1;
    temp1 = arg % modn;
    temp2 = arg / modn;
    if(temp2)
        str=lmod(str,temp2,modn);
    *str++ = (temp1 > 9) ? (temp1 + 'a' - 10) : (temp1 + '0');
    return (str);
}

char *_itoa(str,arg)
char *str;
long arg;
{
    char *rtn=str;
    if(arg < 0) {
        arg = -arg;
        *str++ = '-';
        if(arg <0) {
            strcpy(str,"2147483648");
            return(rtn);
        }
    }
    str = lmod(str,arg,10);
    *str ='\0';
    return(rtn);
}

/* sends a message to standard error path */
int err_message(mess_str_ptr)
char *mess_str_ptr;
{
    writeln(2,mess_str_ptr,strlen(mess_str_ptr));
}
```

```
/* sends a message, an integer and a carriage return to standard
error path */
int err_val(mess_str_ptr,arg)
char *mess_str_ptr;
int arg;
{
    char string_buf[12];
    writeln(2,mess_str_ptr,strlen(mess_str_ptr));
    writeln(2,strcat(_itoa(string_buf,arg),"\n"),13);/* upto 10
digits + sign */
}
```

APPENDIX E.3 EXAMPLE C SUBROUTINE: WAIT_S.C

```
/*
wait.c wait/decrement subroutine for basic

Basic calls will be of the form:-
TYPE SYS=static_store,event_id,port_ch:INTEGER; ioe_type: BYTE
TYPE IOE=value:INTEGER; system:SYS
DIM event1, event2:IOE
RUN link_event("<machine_name>","<event name>",<event_struc>) ---- 3
params.
RUN look(<event_struc>) ---- 1 param.
RUN set,pulse,signal(<event_struc>,{<flag>}) flag = ALL or FIRST - 1
or 2 parm.
RUN wait(<event_struc>, {<min>,<max>}) --- 1 or 3 params.

RH April 89 min,max now both 1.
*/

# include <errno.h>
# include <strings.h>

# include "/h0/umc/edfiles/temtsk.h"
# include "/h0/umc/edfiles/subtskev.h"

# define FALSE 0
# define TRUE  1
# define SIZE1 44
# define SIZE2 4    /* int minimim wait value */
# define SIZE3 4    /* int maximum wait value */
# define MIN_COUNT  1   /* in this case only counts of 1 or 3
allowed */
# define MAX_COUNT  3


extern char *_itoa();

struct event        /* template */
{
    char ev_name[32];   /* 32 */
    int ev_modptr;      /*  4 */
    int ev_no;          /*  4 */
    int ev_value;       /*  4 */
};                      /* 44   total size */


main(count, e_struc1, size1, e_min2, size2, e_max3, size3)
int count;
struct event *e_struc1;
long size1;
int *e_min2;
long size2;
int *e_max3;
long size3;
{
    int init_errno;         /* used until static storage is changed
*/
    int error, store, point;
```

```
    int ev_min, ev_max;     /* range of vals. for which increment is
applied */
    int bmem_add;           /* for A6, basic static storage address
*/
    int tskev_id;           /* event id */
    int offset;             /* id offset */

    error = FALSE;          /* initialise */
    store = FALSE;          /* static storage flag */
    ev_min = 1;             /* default wait continue range values */
    ev_max = 1;

    /* always check number of parameters passed first */
    if(error == FALSE)
    {
#ifdef DEBUG
        err_message("wait: Debug - Parameter data received:\n");
        err_val(" parameter count = ",count);
        err_val(" size of 1st parameter = ",size1);
        err_val(" size of 2nd parameter = ",size2);
        err_val(" size of 3rd parameter = ",size3);
#endif
        if(count != MIN_COUNT && count != MAX_COUNT)    /* not 1 and
not 3 */
        {
            error = TRUE;
            init_errno = 56;        /* return parameter error */
            err_message("wait: Error - Parameter count.\n");
        }
    }

    /* check ALL parameters for size before changing static storage
*/
    if(error == FALSE)
    {
        if (size1 != SIZE1)
        {
            error = TRUE;
            init_errno = 56;        /* return parameter error */
            err_message("wait: Error - First parameter size.\n");

        }
    }
    if(error == FALSE && count == MAX_COUNT)
    {
        if ( size2 != SIZE2)
        {
            error = TRUE;
            init_errno = 56;        /* return parameter error */
            err_message("wait: Error - Second parameter size.\n");
        }
        else
        {
            ev_min = *e_min2;
        }

        if ( size3 != SIZE3)
        {
```

```
            error = TRUE;
            init_errno = 56;        /* return parameter error */
            err_message("wait: Error - Third parameter size.\n");
        }
        else
        {
            ev_max = *e_max3;
        }
    }

    /* change static storage base address USE WITH CARE */
    if(error == FALSE)
    {
        bmem_add =_pro(e_struc1->ev_modptr + TSK_STATIC);
        store = TRUE;               /* store old A6 contents in
bmem_add */
#ifdef DEBUG
        err_val("wait: Debug - New static storage address = ",
        (e_struc1->ev_modptr + TSK_STATIC));
#endif
    }
    /* check event type - wait valid for inputs,outputs and
internals.
        type internal if port no = 0. */
    /* check if min and max present or assume 1 and 1 , see above */
    /* for a given input or output only certain values are sensible
        eg an. or dig. */
    /* obtain event id from task module using */
    /* wait for event with given id */
    /* if error true then return error */

    if(error == FALSE)
    {
    if ((offset = tsk_event_os(e_struc1->ev_modptr, e_struc1->ev_no,
EV_ID))
     == -1)
    {
        err_message("wait: Error - Reading task event id
offset.\n");
        error = TRUE;
    }
    }

    if(error == FALSE)
    {
    if ((tskev_id = rdmodi(e_struc1->ev_modptr, offset)) <= -1)
    {
        err_message("wait: Error - Reading task event id.\n");
        error = TRUE;
    }
    }

    /* wait for given event to be in range and return its value */
    if(error == FALSE)
    {
    if ((e_struc1->ev_value = _ev_wait(tskev_id, ev_min, ev_max)) ==
-1)
    {
```

```
#ifdef DEBUG
        err_message("wait: Debug - Event value of -1 returned.\n");
        err_val(" ev_min = ",ev_min);
        err_val(" ev_max = ",ev_max);
#endif
        if (((errno >= 167)&&(errno <= 170)) || (ev_min > -1) ||
(ev_max < -1))
        {                       /* in case event value is -1 */
            error = TRUE;    /* make error check as good as possible
*/
            err_message("wait: Error - Failed to wait for
event.\n");
        }
    }
    }

    if (error == FALSE)
    {
#ifdef DEBUG
        err_message("wait: Debug - Error flag = FALSE\n");
#endif
        _epi(bmem_add);
    }
    else
    {
        if (store == FALSE)
        {
#ifdef DEBUG
            err_message("wait: Debug - Error static storage not
changed.\n");
#endif
            _error(init_errno);
        }
        else
        {       /* normal error reporting route if parameters passed
are ok */
#ifdef DEBUG
            err_message
            ("wait: Debug - Error value in event static
storage.\n");
#endif
            _error_epi(bmem_add);
        }
    }
}
```