Design of a Generic Manufacturing Cell Control System.

Derya Pamukcu
Louisiana State University and Agricultural & Mechanical College

Design of a generic manufacturing cell control system

Pamukcu, Derya, Ph.D.

The Louisiana State University and Agricultural and Mechanical Col., 1987

DESIGN OF A GENERIC MANUFACTURING CELL CONTROL SYSTEM

A Dissertation

Submitted to the Graduate Faculty of
Louisiana State University and
Agricultural and Mechanical College
in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

in

Interdepartmental Programs In Engineering

by
Derya Pamukcu
B.S., Bogazici University, Istanbul, Turkey, 1977
M.S., Louisiana State University, 1982
August, 1987

ACKNOWLEDGEMENTS

The author wishes to extend his sincere gratitude to his major professor Prof. William E. Biles. His consistent encouragement, support and guidance were invaluable. The author also wishes to express his appreciation to his professors, colleagues and the staff of the Industrial Engineering Department of Louisiana State University for their constant support and encouragement.

Special thanks to my wife Sibel for her patience! Her undiminishing faith in me and her constant encouragement has made the completion of this project possible.

The author would finally like to acknowledge his parents who gave him the first lead in life by generously providing the love and care he appreciates so much.

TABLE OF CONTENTS

Page

ACKNOWLEDGEMENTS ii

LIST OF TABLES vi

LIST OF FIGURES vii

ABSTRACT ix

Chapter

1. INTRODUCTION 1

OBJECTIVE 4

DISSERTATION OUTLINE 5

2. BACKGROUND 6

COMPUTER INTEGRATED MANUFACTURING 6

HIERARCHICAL CONTROL 14

PREVIOUS WORK 25

3. SIMULATION OF CIM ENVIRONMENT 45

GENERAL 45

GATEWAY SIMULATOR PROGRAM 47

Data for Simulation 48

Organization of the Program 49

Operator Interface 52

Simulation of Workstation Responses 53

SUMMARY 54

4. MANUFACTURING CELL CONTROLLER 55

FUNCTIONS OF A MANUFACTURING CELL CONTROLLER 56

IMPLEMENTATION OF THE MANUFACTURING CELL CONTROLLER 58

CELL CONTROL PROGRAM 60

Databases Maintained by the CCP 61

Task Decomposition 63

Monitoring And Control 64

Operation of the Cell Control Program 65

Messaging System 70

SUMMARY 77

5. MACHINING WORKSTATION 78

WORKSTATION CONTROLLER 79

Implementation 80

Operation 84

Programming 86

PROGRAMMABLE EQUIPMENT 90

Manipulator Arm 91

Operation 93

Programming 104

Programmable Logic Controller 110

Programming 112

SUMMARY 114

6. MATERIAL HANDLING SYSTEM SIMULATION 115

IMPLEMENTATION 118

OPERATION 120

7. DISCUSSION AND CONCLUSIONS 122

FURTHER RESEARCH 126

REFERENCES 130

APPENDICES

A. ROBOT-5 INSTRUCTIONS 133

B. ABC PROTOCOL AND THE COMMUNICATIONS INTERFACE MODULE OF THE PLC 150

C. MAJOR FLOWCHARTS 153

VITA 164

LIST OF TABLES

Table		Page
4.1.	Databases Required for FMS Control	62
4.2.	Status Codes for Workstations	62
4.3.	Messaging Protocol for Operational Messages	73
4.4.	Messaging Protocol for Variable Length Messages	73
4.5.	Operational Messages	74
4.6.	Database Update Messages	74
4.7.	Message Format for Fixed Messages	75
4.8.	Messaging Format for Database Update Messages	75
4.9.	Opcodes and Operands of Operational Messages	75
4.10.	Database Message Information	76
5.1.	Opcodes for Workstation Programming Instructions	89
5.2.	Workstation Program Instruction Structures	89
5.3.	Message Format	109
5.4.	The Messaging Protocol	109

LIST OF FIGURES

Figure		Page
2.1.	Computer Integrated Factory	8
2.2.	Manufacturing Database	8
2.3.	Order Release to Shop Floor	10
2.4.	Automated Material Handling and Warehouse	10
2.5.	Material Handling System Coupling	12
2.6.	Sharing Information Through Factory-Wide Network	13
2.7.	A Simple Hierarchy	17
2.8.	An Interconnected Dynamical System	17
2.9.	A Simple Problem Decomposition	20
2.10.	A Simple Control Unit	20
2.11.	Feedback to Hierarchical Control System ...	22
2.12.	Correction of Performance	24
2.13.	AMRF Control Levels	26
2.14.	AMRF Control System Hierarchy	26
2.15.	Generic Control Module	30
2.16.	Evolution of An Intelligent Automated Control System	30
2.17.	Current AMRF Control Structure	31
2.18.	Hierarchical Control Structure by Gomoa ...	39
2.19.	Generalized Hierarchical Structure of Expert Systems	40
2.20.	Expert Control System Block Diagram	40

2.21. Organization of NNS 42

3.1. Gateway Simulator Organization 51

3.2. Expanded Simulation Model 51

3.3. Operation Menu of Gateway Program 52

4.1. Cell Controller Organization 59

4.2. Task Decomposition 63

4.3. Cell Control Program Display of Status 69

4.4. Fully Interlocked Protocol 70

5.1. Machining Workstation 83

5.2. Multiple Workstations with Dissimilar Configurations 83

5.3. Binding of Machines in Workstation 84

5.4. Manipulator Arm Interfacing to the Microcomputer 92

5.5. Remote Operation of the Manipulator Arm ... 92

5.6. Manipulator Arm Control Program Main Menu .. 95

5.7. Manual Control of Microbot 95

5.8. Edit/Modify Program Menu 99

5.9. Listing of a Program 99

5.10. Edit/Modify Registers Menu 103

5.11. Remote Command Bypass Feature 111

5.12. Device Interface to CIM 111

6.1. Material Handling System Requests 117

ABSTRACT

The feasibility of design and demonstration of a cell control system to function in the fully integrated manufacturing environment independent of the parts produced or the manufacturing processes involved was investigated. A hierarchical control structure was used. Free standing implementations of a cell controller, a workstation controller, and programmable device interfaces were designed. The system is data driven, and was designed to use the manufacturing databases that exist in the computer integrated manufacturing environment.

Operation of the cell controller and its interaction with the rest of the system was demonstrated in real-time by simulating the computer integrated manufacturing environment on microcomputers connected to each other via communication links.

CHAPTER 1

INTRODUCTION

Computer Integrated Manufacturing (CIM) is the name given to the image of the production facility whose engineering, production and management functions are computerized and combined to achieve efficient operation. Several levels of computerized automation have been implemented in production facilities around the nation. These facilities have been called "flexible manufacturing facilities", "computer aided manufacturing facilities", "computer integrated manufacturing cells", etc.. However, most of these facilities have been solely the integration of a few pieces of programmable equipment, sometimes referred to as "islands of automation". These implementations are far from the fulfillment of the above given description of CIM. To distinguish from these, the facility that fits the above description of CIM is generally termed "full CIM". No commercial implementation of full CIM exists, however there are several limited applications.

Full implementation of CIM demands advanced computer aided manufacturing (CAM) and computer aided engineering (CAE) systems to aid in all aspects of manufacturing planning of the designed part. Products of this system would include flexible routings for production of the part as well as flexible numerically controlled (NC) part programs, fixture and handling information such as gripper and robot approach sequences. These products of CAD/CAE are crucial to achieving the goals of CIM because they introduce the flexibility required for efficient operation of the manufacturing facility. Implementations of many CIM (or FMS) systems have actually been only programmable production facilities rather than flexible ones because of the absence of the above mentioned CAD/CAE capabilities tied to them.

Full CIM implementation generates voluminous amounts of data to be shared among the components of the CIM. Management of this data throughout the system calls for communication networks, database management systems and a complicated control structure. Vendors of manufacturing systems have designed interim solutions to fulfill the requirements of customers. The capital-intensive nature of the automated manufacturing market has prohibited rapid acceptance of state of the art equipment, and developments in the automated manufacturing equipment have led to more advanced islands of automation.

Vendors of automated manufacturing equipment have developed a variety of computer controlled proprietary equipment. Efforts are now concentrated in integration of the facilities without losing the advantages of the multi-vendor environment. The first step in integration was the establishment of connectivity and communication standards. This has been established by the International Standards Organization's (ISO) efforts in establishing the Open Systems Interconnection (OSI) model, and development of the

Manufacturing Automation Protocol (MAP) with the leadership of the General Motors Corporation.

The issues immediately following communications involve the structure of the information flow between the components of the computer integrated factory. Considering the voluminous amounts of data generated by the closely monitored computerized environment, it is obvious that not all data should be passed to every other component. For example, if a machine tool controller monitoring the condition of the cutting tool should try to report the cutting tool dullness to other controllers every fraction of a second, the result would be congestion at every component. Hierarchical structures, where information is digested and filtered at each level, have therefore been used for development of control systems in computer integrated factories.

Several systems for production monitoring and control in automated factories are being developed by vendors of automated machine tools and computer equipment. However, most products being developed address interim needs of the market rather than provide solutions to the problem of efficiently controlling operations in a highly integrated factory environment. While interim solutions to adapt the existing environment and provide smooth transition to the fully integrated environment are essential to the industry, development of control systems for the fully integrated environment is necessary so that the software required to achieve the efficiencies of that environment can be developed and tested. This is an essential step in the justification of the huge investments in fully integrated factories.

OBJECTIVE

The objective of this dissertation is to investigate the feasibility of design and demonstration of a shop floor control structure to function in the fully integrated manufacturing environment. The control structure designed should have the following characteristics:

. Independent of the parts produced or the manufacturing processes involved.

. Utilize the information created by the pre-manufacturing operations in CIM such as design and process planning and should not require extensive post processing.

. Modular so that its components can be replicated to accommodate changes in the system capacity and configuration.

. Expandable so that modification of the algorithms for planning and control of production in the control structure can be easily implemented.

DISSERTATION OUTLINE

The dissertation project can be broken down to the following tasks: (1) synthesis of the existing information on Computer Integrated Manufacturing Systems, (2) development of simulated controllers for programmable equipment, (3) development of the production planning and control method at the cell level, (3) development of the cell control program, (4) design and development of a simulation model of the Computer Integrated Manufacturing environment.

This dissertation is organized as follows. Chapter 2 presents information on the Computer Integrated Manufacturing, hierarchical control, and a review of the previous work. Chapter 3 presents the general description of the environment around the cell controller. Chapter 4 describes the cell controller, discusses the implementation of the cell controller and the details of operation of the cell controller program. Details of the workstation controller and programmable equipment designed for the model workstation are given in chapter 5. Chapter 6 describes the operation of the automated material handling system and the simulation of its operation.

CHAPTER 2

BACKGROUND

COMPUTER INTEGRATED MANUFACTURING

Computer Integrated Manufacturing (CIM) is the name given to the image of the production facility whose engineering, production and management functions are computerized and combined to achieve efficient operation (Figure 2.1). Computer Aided Design (CAD) and Computer Aided Manufacturing (CAM) are the basic technologies underlying CIM.

CAD functions are essential to CIM, since the part description and design information generated at this phase is used for CAM functions. It is essential that the capabilities of a CAD system be used for design of the non-standard fixtures or tools required for the manufacturing of the part. Indeed, it is difficult to determine the boundaries of CAD and CAM in a CIM environment, since functions of both should be used by the design engineer simultaneously. It is expected that a number of subsystems implementing CAD functions would be available to the design engineer, such as expert systems to check whether the part can be manufactured as designed.

After completion of the design of the part, an expert system would create the process plan (or alternative process plans) for fabricating the part. Numerical control (NC) programs, robot approach sequences, etc. would be generated and stored in the manufacturing data base (see Figure 2.2). Group Technology, i.e., classifying parts according to similarities in their physical shape and dimensions, or according to the steps required for manufacturing the parts, is the primary basis for realization of the above functions.

Based on the quantity required and the due date of the order, resources required for the manufacturing of the part, such as the raw material, tools and fixtures, are either manufactured or are ordered for purchase. The cost of manufacturing the part is available as soon as the part is designed and the manufacturing data base is prepared. Analysis of plant capacity and outstanding orders can be undertaken to determine the possible manufacturing lead time for the order. Availability of real-time information is expected to change the whole control structure of the factory, as methods of static planning and control are augmented by algorithms to implement dynamic control of production [21]. The large amount of data produced by the processes and the variety of monitoring functions that can take advantage of the data available makes the local control of production units an attractive choice [4,18]. Allocation of data storage and processing at each level is closely related to the allocation of the control functions which will use them.

When the resources for manufacturing of the part (or group of parts) are ready, the order is released to the shop

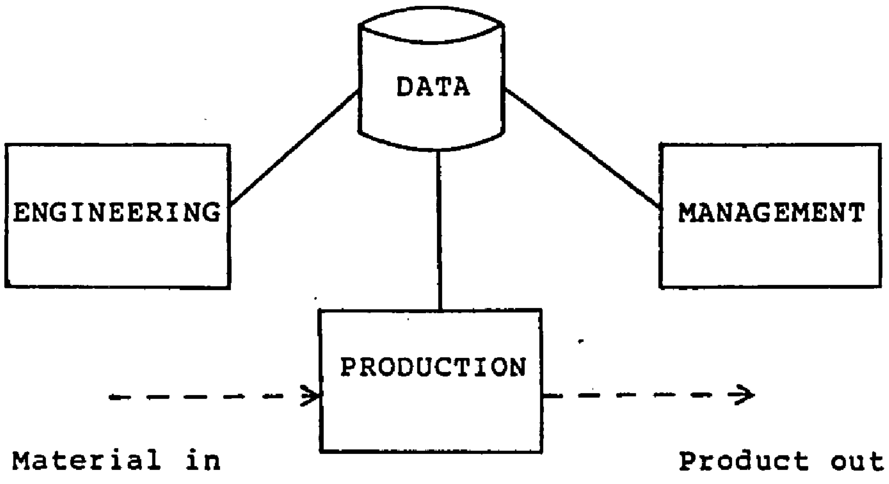
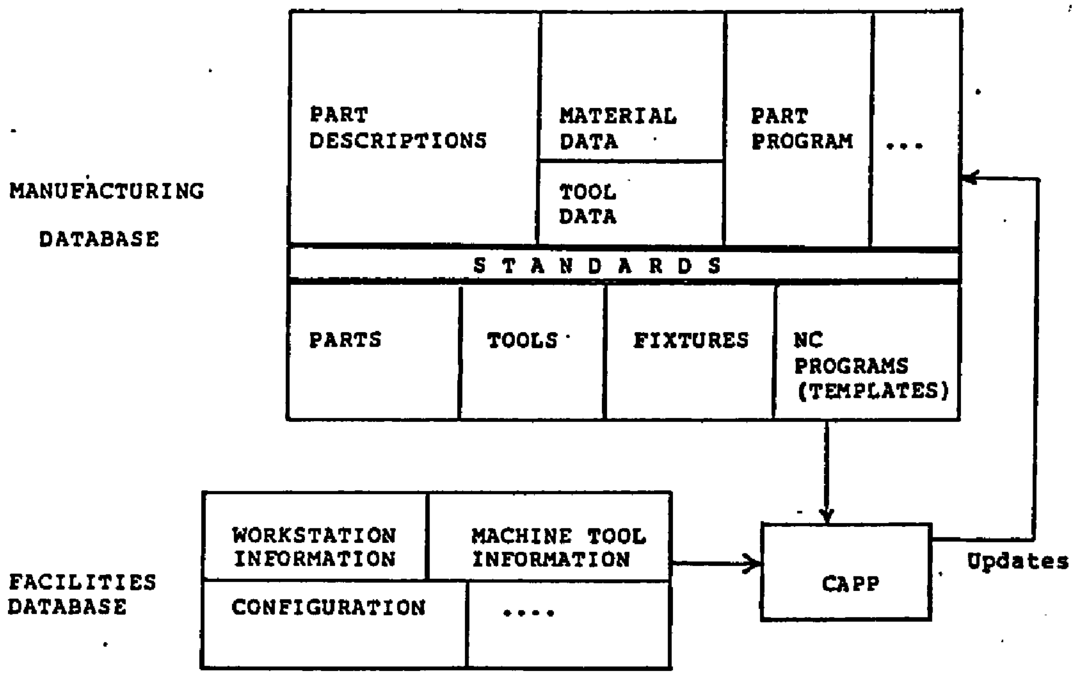


Figure 2.1: Computer Integrated Factory

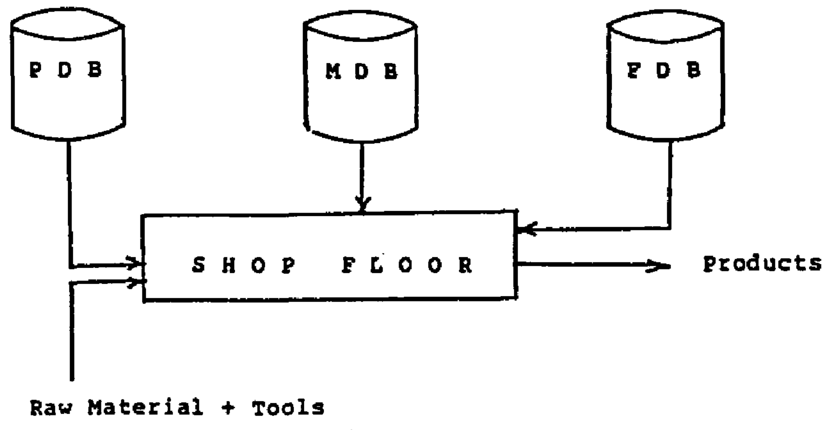


CAPP - Computer Aided Process Planning

Figure 2.2: Manufacturing Database

floor (Figure 2.3). Control of the shop floor is totally computerized, with the information from the manufacturing data base being used for scheduling, and NC programs for workstation control. At the same time, data from the shop floor flows upward to the production control database, enabling monitoring for production control and quality assurance purposes. Development of fully integrated systems should lead to the capability of efficiently manufacturing a variety of products from mid-volume to a lot size of one [27].

Manufactured parts, raw material and semi-finished workpieces are stored in automated warehouses and retrieved when requested by the control system. Automated material handling and warehousing is an integral part of CIM (See Figure 2.4). Workpieces are transported to and from workstations by the Material Handling System (MHS). Automated loading and unloading of parts to the MHS is accomplished by roller carts, robotic manipulators, or whatever method is suitable to the part and the material handling system. Several material handling methods such as conveyors, overhead transports, and automated wire guided vehicles (AWGV's) may be used on the same shop floor (see Figure 2.5). The MHS controller is responsible for making sure that the proper method(s) are used for transport of the workpiece to the desired destination. Detailed description of the physical facilities, such as the transport systems, locations of workstations and the types of interfaces to the



PDB - Production Control Database (schedules, orders, etc.)
MDB - Mnufacturing Database
FDB - Facilities Database

Figure 2.3: Order Release to Shop Floor

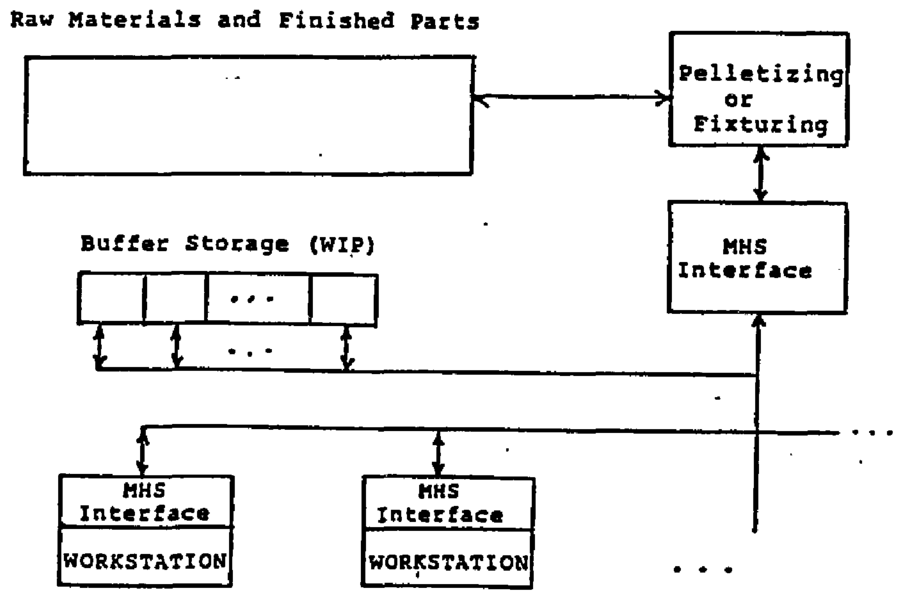


Figure 2.4: Automated Material Handling and Warehouse

MHS, is kept by the CIM in the facilities database. This information is shared by production planning and control, as well as other systems such as MHS for use in performance of their duties.

Another critical component of CIM is a communications network, enabling information flow within the system [25]. It is through this network that the databases for manufacturing and facilities are shared, commands and inputs to the shop floor are sent, and data from the shop floor is transmitted (Figure 2.6). The communications network for manufacturing systems have to be suitable for the type of control system. Varying amounts of data, from simple status indicators to potentially very large NC programs, and in varying urgencies, from emergency handling to routine signals, will have to be handled by the network. The absence of an accepted network standard has been a handicap to the development of CIM systems [11,14]. The Manufacturing Automation Protocol (MAP), a protocol based on the Open Systems Interconnection Model (OSI) of the International Standards Organization (ISO), has recently emerged as a standard accepted by many, and is likely to be the standard for manufacturing systems. Detailed explanation of MAP is given in [13].

Establishment of communications standards for the shop floor has enabled vendors of automated manufacturing equipment to provide communications capability compatible with the others to their products. Given the connectivity of all equipment, the task of controlling the manufacturing system as a whole to achieve optimal performance of operations remains as an area to be researched for a long time.

Most of the systems to control manufacturing opertions that have been developed or are being developed involve hierarchical control structures. The next section presents a review of the hierarchical control, and the section following it describes previous work in this area.

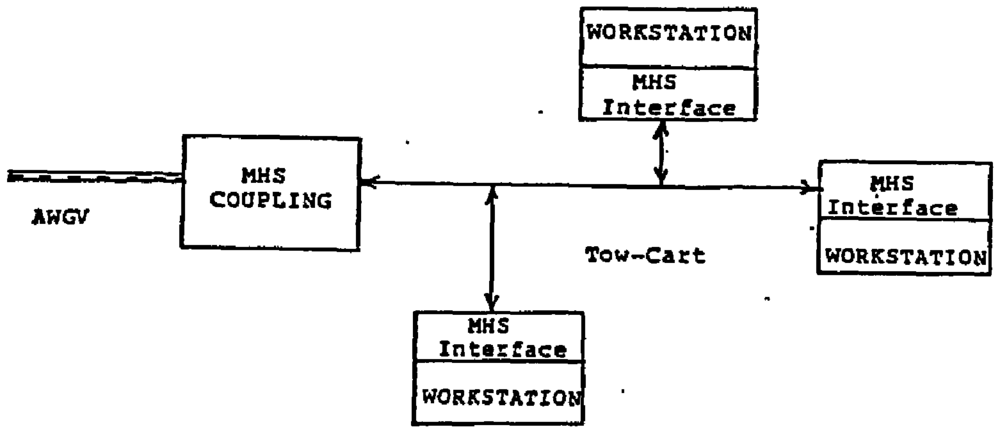


Figure 2.5: Material Handling System Coupling

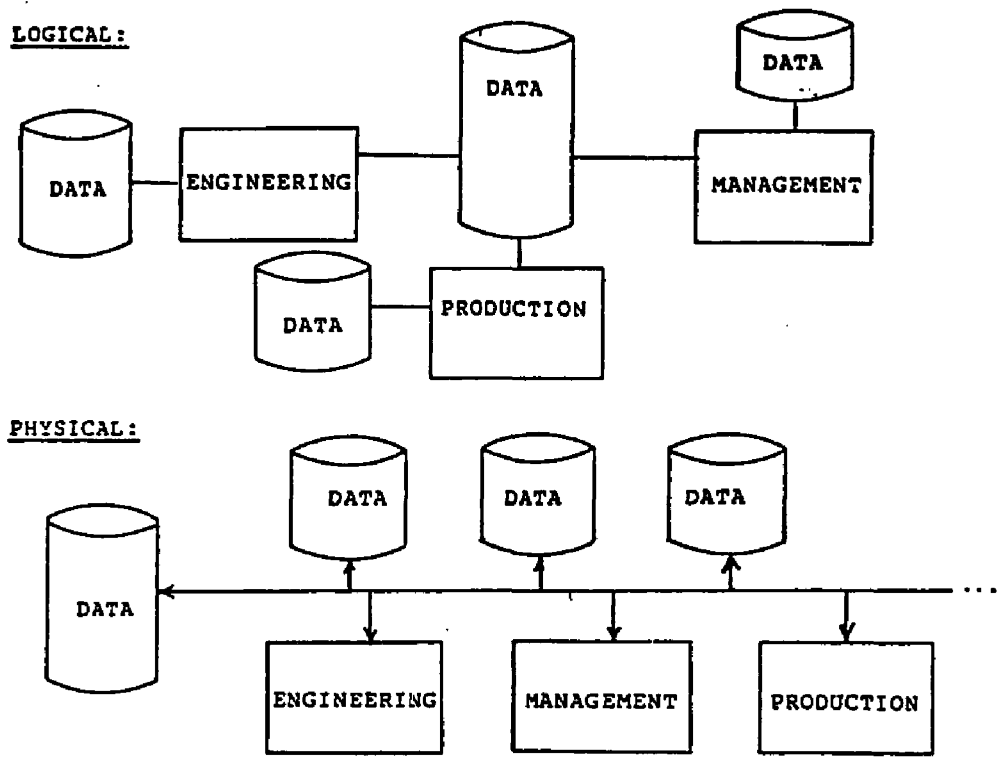


Figure 2.6: Sharing Information Through Factory-Wide Network

HIERARCHICAL CONTROL

Controlling any system of reasonable complexity requires a structure involving delegation of command and responsibility. This hierarchical method of control is most obvious at the institutions such as the army, and less obvious (nevertheless similar) in control of complex processes such as refineries or manufacturing installations. Since most engineered systems can be decomposed into a collection of interconnected subsystems, each of which can be controlled individually, it may be possible to control such systems by generating hierarchies.

A simple hierarchy is shown in Figure 2.7. Fundamental properties of a hierarchy [26] are as follows:

1. Hierarchies consist of decision making units arranged in a tree-like structure where at each level a number of such units operate in parallel.

2. Hierarchical structures exist in systems which have an overall goal, and the goals of all the decision makers who constitute the hierarchy are in harmony.

3. There is an iterative information exchange between the decision making units on the various levels of the hierarchy with a precedence for the information going down which is treated as a command by the lower levels which try to obey it if they possibly can.

4. The time horizon of interest increases as one goes up the hierarchy.

Hierarchies arise due to the following reasons:

a) The system having a definite goal is too complex for one decision maker to comprehend let alone control, since decision makers have limited information handling capacities.

b) Since time flows sequentially, it is possible to perform more tasks in a given period of time if the jobs are done in parallel and this leads to parallel decision making by decentralized controllers.

c) Decentralized decision makers need to coordinate their activities to satisfy the overall goal and it is more efficient to have a specialist coordination function and a hierarchy than constant communication between all the decision makers since it increases the burden on each decision maker.

Advantages of hierarchical systems are:

a) Flexible configuration, and the possibility of increasing capacity easily.

b) High reliability due to ease in adding parallel redundant systems.

c) Lower cost due to simpler software and possible use of standard components at each level.

Singh [26] presents the problem of synthesizing hierarchical structures for large interconnected hierarchical systems. Formulation of the problem for a collection of N interconnected dynamical systems, where the system can be described by linear differential or difference equations, and the cost function which defines the overall goal of the hierarchy is a quadratic function of the states and controls is shown in Figure 2.8.

For the ith subsystem:

$\underline{x}_i$ is an n_i dimensional state vector

$\underline{u}_i$ is an m_i dimensional control vector

$\underline{z}_i$ is an r_i dimensional vector of inputs from other subsystems.

Assuming linear system dynamics:

$$\dot{\underline{x}}_i(t) = \underline{A}_i\underline{x}_i(t) + \underline{B}_i\underline{u}_i(t) + \underline{C}_i\underline{z}_i(t) \qquad \text{----------} \ (1)$$

with $\underline{x}_i(0) = \underline{x}_{io}$

Assume that the vector of inputs $\underline{z}_i$ is a linear combination of the states of the N subsystems:

i.e., $\underline{z}_i = \text{sum}(L_{ij}\ \underline{x}_j)$ for $j = 1$ to N ------ (2)

It is then desired to choose the controls $\underline{u}_1,\ldots,\underline{u}_N$ in order to minimize the function of the kind:

$$J = \sum_{i=1}^{N} (1/2\ ||\underline{x}_i(T)||^2\ Q_i + 1/2[\ ||\underline{x}_i(t)||^2\ Q_i + ||\underline{u}_i||^2\ R_i + ||\underline{z}_i||\ ^2\ S_i\ ]\ dt$$

subject to constraints (1) and (2).Several approaches to solve this problem, both feasible and infeasible are given in [26]. A similar formulation of the same kind of problem is presented in [28] to minimize the Work-In-Process, thus optimizing the production planning and control function.

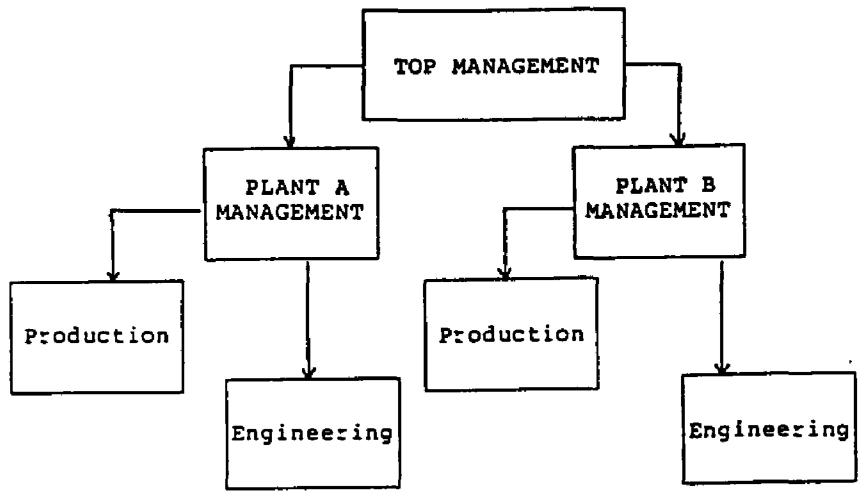


Figure 2.7: A simple Hierarchy

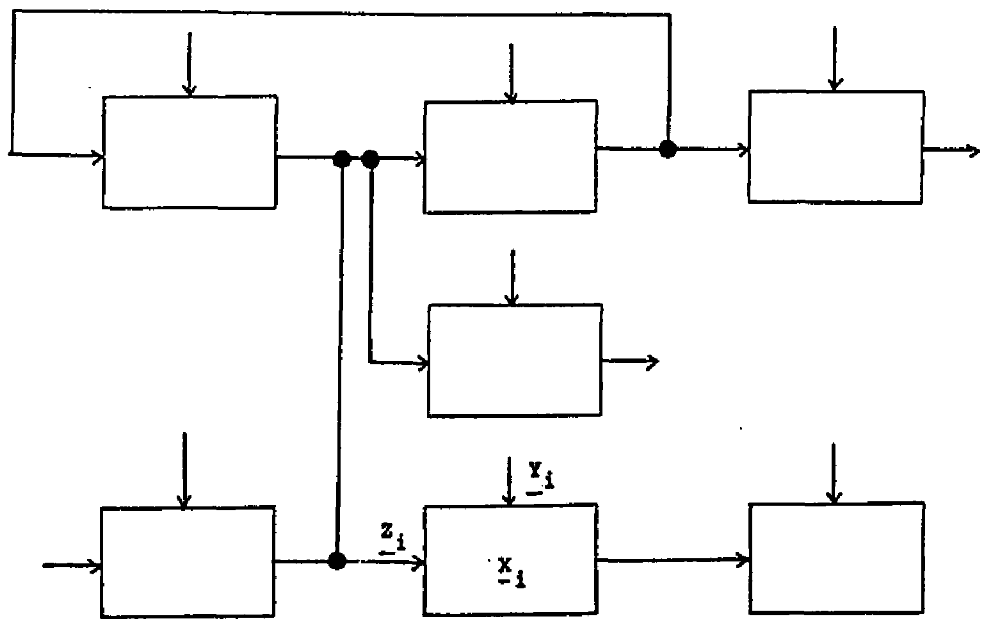


Figure 2.8: An Interconnected Dynamical System

Each work section in layer j is represented by a similar equation, and the objective function is of the same kind. It is stated that the result would be impractical to apply, since it inevitably yields an approach too cautious to apply, reserving an unused margin to counteract the event effects. Thus the solution would yield an inefficient production planning and control system. The difficulties of multivariable optimization in real-time are also discussed in [19].

An approach to solving problems of reasonable complexity is to map solution strategy by decomposing the problem at hand to subproblems that can lead to the solution of the main problem if they can be solved. These problems than can be further decomposed into subproblems themselves, using the same approach. At some point in the decomposition process, subproblems whose solutions are known are generated, and thus the solution to the principal problem is obtained. The decomposition of problem A to subproblems B and C (solution of either one would lead to the solution of A), and then to subproblems D,E,F,G, and H are shown in Figure 2.9. AND nodes are shown with a bar joining their arcs, the others are OR nodes. Solutions to primitive problems D and E, or to F,G and H yield the solution to the main problem A. AND/OR decomposition is a valuable tool for analyzing hierarchical networks.

This mapping of the solution to problem A is relevant for most purposes, mainly for planning, however is not readily applicable to real-time control systems because the variable time is not explicitly represented. A method to convert these AND/OR graphs to a continuous form in which time is explicitly represented is necessary to apply the ideas to hierarchical control systems. Such a method has been developed at the National Bureau of Standards (NBS) [2]. It defines a mapping H from input space to output space, where the input vector $\underline{S}$ can be decomposed into command plus feedback:

$$\underline{S} = \underline{C} + \underline{F}$$

Thus the input vector describes the possible input vectors(S) corresponding to the same command (C), and the outputs to these inputs (See Figure 2.10):

$$\underline{P} = H(\underline{S})$$

The input vector $\underline{C}$ represents a goal and the output vector $\underline{P}$ may be subgoals to the several levels below. In this case the function H may be viewed as a decomposition function where the command $\underline{C}$ is decomposed into subtasks. In another case, this system may be a servomechanism where $\underline{C}$ is the setpoint and feedback is used to compute the error signal. In this case H is the transfer function and the outputs are the drive signals to physical actuators. When the feedback (or the error signal) is not continuous but discrete (such as measurements at discrete points of a continuous variable), then the continuous analysis degrades to a discrete one. As long as the H functions are correctly

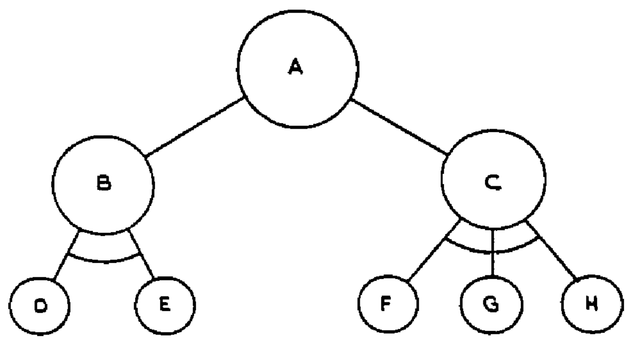
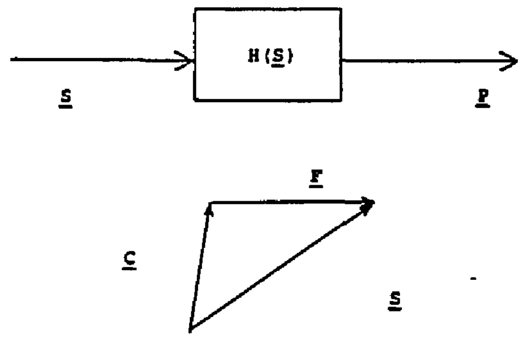


Figure 2.9: A Simple Problem Decomposition
Solution to A: (D and E) or (F and G and H)



S is the STATE vector
C is the CONTROLS Vector
F is the FEEDBACK Vector

Figure 2.10: A Simple Control Unit

formulated, and the sampling of feedback (error) is frequent enough, stability of the system can be maintained.

A hierarchy of task decomposition operators, or servomechanisms, can be used to describe the controls for a complex system (see Figure 2.11). Feedback enters this hierarchy at every level. At the lowest levels, the feedback is unprocessed, or nearly so, and hence is fast acting with very short loop delays. Feedback therefore closes a real time control loop at each level in the hierarchy. The higher level loops are more sophisticated and slower. The time rate of change of the output vector $\underline{\mathbf{P}}$ will be of the same order of magnitude of that of $\underline{\mathbf{F}}$ and significantly faster than that of the command vector $\underline{\mathbf{C}}$. This slower time rate of change of the $\underline{\mathbf{P}}$ vectors at the higher levels is not because the processing rate of the higher level H operators (which indeed is the same as any other), but due to the fact that the $\underline{\mathbf{F}}$ vectors driving the higher levels convey information about events which occur less frequently. In some cases higher level $\underline{\mathbf{F}}$ vectors may require the integration of information over long time intervals.

The composition of the feedback at each level, that is, the amount of sensory information from the environment or the lack of it, will determine the sensitivity of the controls at that level to the conditions in the environment. If the feedback vector contains many external variables, the task decomposition at that level will be capable of responding to the environment. If the feedback vector

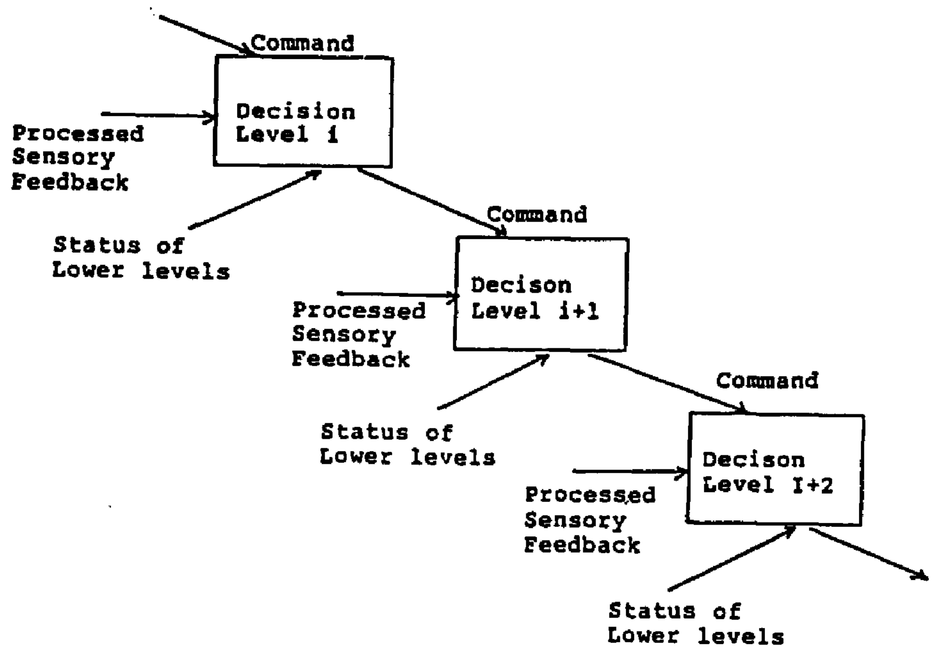


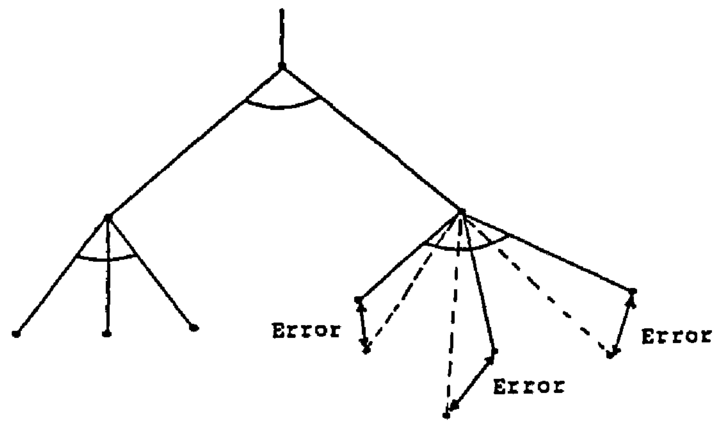
Figure 2.11: Feedback to Hierarchical Control System

consists solely of internal variables, then the decomposition at that level will be stereotyped and insensitive to the conditions in the environment.

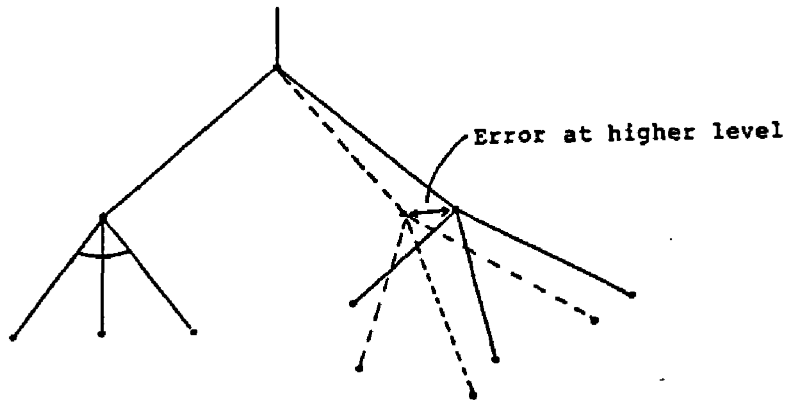
The success of performance of tasks depends on the capability of the H functions at each level to provide the correct mapping despite perturbations and uncertainties in the environment. To maintain control, the transfer functions must be defined around the regions of perfect performance as well as the expected points, and must be able to direct the actions to correct the deviations in the perfect performance to maintain stability. If they fail to perform in the presence of perturbations, then the system fails.

Small perturbations can be usually corrected by low level feedback loops, since they require relatively little sensory data processing (see Figure 2.12a). Larger perturbations in the environment may overwhelm the lower level feedback loops and require strategy change at higher levels in order to maintain the system within the region of successful performance. These are shown in Figure 2.12b. The changes in the environment beyond the correction capabilities of the lower level controllers is detected at the higher levels and new command vectors for the lower level controllers is calculated, representing a change in the strategy to cope with the perturbations.

Integrated factory models have been developed by several institutions utilizing the theory of hierarchical control discussed above. These models are described in the next section. Hierarchical control theory was also utilized for development of the model in this research.



a) Small perturbations corrected at lowest level



b) Lower level overwhelmed by the error (change in strategy)

Figure 2.12: Correction of Performance

PREVIOUS WORK

Several institutions around the world have been researching the possibilities of designing control structures for efficient control of automated manufacturing facilities. This section presents brief descriptions of major relevant research in this area.

Most notable is the research continuing at the National Bureau of Standards Automated Research Facility (NBS-AMRF). It has been documented in [1,2,3,16,17]. Following is a description of the model and current implementation of the AMRF control system.

This facility is being developed by the NBS as a testbed for developing standards in all aspects of managing the factory of the future. The production control model developed at AMRF consists of five layers: facility, shop, cell, workstation, and equipment (see Figure 2.13). It is a hierarchical control structure, in which the commands to each control module is processed in a similar manner. The current input command to a level is decomposed into procedures to be executed at that level, to commands to be issued to the lower levels, and to feedback to be transmitted to the supervisory level. The feedback is used to close the control loop at each level, supporting the adaptive behavior of the system (see Figure 2.14).

The facility control system implements the highest level of control, and has a planning horizon of anywhere

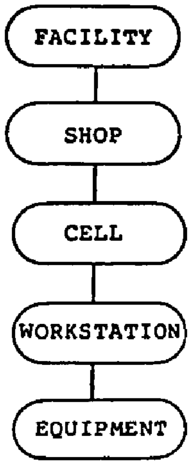


Figure 2.13: AMRF Control Levels

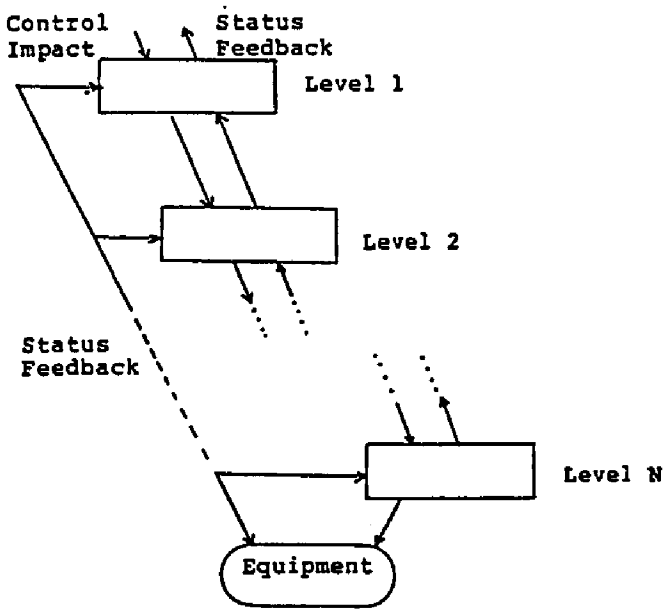


Figure 2.14: AMRF Control System Hierarchy

from several months to several years. This level is broken down to three major functional areas: manufacturing engineering, information management, and production management. Manufacturing engineering involves generation of bill of materials for assemblies, as well as the process plans necessary for the manufacturing of parts. Information management provides the user and data interfaces for the necessary administrative and business management functions such as order handling, billing, payroll, etc. Production management generates long range schedules, determines the need for capital investments to meet production goals, and summarizes production data. The long range schedules produced by this system are used to generate work orders at lower levels.

The shop control system is responsible for coordinating the production and support jobs on the shop floor. This system is also responsible for the allocation of resources to those jobs. The planning horizon for the shop control system varies from several weeks to several months. Major components of this system are task manager and resource manager. The task manager schedules job orders, maintenance and shop support services. It also tracks equipment utilization and schedules preventive maintenance for equipment and tools in the factory. The task manager is responsible for capacity planning, grouping orders into batches, activating and de-activating "virtual" cells, allocating resources to individual cells and tracking individual orders to completion. Resource manager allocates workstations, buffer storage areas, tooling etc. to cell level controllers. It also updates inventories of all equipment and tools in the factory. The dynamic allocation of workstations to cell controllers makes it possible to change configuration of cell compositions to fit the production requirements.

The cell level controllers are responsible for sequencing batches of jobs through workstations, and supervising various other support services, such as material handling or calibration. The planning horizons of cell controllers vary from several hours to several weeks. Components of the cell control system perform task decomposition, analyze resource requirements and prepare requisitions, report job progress and system status to shop control, make dynamic batch routing decisions, schedule operations at assigned workstations, dispatch tasks to workstations, and monitor the progress of those tasks.

The workstation control system has a planning horizon from several minutes to several hours. Activities of small integrated physical groupings of shop floor equipment are coordinated and directed by this level of control. A typical workstation consists of a robot, a numerically controlled machine tool, a material storage buffer and a control computer. Interface of the cell to workstation controllers is designed to be independent of the buffer and a control computer. The of the cell to workstation controllers is designed to be independent of the workstation type so that the assignment of the workstations to the cell can be made without difficulty.

Equipment Control Systems are closely tied to the commercial equipment or industrial machinery on the shop floor. These controllers have a planning horizon of several milliseconds to several minutes. They in fact are the interfaces of the commercial equipment to the workstation control system.

Every control module in the AMRF hierarchy reacts to inputs in essentially the same way: input commands from the supervisory level are decomposed, status feedback from subordinates are processed, and new outputs in the form of commands and status are generated (see Figure 2.15). This mode of operation, referred to as reaction, represents the first of several levels of intelligent control envisioned: reaction, planning, optimization, and learning (see Figure 2.16).

To aid in specifying the required task decomposition and task processing, a programming language and program development environment called the Real-time Control System (RCS) was implemented. It permits specification of programs at each level as state tables, and the programming environment permits the generation, editing, emulation, and evaluation of these state tables. Details of this system are given in [3,15].

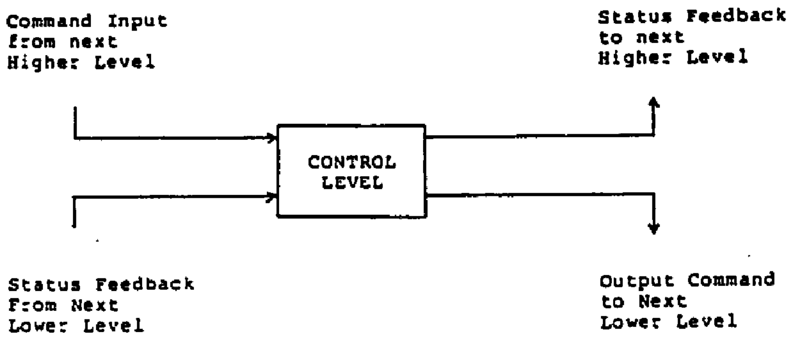

Command Input from next Higher Level
Status Feedback to next Higher Level
CONTROL LEVEL
Status Feedback From Next Lower Level
Output Command to Next Lower Level

Figure 2.15: Generic Control Module

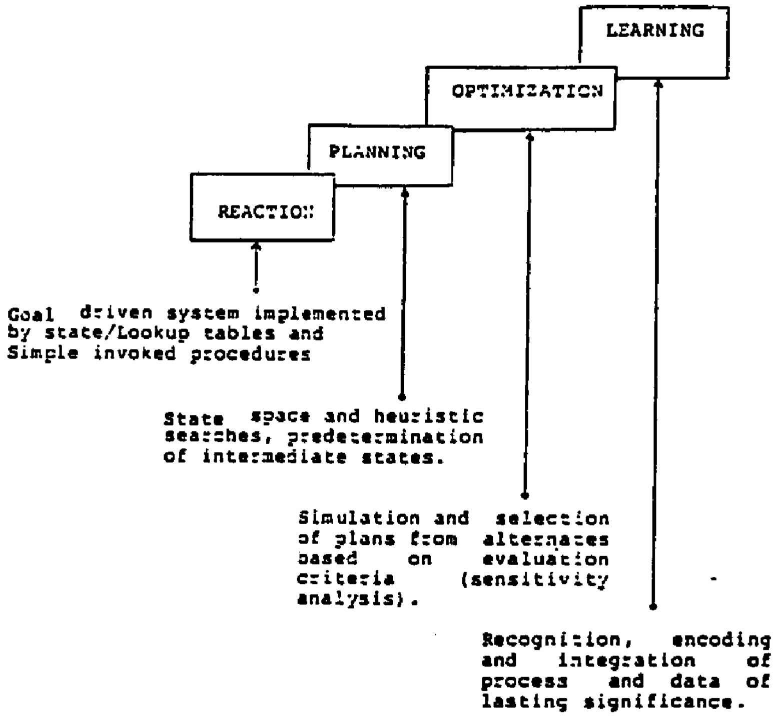

LEARNING
OPTIMIZATION
PLANNING
REACTION
Goal driven system implemented by state/Lookup tables and Simple invoked procedures
State space and heuristic searches, predetermination of intermediate states.
Simulation and selection of plans from alternates based on evaluation criteria (sensitivity analysis).
Recognition, encoding and integration of process and data of lasting significance.

Figure 2.16: Evolution of an Intelligent Automated Control System

As it is currently implemented, the AMRF consists of a cell control system which receives commands from an operator interface. It coordinates operations of a horizontal, a vertical, and a turning machining centers, and a material handling system (see Figure 2.17). Each machining workstation manages four equipment level systems, a robot, a machine tool, fixturing devices, and a local material storage area. The material handling system manages a robot cart, a storage and retrieval system, and a loading/unloading area that is tended manually. Several programming languages and environments have been used in the implementation. The cell control system and the material handling system have been implemented on a minicomputer. Systems running on a computer use memory locations labeled as mailboxes for communication. A network is planned for communication of processes not implemented on the same computer.

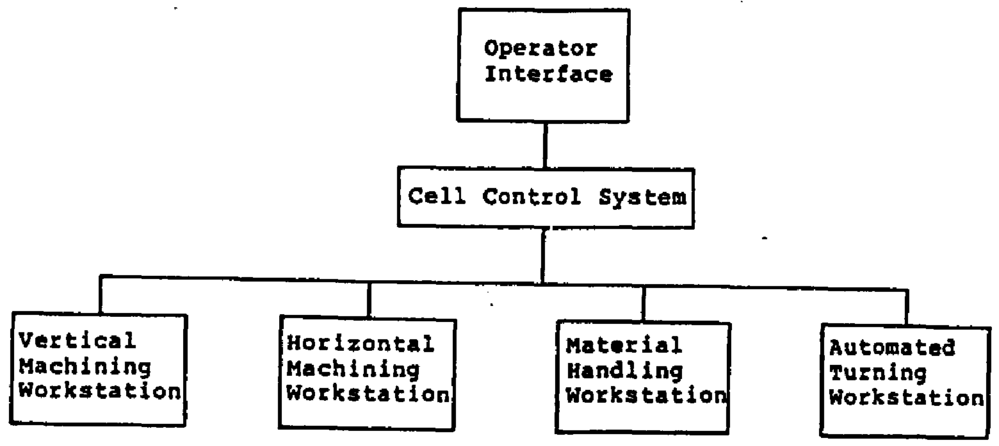

Operator Interface
Cell Control System
Vertical Machining Workstation
Horizontal Machining Workstation
Material Handling Workstation
Automated Turning Workstation

Figure 2.17: Current AMRF Control Structure

The AMRF is being developed to fulfill the goal of the NBS, to develop standards for automated manufacturing. It therefore considers the interim needs of the industry as well as the future. The interfaces to the operator and other manual operations such as programming are adding to the complexity of the systems being developed. The concepts being implemented, however are valid in either case. The programming systems developed involves forms of many types that are efficient for manual data entry, but would require additional processing of the available data to be generated by a fully integrated system.

Use of state tables in the decision making elements may cause fragmentation of the problem because of the exponential increase in the possible number of states with the number of elements to be controlled. As long as the implementation of decision makers is on one computer, the real-time nature of the response can be preserved. If however the state table based decision makers are to be implemented on multiple computers, it might not be possible to respond in a reasonable time frame because of the large number of transfers required between the decision makers.

The structure planned for AMRF is a typical hierarchical control system for CIM except for the MHS. The implementation of the material handling system as a workstation to the cell controller is acceptable for a facility consisting of one cell, however it would not be practical in the presence of many cells, especially in an environment where the workstations are allocated to cells by an auctioning scheme. Rather, it would have to be an autonomous controller accepting requests from all cell controllers, and responding to them.

Development of another hierarchical control system to carry out FMS integration is given in [21]. This system involves following basic levels: dynamic scheduler, process sequencer, resource allocation, and the communications level. The dynamic scheduler determines the instantaneous production rate of each part type, planned routing and dispatch timing to best utilize the the varying capacity of the system. It uses real-time data as well as the aggregate data to make best use of the manufacturing system's flexibility. The Process Sequencer infers, based on the manufacturing system status and the production state of the part, the next process, the appropriate material handling move, and the production program to download, if required. Major components are the knowledge base, where the facts and the rule set reside, local data base where the system status and production state are held and the inference engine (IE). IE presents the current system status and the production state to the facts and rules, to come up with a set of actions whose conditions are fulfilled.

The IE consists of four components: IE flow control manages the program execution and interacts with the rest of the software control. Local data base manager updates and enforces consistency when new data arrives, internal scheduler controls the order of the rules. The Interpreter actually parses the rules using the facts and current data .

The objective of the Dynamic Resource Allocation Module is to dynamically allocate the shared resources according to the next task to be performed in the FMS, complying with some priority policy. The output of this level are commands to the communications interface module, in order to execute particular movements.

The communication level transmits the decisions and receives feedback from the direct machine controllers. Part of this level is also in charge of collecting statistical data, monitoring options of the system, and providing run-time services.

The control system design outlined above may be implemented with success and work effectively for an FMS system with limited capacity. It however would be difficult to implement this system in CIM environment, because the design is an integrated control system, instead of a distributed one. The allocation of the scheduler and other functions would have to be redefined given the necessity for grouping of workstations (or cells) in a factory wide implementation. The data required by this system, namely the rules for the inference engine and the program for the FMS control system, are not the type of information found in the CIM environment naturally. They are the products of additional processing of the production and product data bases in CIM.

[14] gives a comparison of the products available from vendors and address the issues of data compatibility and communication in an integrated environment. It is stated that the systems available from vendors are all in development stage and mostly involve implementation of the higher-level functions such as production scheduling and inventory control than the cell or workstation level controllers. It is also observed that the products entering the market stress color graphics displays, and sophisticated user interfaces rather than providing solutions to the problem of distributed control in the shop floor. Authors suggest a modified disk operating system type of approach to the needs of a distributed control system on the shop floor.

Effective design of the control systems for Flexible Manufacturing Systems is discussed in [5]. A conceptualization of FMS environment is outlined and two control structures, a single level and a two level are described. An implementation of the two-level control system is also detailed. The control system described here is based on the use of a procedural language (termed CPL, Cell Programming Language) for the manufacturing cycle. For each piece to be manufactured in FMS, a program must be written. From the language point of view, FMS is a set of devices classified according to type and features. CPL consists of instructions to request services or information from devices, and instructions to reserve, activate or gain exclusive use of a device.

The supervisory level controller consists of an operations sequencer, a part loading module, and a table manager. Three tables, active missions (workpiece tasks) table (AMT), active step (operation request) table (AST), and a table describing the current state of the devices in the system (CDT) are kept by the supervisor. The interfacing of the lower level (devices) with the controller is through the common tables, particularly the AST.

The system described for control of FMS utilizes cell controller programs for each part in addition to the part programs for manufacturing processes. The cell control programs, written in Cell Programming Language (CPL), are then interpreted by the cell controller and reduced to a set of common tables through which the workstations are assigned, activated, etc. One of the preliminary advantages cited for the system developed here is the ease of programming by writing macros in a general purpose high level language.

In the full CIM environment, the cell control program for each piece would have to be written after the creation of the process plan for manufacturing of a part. The fact that all the information in the CPL program will be derived from the manufacturing and facilities data bases in the CIM environment, and will be reduced back to a table form in the cell controller, suggests that it is a step redundant in the CIM environment where data communications provide global access to common databases. Furthermore, the integrity of data in an environment where different forms of the same information are present would be hard to preserve. The operational control system so provided, therefore, may not be as efficient in the Full CIM environment as it might be in the FMS where the pre-manufacturing activities may not be automated or integrated to the shop floor control.

A generalized hierarchical control structure for machine control applications is presented in [20]. Five major components are identified: command translator, command interpreter, device manager, exception monitor and subsystem monitor. The command translator serves as the communications interface between the external host and the machine controller. The command interpreter breaks down the incoming command into elementary commands for further processing by the device managers. The device manager executes the incoming command by directly operating a hardware device or by breaking down the command into more elementary ones for further processing by the lower level device managers. The exception monitor gathers and reports error conditions detected by the subsystem monitors. The exception monitor can command the machine to stop if the conditions so dictate. The subsystem monitors process the inputs and monitor the conditions of the passive hardware systems, such as position calibration.

Control hierarchies may be built by using the same controller model at each level. The termination point of the control hierarchy is where the command action can be executed by operating a single device in a simple sequence of actions. Examples of similar hierarchical control designs are cited (one by Albus, Fitzgerald and Barbera, and another by Gomoa). The command processing chain of a hypothetical application of the generalized hierarchical structure is described. Hierarchical control structure by Gomoa is shown in Figure 2.18.

The described generalized control module can be implemented in many different ways and still fit this framework. This can be observed by comparing the examples cited and the hypothetical example given. The control system designed does not show the lower level controllers which would have to be different due to the characteristics of the lower level inputs. Authors stress the importance of the messaging service in the design of the controller hierarchies and suggest use of a queuing system for messages and use of semaphores.

Another approach to solution of the production planning and control problem using hierarchical dynamical control structure is given by [28]. A tree organized Generalized Hierarchical Structure of Decision Making Modules (expert systems) is suggested. It involves definition of a set of solution procedures, parameterized on the possibly occurring event types, for all sub-problems within the hierarchy, and

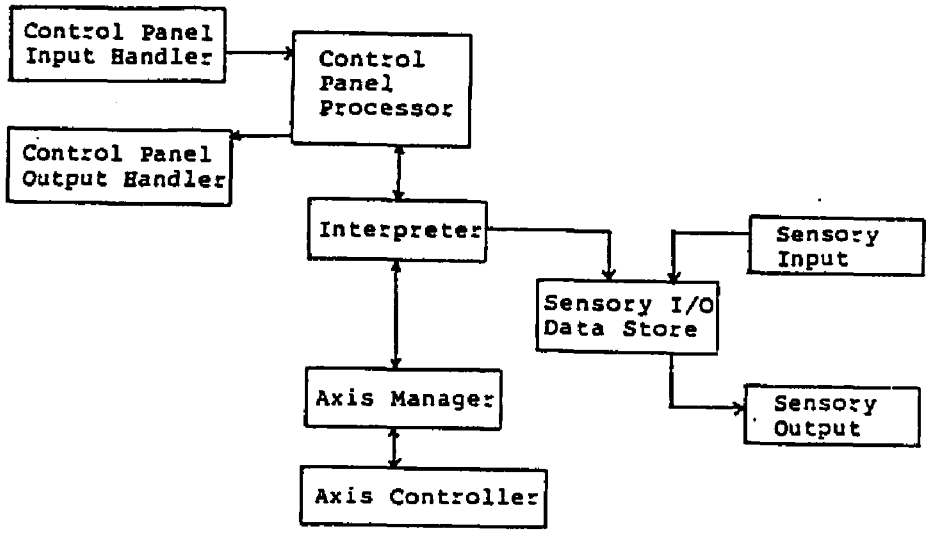


Figure 2.18: Hierarchical Control Structure by Gomoa [20]

the organization of them into a relational framework able to make event to control strategy matchings. Figures 2.19 and 2.20 show the block diagram of a generalized hierarchical structure. The resulting Generalized Hierarchical System then can be viewed as a network of expert control systems, each consisting of an on-line learning process, an on-line local planning design process, and an on-line coordination process.

Another expert system based control hierarchy applied to a flexible assembly cell is presented in [8]. NNS is a complete on-line system for control of multi-robot assembly workcells. The designed control hierarchy (see Figure 2.21) covers three levels of abstraction: task level, functional

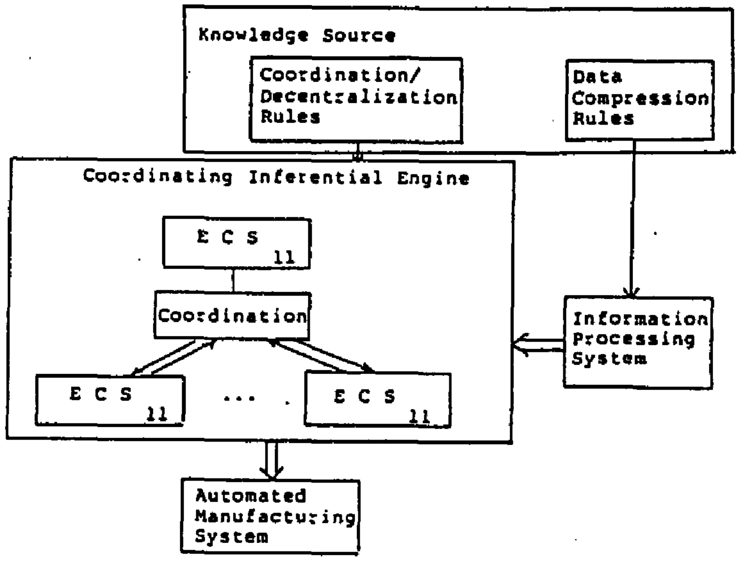

Knowledge Source
Coordination/ Decentralization Rules
Data Compression Rules
Coordinating Inferential Engine
E C S 11
Coordination
E C S 11
...
E C S 11
Information Processing System
Automated Manufacturing System

Figure 2.19: Generalized Hierarchical Structure of Expert Systems [28]

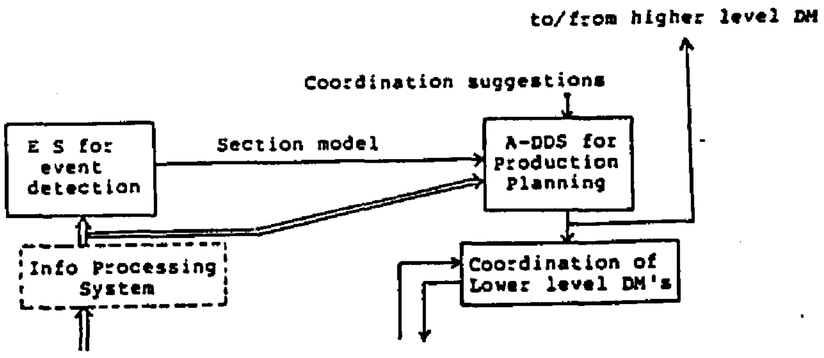


Figure 2.20: Expert Control System Block Diagram [28]

level, and command level. The task level reasons about what happens in the cell and what the system has to perform. It involves planning, failure analysis, interface with higher level systems (operator, workshop), execution monitoring and action generation. The functional level is in charge of action execution, in terms of functional primitives available in the cell. It consists of two modules: the sequence manager and the interpreter. The command level interacts directly with the controllers of the components of the cell, such as the robotic manipulators, or sensors.

NNS is implemented as a set of processes running on a number of processors. Interprocess messages are used for communications. Each component of the main cell is controlled by a specialized module, implemented as an independent low level interpreter. The specialized modules have access to the state of the cell. The functional and higher levels are implemented as processes in Lisp running on a minicomputer.

This implementation of an assembly workcell uses knowledge based decision makers at higher levels and interpreter-monitors at lower ones to execute action requests. The state of the flexible assembly cell is maintained, together with the rules, by the decision makers. Planning and monitoring is done based on the interpretation of the state of the system and the rule base.

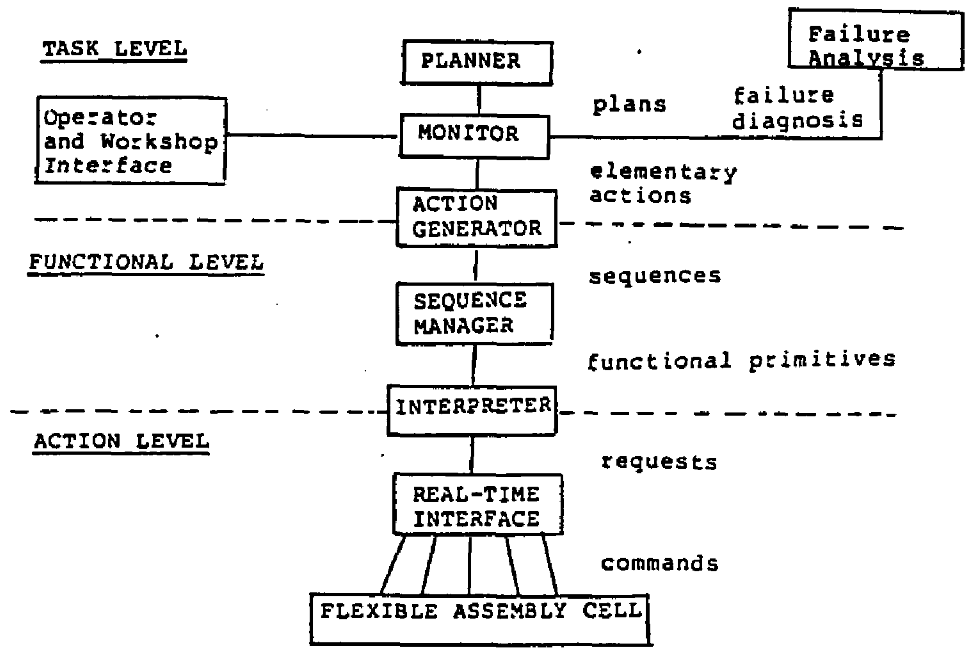


Figure 2.21: Organization of NNS [8]

Expert systems are expected to be a part of solution to almost any problem in the automated manufacturing area. However, given the complexity of the CIM, it is not possible to formulate the rules to manage the whole system, hence to build an expert system to manage CIM optimally. Introduction of expert systems to the manufacturing control systems therefore has to be in the context of specialized decision makers at several points in the hierarchy. The information required by these systems may then be supplied by the experts in these areas. The discussion in [28] does not address the problem of relating the types of data and rule sets required by the proposed control system to those generated by the CIM during pre-manufacturing operations.

Several other commercial implementations of flexible manufacturing cells such as the ones in Caterpillar, Renault [12], and Martin Marietta [23] are reported. These implementations are geared to solving particular problems in well defined product classes and have controllers custom designed for the particular implementation. The emphasis is in improving productivity and quality by semi-automated manufacturing groups. The requirements for unmanned operation of the flexible manufacturing facilities and the difficulties of conversion to the fully unmanned factory are discussed in [18]. There are also several implementations of traditional software for production control and scheduling that has been re-written on microcomputers, using the improved user interface of these machines [7]. The problem of dynamically coordinating the shop floor operations to achieve goals of production have been totally overlooked by such implementations, and the real-time nature of the control systems have been ignored.

Many other organizations such as the consortium of companies and schools in Europe [9] are investing in research in this area but few results have yet been published. There are no published reports of an implementation of a cell control system that is modular, de-centralized, portable and data driven to efficiently utilize the highly standardized fully computer integrated manufacturing environment. The contribution of this research is expected to be a demonstration of the feasibility of such a system.

CHAPTER 3

SIMULATION OF CIM ENVIRONMENT

GENERAL

An implementation of "Full CIM" involves computer aided design and manufacturing systems, computerized planning and tracking of orders, inventory, costs, quality, etc. This research involves one small portion of this integrated system, namely the manufacturing cell controller. It is unrealistic, if not impossible, to describe the cell controller and the manufacturing unit without making some assumptions about the environment in which it operates. To simulate one segment of an integrated system, it is necessary to define the interfaces of the simulated portion to the system. This enables us to determine the inputs of the environment to the segment of interest, as well as outputs of the simulated segment to the others. It is therefore necessary to describe the domain of interest and simulate the environment surrounding it, at least as far as the interactions are concerned.

The manufacturing cell controller program runs on a microcomputer. It is connected to another microcomputer which is capable of simulating the CIM environment's responses. These include responses expected from the workstations, supervisory level controllers and the material handling system controller.

One crucial requirement of the CIM environment is the presence of certain information at each level of operation. It is therefore assumed that the information required at each simulated level is available through the factory-wide network. In the absence of the network, this data is read from local disk or obtained through the serial communications link connecting elements of simulated environment. Much of this data, such as NC programs, process plans, etc., are actually products of the integrated system itself, and are assumed to have been created previously.

A program running on a microcomputer simulates the gateway to the world external to the cell controller. It handles all communications in and out of the cell controller, as well as coordinating or simulating the Material Handling System (MHS), network controller, workstation controllers, and provides an interface to the operator. Commands from the control levels above the cell controller are entered using the operator interface provided on the microcomputer. The cell controller responds by issuing commands to the workstation level controllers and commands to the Material Handling System, as well as status reports to the supervisory levels.

The CIM environment outside the cell controller can be simulated on the microcomputer acting as the gateway, in which case the simple modules built into this program will be used, or some of the units such as the material handling system or the workstation controller may be simulated on separate computers. If external simulation programs are attached, the gateway simulator program will act as a link between those and the cell controller.

This flexible design of the system makes it possible to use it in an environment when some of the simulations of systems and workstations is not made on separate computers, but on the gateway computer. It is therefore possible to run the system simulation using only two computers.

The gateway simulator program and the limited simulation of external systems, such as the workstation controllers and material handling system controller, within the context of this program, are described in this chapter. Following chapters present the manufacturing cell control program, the workstation controller program and the material handling system simulator program.

GATEWAY SIMULATOR PROGRAM

Since a variety of functions are assigned to it, the simulation program is designed in a modular fashion, each function implemented as a set of modules executed only when necessary, and returning to the monitor program without suspending the user interface. This means that unless it is absolutely necessary, functions do not do user interaction, so that the simulation of the operation of the cell can be carried out in real time.

The simulator program operates with queues. Each operational message received is placed in the appropriate queue. Informative messages are displayed (and lost) after the statistics are updated. Event times are determined when messages to start operations ending with events (such as workstation operation, piece transport, etc.) are received. The operator can override the scheduled event times, as well as causing things to happen, even if not scheduled. Simulated occurrence times of the events in the queues are checked periodically, and those events whose times match the current time are executed in the form of responses to the cell controller from the appropriate unit.

Simulator program operates in synchronization with the cell control program (CCP) running on the remote computer. Upon initialization, CCP issues a synchronization signal, which is acknowledged by the simulation program, and the two programs reset their system timers.

Data for Simulation

The simulator program contains some of the data that is required for running the simulation, however most of the Manufacturing Data Base, which would otherwise be present in the network, is not fed to the simulator. This avoids duplication of the data that could otherwise introduce integrity problems into the system. The remaining data (those available at the databases of the cell controller program) are requested from the CCP in form of database queries initialized by the simulator program itself. Other parameters required by the simulation program are passed to it by the CCP using fields in the messages that otherwise would be disregarded by the simulator, but would have to be sent anyway to comply with the messaging format of the communications system. One example is the second operand in the "start operation" message. It normally would be ignored by the receiving party, but is actually used to convey an expected processing time to the simulator.

Simulation of the Material Handling System controller requires presence of facility data in the simulator. It is fed into the simulator from a file kept on the same machine, and can be changed using an editor. Since changes in the facilities is not normally expected, no provision exists for interactive alteration of the facilities database during simulation.

Organization of program

The simulation program is organized as functional modules attached to a communications interface (see Figure 3.1). This organization provides for expansion of the simulation program to many computers, as required. The implemented functional modules may be overridden by the configuration specification in the configuration file. Doing so results in routing of the messages intended for those systems to respective ports on the gateway computer, reducing the gateway program to a link for those functions.

Each function therefore may be implemented as realistically as required. Controllers may be implemented as expert systems or may use sophisticated algorithms requiring the power of a separate computing unit, and executing in parallel (see Figure 3.2).

Data flow into the simulator is through the communication module or the operator interface. The operator interface is handled similar to the communication interface: real-time operation is emphasized. Operational messages received at the communications interface are assigned to queues based on the type of message. It is then the duty of each functional module to recognize and respond to the events as represented by the initiation messages. The scanning cycle of each unit is currently short, however, when sophisticated simulation modules of a function is desired (such as planning modules with optimization), then it may be lengthened and interfere with the real-time operation of the simulation.

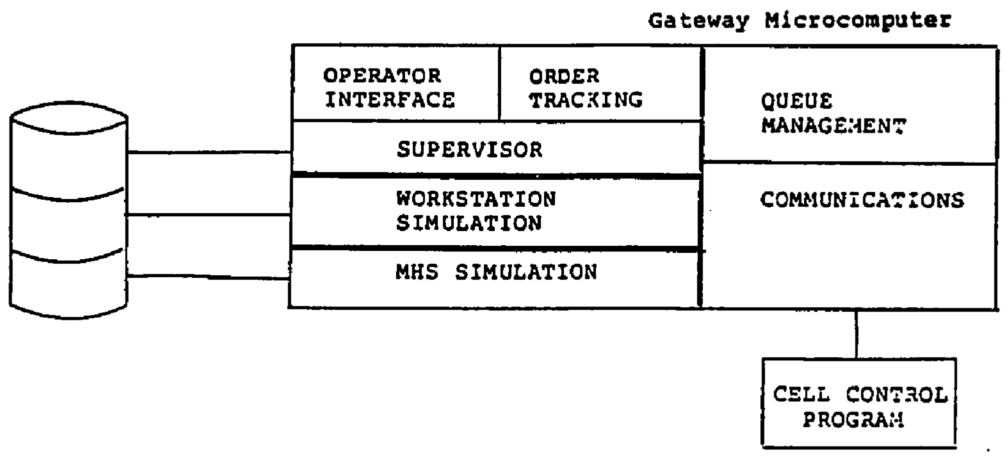


Figure 3.1: Gateway Simulator Organization

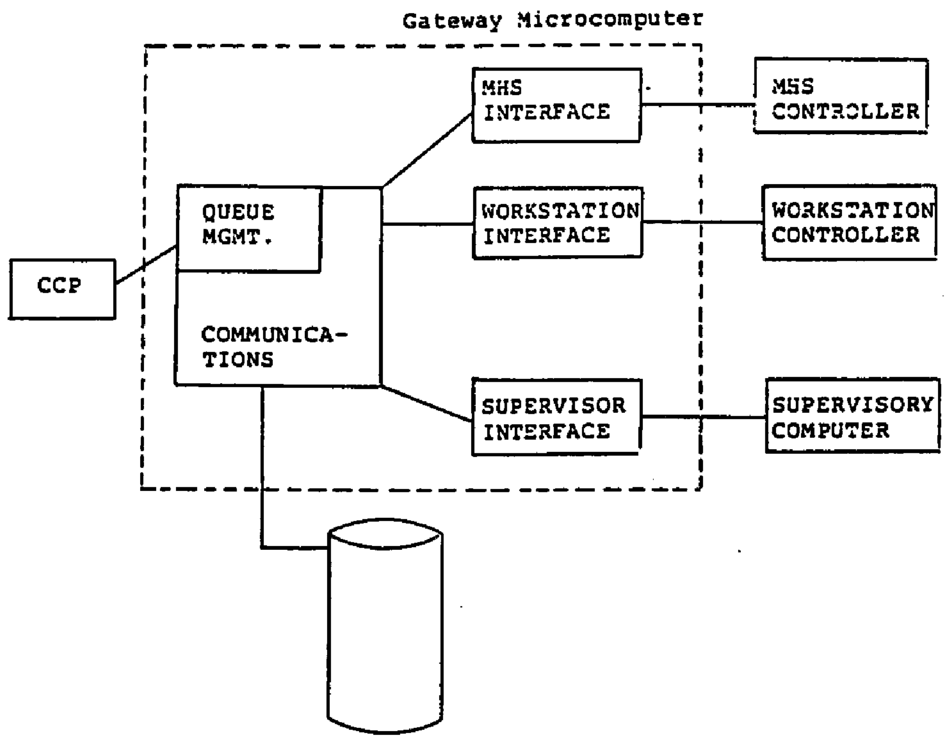


Figure 3.2: Expanded Simulation Model

Operator Interface

The simulation program has an operator menu which can be called by the operator by typing an "M" at the keyboard. Because of the real-time emphasis, menus are not presented all the time. Even when the operator menu shown in Figure 3.3 is presented, the simulation program keeps operating until the operator enters his request. At any point, the operator may ask for the status of the system, the contents of the queues, or any other statistic of interest. The operator can also send a message (either operational or database query) to the cell control program at any time by using the "Send Message" option of the operator menu. It is not very desirable however because some events (such as pending messages) may be delayed, since the user is so much slower to respond and assemble the message he wishes to send.

```
Time is:   2210

WHAT DO YOU WISH TO DO NEXT?

S. Send Message
H. Respond to MHS Messages
F. Respond to Fixturing Requests
P. Respond to Program Download Messages
W. Respond to Operation Start Messages
D. Database Update
R. Read incoming Message (if any)

Selection ==>
```

Figure 3.3: Operation Menu of Gateway program

One example of required operator interaction is the messages to the cell controller for manufacturing parts. Since the operator interface is used for simulation of the supervisory level controller's task allocation function, it is necessary that the interface be used at least in this capacity. Since this is not a frequent event within the time period of interest to the cell controller, it does not impede the operation of the simulation.

Simulation of Workstation Responses

The workstation simulation portion of the gateway to the cell controller simply responds to commands from the cell controller. The command to start operation is responded after an appropriate time interval representing the operation has elapsed. The operation times are deterministic however a random element representing the dulling of the cutting tools or similar events may be invoked if desired. Ongoing operations are assigned a completion time and the list of ongoing events is periodically scanned by the simulation program to determine if any of the event times have been reached. When one is found, a service completion message is sent to the cell controller for that workstation.

When the computer integrated manufacturing environment simulator is configured such that there is an attached computer acting as a workstation controller, the messages related to that workstation are routed to the appropriate port, and responses sent to the cell controller. The stand alone version of workstation controller simulator works in a similar way, however it actually controls operation of physical equipment, not merely a simulation of equipment. The programmable equipment physically implemented were instrumental in better understanding the needs of remotely controlling such devices with local intelligence, thus influencing the design of the workstation and cell level controllers. Information on the operation and programming of the workstation controller is given in a following chapter.

SUMMARY

A program to act as a gateway to the cell controller was designed. Elements of the computer integrated manufacturing environment that the cell controller routinely interacts with, such as the workstation controllers and material handling system controllers, are simulated by this program to enable operation of the cell controller.

If material handling system controller or workstation controllers are simulated on separate units, this program can be configured to coordinate the communications between computers. This feature makes it possible to distribute simulation of the system to several computers as required by detailed models.

CHAPTER 4

MANUFACTURING CELL CONTROLLER

This chapter introduces the highest level decision maker implemented in the context of this project, the Manufacturing Cell Controller. The design and implementation of the cell controller is described in the following sections, after a review of the functions of the manufacturing cell controller. Since the workstation controllers and the material handling system controller operate upon instructions from the cell controllers, understanding the operations of the cell controller is essential for clear interpretation of the workings of the manufacturing cell.

This chapter also describes the messaging system used by the cell controller in detail. Since these messages are received and responded to by the other controllers that are connected to the cell controller, an explanation of the messages and the messaging system at the end of this chapter may be used as a guide when the other controllers are being considered as well.

FUNCTIONS OF A MANUFACTURING CELL CONTROLLER

A. Task Decomposition. The cell controller must be able to efficiently reduce the higher level commands (requests) from the supervisor to lower level tasks. Lower level tasks must then be further decomposed to either direct commands to shop floor equipment, or to high level commands to subsystems (such as the material handling or tooling subsystems). Models for achieving this function have been constructed, generally using state tables or petri nets.

B. Monitoring and Control. The workstation must monitor all the equipment and subsystems within the domain of the cell, and take action when necessary.

   1. The monitoring functions can be grouped as process monitoring, equipment monitoring, handling monitoring, and quality monitoring.

      a) Equipment monitoring involves checking the performance of the equipment, such as machines or robots, to make sure that they are functioning properly. Included are the cleaning systems, safety systems, etc, which may report to the cell controller.

      b) Process monitoring involves monitoring of chip removal, tooling, and progress of the manufacturing processes. Tool wear or breakage, or other mishaps during the process will have to be detected through the monitoring of the process.

c) Material handling is monitored by tracking the execution of the commands given to the material handling system so that coordination of the operation of the cell with responses from material handling system can be achieved.

d) Quality monitoring involves the dimensional checks on workpieces as well as calibration of the equipment. Results of these may be used by the controller for maintenance planning.

2. Controls must be exercised by the cell controller on the equipment in the cell and on the associated subsystems such as to correct faulty operations or to achieve production goals.

a) Direct control involves evaluation of the monitoring data by the cell controller to take action. Sensory data is directly monitored by the cell controller.

b) Supervisory control involves coordination of the control functions of the lower level direct controllers.

C. Reporting is a function of the cell controller which covers a wide range of summary and statistical reports, as well as real-time status reports. These reports are used for inventory control, production scheduling, maintenance planning, and managerial functions.

D. Communications is an essential function of the cell controller. The cell controller must be connected to the network and should be able to communicate with supervisory level computers, as well as lower level equipment controllers.

IMPLEMENTATION OF THE MANUFACTURING CELL CONTROLLER

The asynchronous nature of the events in the cell necessitates a flexible implementation of the functions defined in the above section. A simple but effective organization, consisting of modular functional units attached to a messaging system, was implemented (see Figure 4.1). This organization, in addition to being capable of responding to asynchronous events, is also advantageous because of the ease of maintenance and improvement of the program. Evolutionary changes, such as substitution of a complex algorithm for workstation loading, can be made easily by replacing functional modules.

It is assumed that the functions implemented in the cell control program will be replaced by better ones in time, leading to a series of intelligent decision-makers that control functions of the cell controller in the future. Listing of the Cell control system programs and programmer's manual are given in [24]. Major flow charts for the cell controller and simulator are given in the appendix.

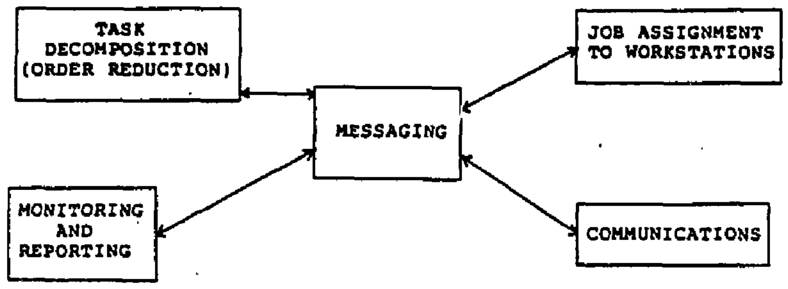

TASK DECOMPOSITION (ORDER REDUCTION)
JOB ASSIGNMENT TO WORKSTATIONS
MESSAGING
MONITORING AND REPORTING
COMMUNICATIONS

Figure 4.1: Cell Controller Organization

The cell controller is implemented as the Cell Controller Program (CCP) written in Prolog language. Prolog was chosen for the ease of programming in the intended modular fashion and without the procedural definition of each detail. A discussion of the suitability of Prolog to this type of problem is given by Bullers [6].

CELL CONTROL PROGRAM

The Cell Control Program (CCP) implements the functions of a manufacturing cell controller. Decisions for assignment of jobs to machining centers and control of activities within the cell are made by this program. The status of the workstations in the cell and that of the manufacturing orders are displayed in real-time.

Briefly stated, this program receives orders to be fulfilled from the supervisory levels, decomposes the orders to tasks, and monitors the execution of these tasks by workstations in the cell. The coordination of activities in the workstations and subsystems such as Material Handling are also made by this program. CCP is written in Prolog, and is a modular program.

CCP is a data driven program. All the functions implemented work on the existing databases and apply transformations to it. The link between functions are the changes in the databases; algorithms remain the same irrespective of the orders or configuration.

Databases Maintained by the CCP

Databases maintained by the cell controller cover a wide range of information: cell configuration, equipment status, production status, product information, and operational data (see Table 4.1). These databases may be queried by supervisory level controllers, or subsystems such as MHS and tooling. Databases may also be updated by the supervisory level controllers through the network.

Cell configuration information include the types of workstations in the cell, designations of individual workstations, and information necessary to make dispatching decisions such as input output port types, buffers, etc.

Equipment status data include the starting times of workstations and the tasks on which they are working. Status codes are given in Table 4.2. Error conditions, the time since the last failure and statistical information are also kept on workstations.

Product information includes the process plans for the parts currently assigned to the cell, as well as the types of the workcells required, fixturing, tooling, material handling information, NC program designations etc. This data are normally kept by the central system and dispatched to the cell controllers as jobs are allocated. NC programs may be kept by the central system and downloaded directly to the workstations upon request by the cell controller.

Production status data and operational data are the piece counts, message queues (in and out), waiting lists for equipment, material handling etc. are also used by the control program.

Table 4.1: Databases Required for FMS Control

Order Id, Part_id, Quantity, Due_time

Part id, Material, Process_plan_#

Process plan #, Operation no, Machine_typ, Tool, Fixture, Program_length.

Workst id, Machine_typ

Workst id, Status, Order_id, Part_id, Operation_no, Start_time

Table 4.2 Status Codes for Workstations

0 -	Available (idle)
1 -	Allocated
20 -	Program Download Complete
50 -	Workpiece transport completed
100 -	Operation Initiated
150 -	Maintenance (scheduled)
180 -	Error Condition Pending

Task Decomposition

The task decomposition function is based on a process driven by birth and death of tasks. A task is an entry in a "task table". Tasks are classified in three levels: master, intermediate, and terminal. Tasks give "birth" to a number of subtasks at a lower level (see Figure 4.2). Terminal tasks "die" upon realization of certain conditions in the system status, triggered by events in the system such as completion of a machining operation in a workstation. Tasks die when all the subtasks of the task are dead.

Each order for manufacturing a part (or a group of parts) by the cell creates a master task, which in turn creates subtasks. Each of the subtasks also create subtasks to cover every step of the work to be done for completion of the order.

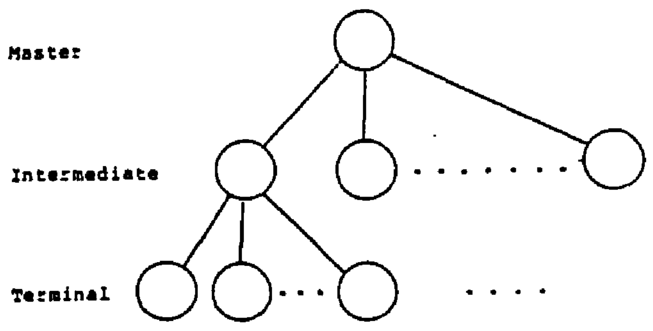


Figure 4.2: Task Decomposition

Scheduling of the jobs to the workcenters is currently based on a few simple rules: a workcenter must be available; a task which is waiting for the type of workcenter must be eligible (i.e. satisfy precedence relationships). Sophisticated priority schemes may be added to the Cell Control Program without difficulty.

Monitoring and Control

Monitoring and control functions require prompt handling of asynchronous events in the cell, which is difficult to implement efficiently in a non-multitasking environment. To achieve good performance and flexible operation, all I/O was implemented through queues, and functions of the program were tied to these queues instead of real I/O, therefore relieving the network connection which otherwise would be a bottleneck.

Interfacing of the CCP to the central network is made through I/O queues. Messages are inserted to these queues by various functions of the CCP. Portions of the program scan incoming messages and remove messages that are addressed to them. This arrangement provides for flexibility in the handling of input output and the monitoring functions. Protocols used by the messaging system are strictly interlocked to ensure correct transmission.

Also implemented is the real-time display of the status of the workstations in the cell, and reporting of the part completion, order completion, etc, to the supervisory and subordinate level controllers.

Operation of CCP

Functions implemented in CCP are periodically scanned and executed in sequence. Most functions, however, require certain input conditions to operate and therefore decline to operate in the absence of them. This provides timely scanning of all functions, as well as monitoring of asynchronous events.

1. The order dispatching function monitors the incoming message queue for new jobs. When a new job is released, a master task is created identifying the order. The master task gives birth to the required number of subtasks: for each part to be produced for the order, a subtask is created (called an intermediate task). A message to the Material Handling System signals the creation of each intermediate task (hence a workpiece). Each subtask then produces terminal tasks, each terminal task identifying an operation to be performed on one part. Terminal tasks place a request for the types of workstations required for performing their respective operations: a waiting queue is formed.

Upon receipt of the message signalling the release of the part, and depending on the number of fixtures available for the workpiece, MHS readies workpieces for transportation. Each part is fixtured as required. When a workpiece is ready for transportation, a message is sent to the cell controller and the associated tasks are marked as active (fixtured).

2. Waiting queues are periodically scanned for matches in waiting tasks and idle workstations. When a match is made: a workstation is available and a task is waiting for the type of workstation, the precedence constraints of the task is checked. If all the task preceding the candidate have been completed, the workstation is allocated to the task. Otherwise, other candidates are considered. If the workstation is allocated to the task, the task is dropped from the waiting queue, and placed in waiting completion queue. The status record of the allocated workstation is updated and messages are sent to the network for dispatch of the workpiece to the workstation. Instructions for downloading the program required for processing the workpiece at the workstation are also sent to the network.

3. Message queues are continuously checked for messages to be sent or received. When an incoming message is detected (currently a request on the port), the incoming message is received and placed in the incoming messages queue. When the presence of outgoing messages in the queue is detected, the status of the transmitter is interrogated. If the transmitter is available, the message is removed from the "outgoing messages queue" and sent.

4. The incoming message queue is continuously checked for messages from the workstations and from subsystems. When an "end of processing" message is received, the waiting completion queue is updated, and the task operating on the workstation is completed ("dies"). The parent process of the completed task is also checked for completion. If the completed process is the last child of the parent process, it dies too, and the link proceeds all the way to the fulfillment of the order (i.e., death of the master task) by the completion of the last operation on the last part in the order. When completion of a task means a part is completed, the "end of task" message is passed to the material handling system, so that the part can be dispatched to the warehouse as a completed part. This also signals the MHS that the fixture is now available for another workpiece, and will be used in case any requests are pending for that type of fixture. Messages from the network ("end of download"), and from the Material Handling system ("end of transport") are used for updating status information and for starting the processing at the workstation when the workpiece is transported, and the NC programs are downloaded.

5. Scheduled maintenance on workstations is performed at the discretion of the "maintenance planning unit", which instructs the workstation controller not to schedule a workstation past a certain time. When the workstation becomes available, the scheduled maintenance event prevents

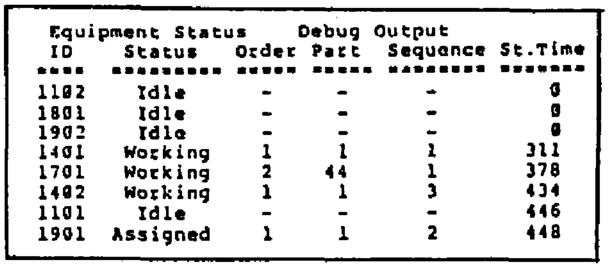
it from being reassigned to another task. The workstation then shuts down for maintenance. Upon completion of maintenance, the cell controller receives a "workstation available" instruction, and the workstation is added to the pool of available equipment for allocation of tasks.

6. When a workstation has to shut down because of an error in operation, the cell controller is informed. If the workstation has an allocated task, it is reinstated in the waiting tasks pool. The MHS is instructed to remove the workpiece, and until a "error condition removed" message is received, the workstation is not assigned to any other task.

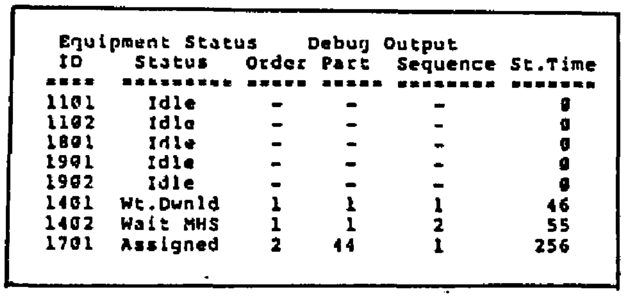
7. At the end of each scanning cycle (i.e. when all the functions have executed), the status of workstations as contained in the databases is displayed. It is therefore possible to see the assignment of jobs to workstations and progress of orders in the cell. (see Figure 4.3).

The Material Handling System (MHS) works as a subsystem of cell controllers.The cell controller places requests to the MHS via messages through the network. Responses of the MHS are passed back to the cell controller the same way. Movement of the workpieces between workstations and buffers is executed by this subsystem. MHS is also expected to manage the workpieces in process (i.e. unfinished parts in the shop floor) by making the decision either to move them to buffer storages or to keep them in the I/O buffers at the workstations. The operation of the MHS and its specific implementation used in this project are detailed in other chapters.

The Tooling System works the same way as the Material Handling System: as a subsystem of the cell controllers. It has not been implemented in the current version but may be easily attached to the system if desired. Control of this subsystem will be similar to that of the material handling subsystem: via messages through the network.



```
 Equipment Status    Debug Output
  ID    Status    Order Part  Sequence St.Time
 ====  ========== ===== ===== ======== =======
 1102    Idle       -     -      -          0
 1801    Idle       -     -      -          0
 1902    Idle       -     -      -          0
 1401   Working     1     1      1        311
 1701   Working     2    44      1        378
 1402   Working     1     1      3        434
 1101    Idle       -     -      -        446
 1901  Assigned     1     1      2        448
```



```
 Equipment Status    Debug Output
  ID    Status    Order Part  Sequence St.Time
 ====  ========== ===== ===== ======== =======
 1101    Idle       -     -      -          0
 1102    Idle       -     -      -          0
 1801    Idle       -     -      -          0
 1901    Idle       -     -      -          0
 1902    Idle       -     -      -          0
 1401  Wt.Dwnld     1     1      1         46
 1402  Wait MHS     1     1      2         55
 1701  Assigned     2    44      1        256
```

Figure 4.3: Cell Control Program Display of Status

Messaging System

Messages are passed to and from the subsystems through a network connected to all the workstations on the shop floor as well as to the cell controllers and supervisory level controllers. Downloading of the processing programs to the workstations is made directly from the network-wide program storage area, upon request by the cell controller. In the absence of the actual network, the messages were passed through a serial link to the simulated "network gateway computer". The responses of the network are also simulated by this computer.

Since the serial link is run asynchronously, and operating systems of all computers do not provide an adequate input/output buffer, a fully interlocked protocol was used to pass error-free messages between computers. It calls for acknowledgement of every byte received (see Figure 4.4), as well as leader and follower byte handshakes. Tables 4.3 and 4.4 show the protocols used.

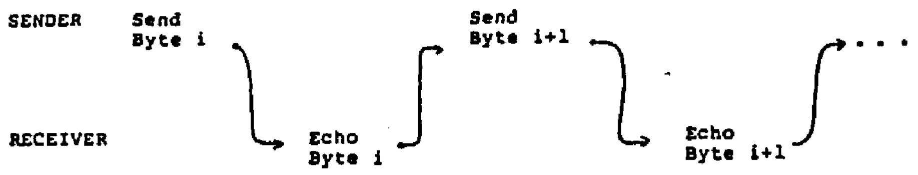


Figure 4.4: Fully Interlocked Protocol

Two types of messages are handled by the cell controller messaging system: operational and database related. There are two types of database related messages: database updates and database queries.

Operational messages are those involving commands to and from subsystems: status requests and diagnostics (see Table 4.5). Database update messages are those that involve insertion of new information to the cell controller databases, or deletion of a record from them (see Table 4.6). Database queries are the messages sent to the cell controller for downloading of certain information in the databases.

Two formats (see Tables 4.7 and 4.8) are used for messages: fixed and variable. Operational messages use the fixed format. Database related messages use both variable and fixed formats. The database update messages use the variable format and contain all fields of the record being updated (only the key fields are used for deleting). Database queries are received in the fixed format and are queued, but the result (most of the time more than two fields), is sent using variable length format.

The three types of messages are handled differently by the messaging system:

Operational messages are queued in and out of the cell controller. Actual processing of incoming messages is performed through the queue by functional modules. The format of these messages is fixed length (see Table 4.9).

Database queries are received in the fixed format and are queued as such. However, when the message is processed, the database is queried using the key in the message, and the result (most of the time more than two fields) is sent in a variable length form distinguished by the request signal.

Certain types of messages are not queued, but are interpreted at once. These types of messages use a variable length message format (see Table 4.8), where the length of the database record updated and the kind of update determines the message length (see Table 4.10). For example, inserting a database record with seven fields requires all fields to be sent, whereas deleting a similar record only requires the key field.

Table 4.3: The Messaging Protocol For Operational Messages

Initiating Party	Receiving Party
Request (a 7 or 15)	Acknowledge ("A")
Hi-byte of opcode	Echo hi-byte
Lo-byte of opcode	Echo lo byte
Hi-byte of first operand	Echo hi-byte
Lo-byte of first operand	Echo lo-byte
Hi-byte of second operand	Echo hi-byte
Lo-byte of second operand	Echo lo-byte
End-of message (a 7 or 15)	Acknowledge ("I")

Table 4.4: Messaging Protocol Of Variable Length Messages

Initiating Party	Receiving Party
Request byte	Acknowledge ("A")
Hi-byte of database code	Echo hi-byte
Lo-byte of database code	Echo lo byte
Hi-byte of update type	Echo hi-byte
Lo-byte of update type	Echo lo-byte
Hi-byte of operand	Echo hi-byte
Lo-byte of operand	Echo lo-byte
(repeated for each operand)	
End-of message	Acknowledge ("I")

NOTES:
Request byte is: a 17 for updates,
25 for query responses.
End-of message byte is 17 for updates,
15 for query responses.

Table 4.5. Operational Messages

From Cell Controller:	**Destination**
Start processing at workstation	Workstation
Download program to equipment	Network
Transport to workstation	MHS
Part complete (transport part to warehouse)	MHS
Raw material in store, fixture	MHS
Operation complete (remove workpiece)	MHS
Remove workpiece (Error in workstation)	MHS
Order completed	Network
To Cell Controller	**Origin**
Processing completed	Workstation
Program download completed	Network
Workpiece transported to workstation	MHS
Workpiece fixturing complete	MHS
Process order (order dispatched to cell)	Network
Scheduled maintenance at workstation	Network
End of maintenance	Network
Irrecoverable error condition	Workstation
End of error condition	Workstation

Table 4.6: Database Update Messages

Insert new order record	or	Delete old order record
Insert new part information	or	Delete old part information
Insert new process plan	or	Delete old process plan
Insert new equipment	or	Delete old equipment

Table 4.7: Messaging Format For Fixed Length Messages

Opcode (2 bytes)

Operand one (2 bytes)

Operand two (2 bytes)

Table 4.8: Messaging Format For Database Update Messages

Database code (2 bytes)

Update type (2 bytes)

Operands (2 bytes each operand)

(Number of operands is determined by the database code and the update type)

Table 4.9: Opcodes and operands of operational messages

Opcode	Message	Operand-1	Operand-2
10	Start processing	Workstation-id	-
11	Completion of processing	Workstation-id	-
20	Download program	Workstation-id	Program-id
21	Download complete	Workstation-id	Program-id
30	Process order	Order Number	-
31	Order completed	Order number	-
50	Transport workpiece	Piece-no	Workstation-id
51	Workpiece transported	Piece-no	Workstation-id
52	Part completed	Piece-no	-
54	Fixture raw material	Piece-no	Material no
55	Fixturing completed	Piece-no	
56	Operation completed	Piece-no	Workstation-id
58	Operation failed	Piece-no	Workstation-id
70	Scheduled maintenance	Workstation-id	Time
72	End of maintenance	Workstation-id	-
80	Error condition	Workstation-id	-
82	End of error condition	Workstation-id	-

Table 4.10: Database Message Information

Database codes for database update messages

Database Code	Update Type	Description
1	1	Insert new order record
1	2	Delete old order record
2	1	Insert new part information
2	2	Delete old part information
3	1	Insert new process plan
3	2	Delete old process plan
4	1	Insert new process plan step
4	2	Delete old process plan step
5	1	Insert new equipment
5	2	Delete old equipment
6	1	Insert new equipment status record
6	2	Delete old equipment status record

Database Codes Used by Query and Update Functions

Database Code/Name	No.of Fields	No.of Keys
1. Orders	4	1
2. Parts	3	1
3. Process Plans	2	1
4. Process Steps	7	2
5. Equipment	3	1
6. Equipment Status	4	1

Opcodes and Operands of Database Query Messages

Opcode	Operand1	Operand2
400+(database code)	Key1	Key2
500+(database code)	<all fields in database record>	

Note: Requests have codes starting with 400
Responses have codes starting with 500

SUMMARY

A manufacturing cell controller was designed and implemented on a microcomputer. The cell controller is data driven, performs independent of the cell configuration, and exploit databases that exist in the computer integrated manufacturing environment. It is an asynchronous program that has a modular structure, and therefore can be easily extended to include sophisticated algorithms for planning and control functions. The cell controller runs on a stand-alone microcomputer, and can be controlled by means of messages through the connection to the factory-wide network.

CHAPTER 5

MACHINING WORKSTATION

A machining workstation consists of a primary programmable machine, equipment to load and unload workpieces to it, and auxiliary equipment such as vision, chip removal, etc. The primary programmable equipment is typically a numerically controlled machine tool, such as a mill or lathe, or an inspection machine. To load and unload the machine tool, it may be necessary to use other programmable equipment such as robotic arms or machine-specific loaders. Manipulator arms or robots may be used in the workstation for loading and unloading the machines, or for inspection and cleaning purposes.

Each workstation must have an interface to the automated material handling system which is used to transport workpieces to and from it. See Figure 5.1. This interface, such as a roller table, is under the control of material handling system. However, the cell controller must be capable of identifying the workpiece at the interface, or at least to verify the presence of it through the sensors in the workcell. The orientation of the workpiece, if necessary, can be verified using the optional equipment such as vision and tactile sensors.

Each workstation is controlled by a computer connected to the factory-wide network: the Workstation controller.

WORKSTATION CONTROLLER

A workstation controller is in charge of operations within the domain of a workstation. Commands controlling operation of the programmable equipment in the workstation are issued by the workstation controller in accordance with the workstation program loaded by the cell controller.

The workstation controller receives instructions from the cell controller to work on a part, and the program required for the operation is downloaded to the controller. The workstation program contains instructions for all available units in the workstation as required for the assigned work. It may contain instructions for the manipulator to load the machine, instructions for the automatic tool changer to set the proper tool, or for the vision system to inquire the orientation of the workpiece.

A workstation may be programmed on-line or off-line. Off-line programming may be done using computer graphics or a programming language designed for programming workstations. On-line programming can be done using follow through sequence and manual control. Several programmable systems for workstation control have been developed [2,3,8,10].

A workstation must be able to run calibration and diagnostic programs as required by the maintenance supervisor, or by the cell controller. Results of such activities are reported to the supervisor. Expert systems are being implemented for problem diagnosis in workstations.

Implementation

In the context of this project, a simulation of responses of a workstation controller is made by the gateway simulation program (see Chapter 2). However, when the manufacturing cell model is expanded and run on several microcomputers, a limited simulation of responses of machining workstations may not be satisfactory. It may also be desirable to construct several workstations and run them under control of the cell controller. A machining workstation controller was designed and implemented so that it can be duplicated and used for as many workstations as are in the cell. This implementation is not only intended to be instrumental in demonstrating the operation of a workstation controller, but also to enable the cell controller simulation to take place by responding to its requests.

Two programmable devices were physically implemented as a component of this research. They were chosen as being representative of a wide variety of programmable equipment controllers (or interfaces to them). They were instrumental in determining the operational requirements of the workstation controller. The types of control required by these devices, and the types of feedback available to the controller were considered in the design of the workstation controller. It was observed that the capability of these devices to handle exception processing reduces the real-time demand on the workstation controller. On the other hand, it was determined that availability of a richer and consequently larger feedback data stream may strain the processing and the communications network. This places additional burden on the planning on what data transfer is to occur and the communications protocols used between devices.

The current implementation of the machining workstation controller is not sophisticated, but provides the foundations for further development. Since its design is not tailored to specific equipment, but assumes presence of an interfacing computer (the device interface), it can be duplicated to form several workstations (see Figure 5.2). Due to the flexible design of the workstation controller, the configuration of the workstation is flexible, consisting of any number of similar or dissimilar types and models of equipment. The configuration of each workstation is specified in the configuration file which is local to the workstation and provide all the necessary information on I/O ports and available equipment types.

The workstation controller can be programmed with a simple programming language provided. The basic idea of programming the workstation is to provide the messages to the programmable equipment to activate, deactivate, perform diagnostics, load programs, read values from sensors as required. The programming language provides simple ways of controlling flow of the program in the workstation controller. Detailed explanation of the instruction set is given in the following sections. Each workstation program also contains the required equipment types and additional information as necessary.

Binding of the equipment in the workstation (as read-in from the configuration file) to the equipment requested by the workstation control program (as read-in with the program) takes place after the program is loaded. The workstation program may or may not address any of the equipment available in the workstation. For example, if there is a vision sensor available, it may not be utilized in the processing of the particular part (see Figure 5.3). However, all the requested equipment must be present in the workstation. Absence of any required items would result in an error condition upon loading of the workstation program.

The workstation physically constructed for this project, and using the workstation controller program consists of a workstation controller computer to which a programmable machine tool and a programmable manipulator arm are attached.

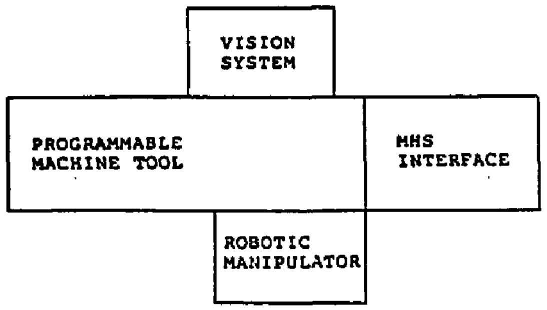


Figure 5.1: Machining Workstation

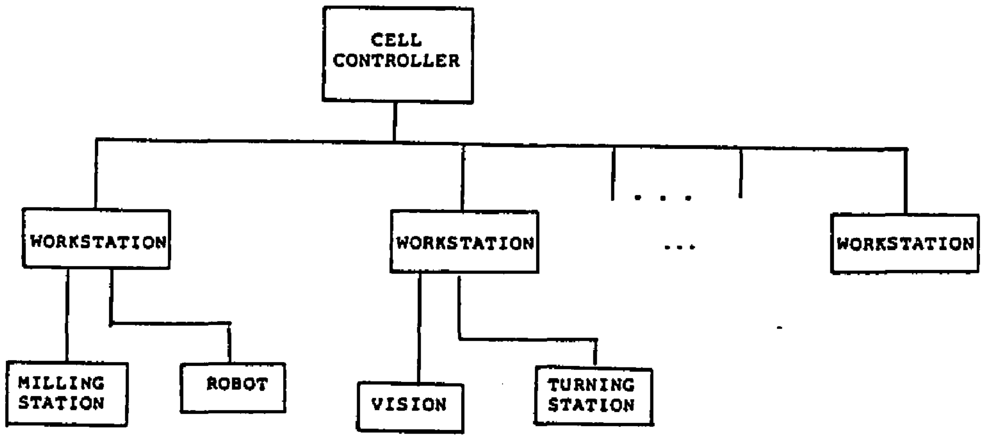


Figure 5.2: Multiple Workstations with Dissimilar Configurations

CONFIGURATION DATA

ID	TYPE	QUANTITY
A	1	1
B	5	1
C,D	6	2

PROGRAM HEADER

NO	TYPE	QUANTITY
1	6	1
2	1	1

AFTER BINDING

NO	ID
1	A
2	C

Figure 5.3: Binding of Machines in Workstation

Operation

The workstation controller receives instructions from the cell controller. Messages received at the cell controller connection are interpreted by the workstation controller and action is taken accordingly. When the cell controller requests a program to be downloaded, disk read operations are substituted instead of requesting the download from the factory-wide network. This is a reasonable substitution in the absence of the global high speed network to access factory-wide manufacturing database. The workstation controller is programmed in a simple way, and programming includes the names of the files containing programs to be downloaded to the attached programmable equipment. Those file names are passed to the controllers of the programmable equipment, where the actual NC programs are stored locally, and are accessed by the programmable equipment directly instead of being downloaded. All programmable equipment programs require an action from the workstation controller to start execution.

Upon receipt of the acknowledgement of program loading, the workstation controller waits for the cell controller to start operation. When the "Start Operation" command from the cell controller is received, signals are dispatched to the programmable equipment to continue execution of the loaded programs. No further interaction is necessary at this point. However, the workstation control program may contain several breakpoints for the equipment at specific times, and may test for status of the equipment by reading values from their registers. Since these are equipment specific operations, programming of each item of equipment unit would have to be done for its particular equipment type. The type of workstation system, however, is not of concern to the workstation controller whose operation is independent of the equipment. The workstation controller simply waits for end-of-operation signals from the programmable equipment, and after all of the equipment units has responded, it reports completion of operation to the cell controller.

Programming

The workstation is programmed using the simple programming language provided. Since the workstation controller carries out all functions by sending and receiving messages to the attached equipment, the programming is based on messages. Simple program flow control structures using internal variables are also provided for use by the programmer.

Workstation programs consist of the header and the body of the program. The program header contains the number of required equipment units, and the type of each equipment required by the program (see Figure 5.3). This information is used by the workstation controller in binding the requested equipment to those available in the workstation (as mentioned in the previous sections).

The body of the workstation program consists of an opcode and up to five operands per instruction. The opcodes implemented are given in Table 5.1. Instructions implemented are explained below:

The "execute" (or send) instruction causes operands two, three and four to be transmitted to the equipment specified in operand one. The transmitted message is interpreted as a command by the equipment receiving it. The workstation controller, however, does not interpret it. All instructions with "unconditional send" opcodes are transmitted as soon as the program starts executing. The fifth operand of all instructions has the number of the instruction to be executed next. If the fifth operand of a send instruction is not given, it indicates that the instruction following it contains the needed information.

The conditional execution instructions are not executed (i.e. their messages transmitted) until a next-to-execute operand points to them. In contrast, all unconditional execute instructions are transmitted as soon as the program starts executing. This facilitates initializing and starting all equipment at once and then waiting for the results of each to continue program.

The "wait" instruction results in the placement of an expected message from a certain equipment to the wait queue. Until the message is received, no action is taken. All the incoming messages are tested against the expected messages. When a match is made, the next-to-execute instruction (operand 5) of the wait instruction is executed. This enables continuous checking for certain error and completion messages to be stacked and the responses programmed as conditionally executed set of statements.

To test for the completion codes from equipment and to be able to read and transmit values to and from equipment, variables are implemented. A total of fifty variables are permitted in each program. Variables may assume values from the messages received from the connected equipment, and may be used within the messages to be transmitted to others. If a value has not been assigned, variables have a value of zero.

Perhaps the most extensive use of variables is made in the test and branch instructions. These instructions (opcode 110) test for the condition (specified in operand 2) of the variable whose number is given in operand one to the value given in operand 3. If the condition holds, the next-to-execute instruction is contained in operand five. If not, the next instruction to execute is contained in operand four. Table 5.2 shows the workstation program instructions and their operands.

The format of each instruction is an array of six integer elements per instruction, where the first element is the opcode and the others are operands. Variables are also implemented as integers. Values -9900 to -9949 represent variables 1 to 50 in the program.

The reasons for this structure are the simplicity and ease of implementation. The fact that these programs are expected to be generated by programs at CAD/CAM systems means that they need not necessarily be intelligible to humans. If manual programming or manipulation of programs is required, an interpreting editor may be used. Currently, any ASCII editor may be used to edit programs.

Table 5.1: Opcodes of Workstation Programming Instructions

Opcode	Meaning
0	Execute (send)
100	Conditional execute (send)
1	Wait for response
110	Test & branch

Table 5.2: Workstation Program Instruction Structures

Instruction: Execute(0) or Conditional Execute(100)
Operands:
1. Equipment number
2. Command (first two bytes)
3. Command (next two bytes)
4. Command (next two bytes)
5. Number of the next instruction to execute (default is following instruction)

Instruction: Wait for response (1)
Operands:
1. Equipment number
2. Response (first two bytes)
3. Response (next two bytes)
4. Response (next two bytes)
5. Number of the next instruction to execute upon receipt of response (default is following instruction)

Instruction: Wait for response (1)
Operands:
1. Variable number (whose value will be compared)
2. Condition (-1,0,+1 for <,=,> respectively)
3. Value to compare (maybe another variable
4. Next-instruction-to execute if condition fails (default = none)
5. Number of the next instruction to execute upon receipt of response (default is following instruction)

PROGRAMMABLE EQUIPMENT

Equipment used in the workstation in this study were a programmable logic controller (PLC) and a programmable manipulator arm. Operation of these units are described in their respective manuals, and the interfaces to the system are described in the following sections. The PLC represents a programmable machine tool, an inspection machine, or any other generic programmable equipment with remote control and communications capability. The programmable manipulator arm is representative of a generic computerized system that may also represent any one of the equipment mentioned above. It consists of a point to point robotic arm interfaced to a microcomputer that acts as the controller and provides local or remote programming capability, operator interface, and interfaces with several sensors that may be used in programs or separately to monitor and coordinate with external events.

The manipulator arm and the device interface for the PLC were implemented physically rather than simulated in software to better understand the requirements of remotely controlling programmable devices with local intelligence. The types of control signals, which are used in controlling such devices were observed along with the urgency and frequency of the messages. The high degree of flexibility built into the design of the workstation controller was based on these observations and used as the basis for interaction with intelligent devices.

Manipulator Arm

Interfacing the robotic arm was accomplished by designing a card to fit the internal slot of the microcomputer. The interface card contained the required hardware to buffer the outputs to the arm and inputs from the external sensors. Signals to the stepper motors on the manipulator were output to the interface and multiplexed by the interface card on the manipulator. The manipulator can be manually moved to the desired position by using the keyboard as a teach pendant, and the position recorded as a step in the program(see Figure 5.4). It can also be programmed using the simple programming language provided and the programs can be edited, single stepped, and stored. It is possible to program the robot manupilator using positions of external switches or the position of the manipulator for branching, thereby creating reactive motion.

The manipulator can also be programmed to send or receive signals on the serial port of the computer. One option relinquishes all control to the serial port, thereby enabling remote programming and control of the manipulator arm. This capability was utilized by the workstation controller. See Figure 5.5.

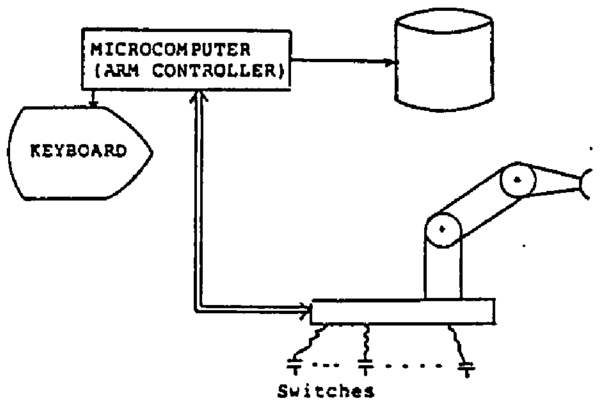


Figure 5.4: Manipulator Arm Interfacing to the Microcomputer

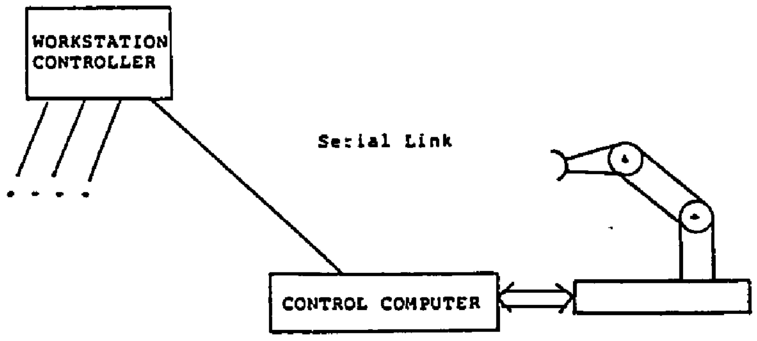


Figure 5.5: Remote Operation of the Manipulator Arm

Operation

Robot5 was designed as an interactive, menu driven system for controlling the Minimover-5 Microbot robotic arm. Six stepper motors control movements of parts of the arm in both directions along three axes of motion. Information on the structure and construction of the hardware is given in the manual [22] and will not be replicated here.

Underlying the whole system are the routines which send the required pulses to activate the stepper motors, thereby causing the robotic arm movements. A simple algorithm is used to move from one point to another in a straight line. There are no transducers to sense the velocity, position or acceleration of the arm. The programmer must be aware of the restrictions in using the program. A switch on the arm senses the tension of the cable activating the gripper and can be read through the software to signal closure of the grip. Several other binary (two position) switches can also be read through the software.

The main menu presented in Figure 5.6 shows the options available to the user. The user is returned to this menu after each function selected is completed. Note that an indication of the program status is displayed at the bottom of the screen.

Upon selection of an item from this menu, the user is given either a menu (example, Edit/Modify Current Program) or asked to verify a request (example, Erase the Program in Memory). This feature provides protection of the program from faulty keystrokes.

Manual control of the Microbot is done using the keyboard and can be conveniently used to set the initial position of the arm (See Figure 5.7). Note that the Home key on this operation will set the step counters of all motors to zero, thereby declaring the current position "Home" (It is also highly recommended that instruction 22 be used to do this at the beginning of each program).

The speed of the motors can be set using the arrow keys, and will result in display of the delay value. The delay value represents the time interval between two consecutive pulses sent to a motor. Therefore, a large delay value will result in slow movement and a smaller delay value will increase the speed of movements of the arm. The user may experiment using different speeds to determine the best delay value for his application. Note that small delay values will result in higher speed, while causing the stepper motors to start slipping. Control of the position versus step values will not be possible under those conditions. Large delay values may result in unacceptably slow speeds.

```
            MICROBOT CONTROL MENU (Ver 5.1)
            ===============================

       SELECT ONE OF THE FOLLOWING:

             X  ... EXIT
             ?  ... Help

             1 ... Manually control Microbot
             2 ... Program Microbot (or teach)
             3 ... Retrieve Program from disk
             4 ... Execute program in memory (ONCE)
             5 ... Execute program in memory (repeat)
             6 ... Save program to disk
             7 ... Edit/Modify/List current program
             8 ... Erase the program in memory
             9 ... Modify/Change Registers
             C ... Continue Execution
             R ... Remote Operation

          *** NO PROGRAM IN MEMORY  ***
```

Figure 5.6: Manipulator Arm Control Program Main Menu

```
            KEYBOARD CONTROL OF MICROBOT
            ============================

       USE THE FOLLOWING KEYS TO CONTROL MICROBOT MANUALLY:

          FORWARD     REVERSE    MOTOR NUMBER
          -------     -------    ------------
             1           Q       BASE
             2           W       SHOULDER
             3           E       ELBOW
             4           R       RIGHT WRIST
             5           T       LEFT WRIST
             6           Y       GRIPPER
             7                   CLOSE GRIPPER (GRAB)

                       HOME      TO ZERO ALL COUNTERS
                   <UP ARROW>    TO SPEED UP
                 <DOWN ARROW>    TO SLOW DOWN
                           #     TO QUIT

        DELAY: 136
```

Figure 5.7: Manual Control of Microbot

The speed of manual control and program execution will may be the same even if the delay values are set the same (the difference may however be negligible). The user may prefer to experiment and get an intuitive feeling of this difference.

Option two, Program Microbot, cannot be used when there is a program stored in memory. If starting over is desired, the program in memory should be erased (using option eight), and then programmed. It is possible to add to an existing program using Option seven (Edit/Modify/List Current Program) conveniently.

An option to Edit is given upon exit from Option two anyway, since the first thing after programming would naturally be to list and see the program.

Programming the robot involves entry of the op-code for each instruction, upon which the instruction's function is presented (so that entry of the wrong op-code can be immediately recognized by the user). The program monitor then prompts the user for each operand required (or optional) for the instruction. The ranges of valid entries are given for most operands and entries are checked for validity. When a wrong instruction is entered, the recommended way to correct this is to exit (op-code 99), edit the program, delete the last instruction, and get back to the Program Editor using the ADD (A) option on the Edit menu.

Retrieve a program from disk (Option 3) will issue a warning if a program is already stored in the memory. If the user verifies the request by entering a file name (in which a program was previously stored), it will be retrieved and overwrite the current program. Registers will also be set to zero.

Three options are available for executing a program. Option 4 will execute the stored program starting with the first instruction and will return to main menu when a "Stop Execution" instruction (op-code zero) is encountered. Option 5 will start execution with the first instruction too, but will continue with the first instruction when a stop is encountered, therefore continuing the execution of the program indefinitely.

Execution of the program can be interrupted at any time by depressing a key on the keyboard, however the current instruction will be completed before the interruption. One exception is any instruction that requires a switch status change for completion. Any of these instructions will be completed as soon as any key is struck. This feature will prevent the program from waiting indefinitely for a nonexistent switch (This feature also applies to the close-gripper instruction).

If starting execution is desired starting with an instruction other than the first one, Option "C" (continue execution) must be used. This option will present the number of the last executed instruction (if any) and will ask the number of the next instruction to start execution. This feature may be used for debugging portions of the program stored or for returning to the execution of the program after inspecting contents of registers.

The Edit (option seven) presents the menu shown in Figure 5.8. This menu is designed for inspection and modification of the stored program. Listing of a program is shown in Figure 5.9. It must be used with caution, however since the "Jump" instructions will not be changed to compensate for the deleted and inserted instructions. A good programming practice is to leave "No operation" (NOP, op-code=1) instructions between most instructions so that additions or deletions can be made without difficulty. Example: Instead of deleting an instruction and adjusting all Jump instructions that are affected, you may replace the instruction with a "NOP". Another example would be to insert an instruction without having to readjust the jump instructions, you may replace an existing NOP with the desired instruction, without affecting any other instruction.

All registers (explained in the following paragraphs) can be read and changed using option nine.

```
Select one of the following and hit return:
      L     ... List
      R     ... Replace an instruction
      D     ... Delete an instruction
      I     ... Insert an instruction
      A     ... Add to the END of the program

      ?     ... Help
      X     ... EXIT

 Selection (L/R/D/I/A/?/X/) ==> L
```

Figure 5.8: Edit/Modify Program Menu

```
NO  OPCODE  Oper1  Oper2  Oper3  Oper4  Oper5  Oper6  Oper7
--  ------  -----  -----  -----  -----  -----  -----  -----
 1       1      0      0      0      0      0      0      0
 2       1      0      0      0      0      0      0      0
 3       1      0      0      0      0      0      0      0
 4       7      0      0      3      0      0      0      0
 5      18      1      3      0      0      0      0      0
 6       3      0      0   -455    385      0      0      0
 7       1      0      0      0      0      0      0      0
 8       1      0      0      0      0      0      0      0
 9       3     70      0      0      0      0      5   1030
10       1      0      0      0      0      0      0      0
11       3    100      0    305     20      0      0      0
12       1      0      0      0      0      0      0      0
13       4      0      0      0      0      0      0      0
14       3    100      0   -250      0      0      0      0
15      11     26      2      1      0      0      0      0
16      19      1      6      0      0      0      0      0
17       3      0   -750      0      0      0      0      0
18       3      0      0    115   -120      0      0      0
19       3      0      0      0      0      0      0    475
20       3      0      0   -385      0      0      0      0
21       1      0      0      0      0      0      0      0
22       5      0      0      0      0      0      0      0
23       1      0      0      0      0      0      0      0
24       1      0      0      0      0      0      0      0
25       0      0      0      0      0      0      0      0
26      11     15      4      1      0      0      0      0
27      18      1      7      0      0      0      0      0
28       3     80    915      0      0      0      0      0
29       1      0      0      0      0      0      0      0
30       3      0      0     45   -200      0      0      0
31       1      0      0      0      0      0      0      0
32       3      0      0      0      0      0      0    615
33       1      0      0      0      0      0      0      0
34       3      0      0      0    200      0      0      0
35       3      0   -360      0      0      0      0      0
36       3      0   -140    200   -270      0      0      0
37       1      0      0      0      0      0      0      0
38       1      0      0      0      0      0      0      0
39       5      0      0      0      0      0      0      0
40       0      0      0      0      0      0      0      0
Pause.
Please press <return> to continue.
```

Figure 5.9: Listing of a program

Remote operation (Option "R") enables the robot controller to function under control from another computer or as a part of a network. When the remote option is selected, control of the robot is returned to the serial port until a key is hit at the keyboard. Commands to read/write register values, to load and execute programs, to suspend and re-start execution of program are implemented. All commands received at the serial port conform to a fully interlocked protocol designed to insure error free transmission. Valid remote commands and detailed explanation of the messaging protocol is given in the later sections.

Some instructions included in the programming language work only in the remote mode. They are skipped if the program is executed in local mode (which would enable the programmer to debug the program locally, without the synchronization). These instructions may provide informative messages or synchronization signals to the remote operator, as well as suspending the execution or transmitting the contents of registers in which statistics on the performance are stored.

When the remote operation is in effect, commands from the remote operator can read and write to the internal registers, and inquire about the operation, such as the last instruction executed. These requests are honored even while the program in the robot is being executed. All commands received from the remote operator are acknowledged and results if any are returned. At the end of the program execution, a message to the remote operator signals that the program has stopped execution. It is also possible to suspend execution of the program using a suspend execution in remote instruction. This instruction, when executed will issue a message to the remote operator, in which the number of the last executed instruction is transmitted. Then the end of execution message will be also issued by the remote monitor. The program can be restarted using "continue execution" command by the remote operator. Serial port is inactive when the remote operation is not in effect.

General purpose storage spaces (called registers) are provided for use by the programmer. These registers may be used for data collection or for control of program flow. Registers are accessible to the user for manipulation through option nine on the main menu. Using this option, the user can initialize, read and change the values of all registers.

The instruction set provides several instructions for accessing registers during program executions (see the section "Instruction Set"). These instructions may be utilized for data collection (counts of events), storage of a position for later retrieval by the program or comparison, and for saving status information on the task being performed.

For ease of manual manipulation (such as Listing contents of registers), a register is marked as "Active" when accessed or altered after bulk initialization of all registers. This enables the user to view only the registers of interest by the "List" command. All registers can be inspected or altered regardless of "Active" status. See Figures 5.10a and 5.10b.

All registers are set to zero (inactive) initially. When a new program is retrieved from disk, registers are also set to the initial status (the idea is that each program uses an independent set of registers). Erasing the program in memory also clears all registers to "Inactive" status and initializes them to zero. Programs that require certain values in registers should load them within the program using appropriate instructions.

One other use of the registers is in the remote operation. Since the registers can be read and written by the remote operator during the remote operation, they may be used to control the flow of program by the remote operator. Instruction set provides instructions to transmit values of registers to remote operator so that they may be used as status indicators that are routinely transmitted during the execution of the program.

A total of fifty registers may be used in any sequence, but the numbers of the used registers may not exceed fifty, or be less than one.

```
    Select one of the following :
          L  ... List active registers contents
          A  ... List ALL registers contents
          D  ... Display a register contents
          C  ... Change a register contents
          R  ... Re-initialize all registers

          ?  ... HELP
          X  ... EXIT

    Selection ==> C

Enter Register No (1 to 50) ==> 32


REGISTER 32 CONTENTS:        0   ACTIVE


Enter NEW value ==> 6


REGISTER 32 CONTENTS:        6   ACTIVE


Pause.
Please press <return> to continue.
```

(a) Change register selection

```
    Select one of the following :
          L  ... List active registers contents
          A  ... List ALL registers contents
          D  ... Display a register contents
          C  ... Change a register contents
          R  ... Re-initialize all registers

          ?  ... HELP
          X  ... EXIT

    Selection ==> L

  NUMBER  CONTENT  ACTIVE?
  ------  -------  -------
     32         6    YES

     1 REGISTERS LISTED

Pause.
Please press <return> to continue.
```

(b) List Registers Selection

Figure 5.10: Edit/Modify Registers Menu

Programming

Twenty-eight instructions have been implemented for programming the robotic arm. They can be grouped as follows:

a) <u>Movement instructions</u>

These instructions cause the Minimover to step motor(s). Included are two move instructions (absolute and relative), an instruction to close the gripper, and an instruction to return the robot to the latest designated "Home" position. Two more instructions cause stepping of motor(s) until a specified condition is met (switch closure or a key strike). Move instructions optionally may have a delay value (to change the speed of movement) which will then become the current delay value for the following instructions.

b) <u>Program Control Instructions</u>

These instructions include several "Conditional Jump" instructions as well as "Stop Execution", "Wait" etc. Conditional Jump instructions cause the program execution to continue at a specified instruction number if certain conditions are met. Conditions are specified in terms of switch status (ON/OFF), register contents (being less than, equal to, greater than, not equal to, etc.), or the position (step counter value) of a motor.

c) <u>Register/Step Counter Manipulation Instructions</u>

These instructions access the internal registers of the robot control system during program execution and can be used for data collection, status indication, or for control of the program flow (when used with related "Jump" instructions). Step counters of the motors can be changed by these instructions thus causing effective change of the "Home" position of individual motors during execution.

d) <u>Remote Mode Instructions</u>

These instructions are executed only when the remote mode is in effect. If any of these instructions are encountered during local mode execution, no action is taken, they are skipped, and execution continues with the next instruction. The instructions implemented are to pause the execution, transmit values stored in registers to the remote operator, and to transmit messages for purposes of signaling events or synchronization. When used in combination with the program control instructions and register manipulation instructions, they form a complete set to remotely monitor and control the operation of the robotic arm.

These instructions form a complete set which may be used to program the robot to perform tasks requiring decisions based on the status of the external switches, process statistics, etc. Details of the structure of the instructions and operands are given in the Appendix.

When the built-in editor is used for programming, prompts for the necessary (and optional) operands of each instruction are displayed, thus making the programming task an interactive and convenient process. However, when an external editor is used, the programmer is responsible for hand assembling each instruction.

During remote operation, commands from the remote operator are received at the serial port. Some of them, such as "Start Execution" or "Load Program" can be issued only when a program is not already being executed. Others, such as read/write registers, can be executed any time.

Commands are transmitted in messages from the remote. Messages consist of an op-code and two parameters. The opcode indicates the command and the necessary operands are transmitted as parameters. Most responses to commands have the same opcode, and if any, the values requested. Sometimes, such as a programmed message a response is generated withcut a command from the remote operator.

A listing and brief discussion of commands are given below:

Opcode = 1, Retrieve Program from disk (or Load Program). This command has one operand, the name of the program. A four-byte filename is transmitted as the parameters.

Opcode = 2, Read Register. The value of the register whose number is the operand of the instruction is transmitted to the remote operator. The response has the same opcode, the same first parameter, and the contents of the register as the second parameter.

Opcode = 3, Write Register. The number of the register and the value to be stored in it are the two parameters transmitted with this message. The request is performed, and the message is transmitted back as acknowledgement.

Opcode = 4, Return Number of Last instruction Executed. This instruction has no operands, and results in transmission of the number of the last executed instruction to the remote operator as the parameter of a message with the same op-code.

Opcode = 5, Execute Program. This instruction causes an acknowledgement (echo back of message) and start of execution of the program loaded in the memory. The remote operator is send an "End of execution message" (Opcode = 8) when the execution ends.

Opcode = 6, Suspend Execution. This command is valid only when a program is executing, and will result in termination of the execution. A response with the number of the last executed instruction as the first parameter will be generated. In addition, an "End of execution" message will also be transmitted.

Opcode = 7, Continue Execution. This instruction will cause the excution to be resumed from the point of suspension. It is not valid when issued during execution of a program. No operands in either the command, nor the response.

Opcode = 8 and 9 are responses to the remote operator. Opcode 8 is an "End of Execution" indicator, and has no parameters. Opcode 9 is an unsolicited response, generated by the executing program and has four byte message as parameters. It is to be interpreted by the remote operator.

Version 6.1 of the microbot controller program is designed to retransmit messages with opcodes 50 to 70 through the second serial port to the device connected there. Operation of the robot controller will be the same in case no messages are intended for the second device, or no connection is present. This feature will also bypass messages from the secondary device to the remote operator (in that case, the workstation controller). See Figure 5.11.

Remote operation of the robotic arm requires communication between the remote operator, which may be a computer, a terminal or other programmable device, and the robotic arm controller program. The portion of the program that performs this function is called the remote operator monitor program. The serial port, which is the connection to the remote, is monitored and messages are interpreted by this program. To be correctly interpreted, messages must conform to the strict transmission format given in Table 5.3, and must be transmitted with the protocol described in Table 5.4.

Table 5.3: Message Format

Opcode	(2 bytes)
Operands	(4 bytes)

Table 5.4: The Messaging Protocol

Initiating Party	Receiving Party
Request (a 15)	
	Acknowledge ("A")
Hi-byte of Opcode	
	Echo Hi-byte
Lo-byte of Opcode	
	Echo Lo-byte
Hi-byte of Parameter 1	
	Echo Hi-byte
Lo-byte of Parameter 1	
	Echo Lo-byte
Hi-byte of Parameter 2	
	Echo Hi-byte
Lo-byte of Parameter 2	
	Echo Lo-byte
End-of-Message (a 15)	
	Acknowledge ("I")

Programmable Logic Controller

The PLC is augmented by a computer to act as a generic programmable machine controller and interface to the computer integrated manufacturing environment (see Figure 5.12). It also acts as the network connection by locally storing and retrieving programs to be downloaded to the PLC, and by accepting and interpreting commands from the workstation controller.

Commands from the workstation controller are received, executed, and responses to the workstation controller, in the form of completion codes, are sent. The machine tool controller also interprets the signals received from the PLC and conveys them to the workstation controller when appropriate. The communications protocol which may be proprietary to the programmable tool is also converted to the protocols used in the CIM environment by the controller. In the case of the PLC, the protocol used for communications is called the ABC protocol. This protocol is used by the computer in communicating with the PLC. Information on the ABC protocol is given in the Appendix. The PLC programs are read from the local disk and downloaded to the PLC using the command structure of the communications interface on the PLC.

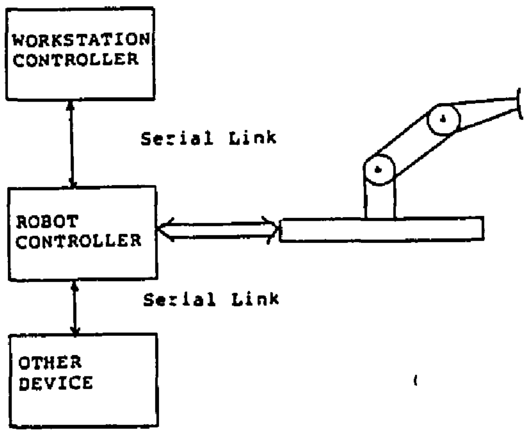


Figure 5.11: Remote Command Bypass Feature

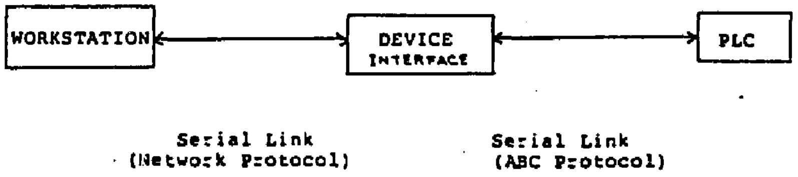


Figure 5.12: Device Interface to CIM

Programming

Commands from the workstation controller are sent in the form of messages with an opcode and two operands. The messaging format and protocol are the same as that of the robotic arm (given in the previous sections and illustrated in Tables 5.3 and 5.4). The opcodes of commands interpreted by the PLC controller are:

Opcode = 51 Load Program to Device. The name of the file to be downloaded to the PLC is the next four bytes transmitted as the two operands.

Opcode = 52 Read Register Value. The value of a register of the PLC is read and sent to the workstation controller. The register number is in the first parameter. Response to this command contains the same message except for the value read from the register is transmitted as the second operand.

Opcode = 53 Write to a Register. Same as the read register instruction, except the value to be written to the register is received as the second operand.

Opcode = 54 Perform Diagnostics. This instruction performs a loopback test to check the operation of the connection to the PLC, and returns error code if necessary. If no errors are detected, zero operands are returned.

Opcode = 55 and 56 are instructions to read and write to the I/O tables of the PLC respectively. Their operation is similar to opcodes 52 and 53, except that the values to be written must be a zero (OFF) or one (ON).

Opcode = 57 Change Loading Address. this instruction alters the address of PLC to which the first program instruction will be downloaded. Normally it is 11, but can also be used to merge programs in PLC memory.

Information on the operation of the PLC is given in the manual for the equipment.

SUMMARY

A workstation controller was designed and implemented on a microcomputer. A small physical model of a workstation was also built by interfacing two programmable devices to the workstation controller. The workstation controller is independent of the the type or number of programmable devices forming the workstation.

One of the devices used is a commercial programmable logic controller (PLC) with communications capability. The other is a programmable manipulator arm with an extensive instruction set and full remote control capability. The two physically implemented devices were chosen as being representative of a wide variety of programmable equipment controllers. They were used to determine the operational requirements of the workstation controller. The interfaces built for the two physical devices are typical of the kind of interfaces that would be required to accommodate different types of devices.

CHAPTER 6

MATERIAL HANDLING SYSTEM SIMULATION

An automated material handling system is a crucial part of any Computer Integrated Manufacturing System. Operation of this system has a significant effect on the performance of the whole system. It is therefore one of the functional modules that simulate the computer integrated manufacturing environment.

The material handling system controller is simulated in two ways: In the context of the network gateway to the cell controller, or separately on a microcomputer connected to the gateway computer. The reason for doing this is to provide the flexibility of being able to run a simple simulation with only two computers, yet to preserve the capability to implement and test complex and powerful decision-makers for the material handling system controller.

In the context of the program simulating the network gateway to the cell controller, a module is implemented to react to the calls for MHS, a simple simulation of the MHS, in case the computer on which the MHS simulation is made is not connected. Also, a mechanism for exchanging messages with the appropriate port when the MHS is addressed exists, so that in the presence of the connection to the MHS controller simulation, messages addressed to the MHS can be routed and responses received.

The MHS is not a subsystem of the cell controller, but receives and responds to requests by the cell controllers. In the simulator it is assumed that the simulated cell controller is the only patron of the MHS, whereas there may be several cell controllers and other subsystems such as maintenance and tooling that may require the services of the MHS.

Operation of the MHS requires decisions on many problems arising from coordination and satisfaction of the demand on the transportation systems (see Figure 6.1). Requests with differing levels of urgency are received, and decisions to commit resources have to be made such as to avoid deadlock situations and to optimally commit resources to activities. In a dynamic environment, the MHS controller must continuously review decisions and requests, such as to continue operation with an optimal set of (or at least a good set of) resource commitment decisions.

The material handling system makes the decisions regarding placement of the work-in-process inventory, and that is simulated in this program. The current implementation is a reaction environment: that is, no effort is made to make intelligent planning at this point. Algorithms for planning and making intelligent resource commitment decisions, or optimization algorithms, could be implemented in the stand-alone version of the MHS simulator and their effects of the change observed in the whole system. These modifications can be made easily since the simulator is modular. The module simulating MHS decision-making would have to be replaced by a module implementing the new algorithm.

The MHS controller requires a facility database, which defines the presence and location of each unit on the shop floor. The facilities database also contains information on the MHS facilities: the types and quantities of transporters, buffer storage, automated storage and retrieval systems, etc., and the types of interfaces of the MHS to the workstations. This data is crucial to the operation of the MHS controller.

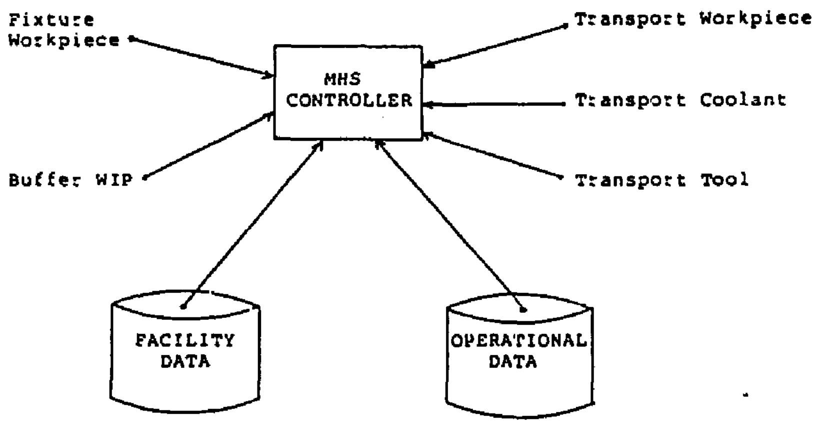


Figure 6.1: Material Handling System Requests

IMPLEMENTATION

The material handling system simulation program runs on a microcomputer. All connections to the factory-wide network are assumed to be in place; however, since the program is running on a microcomputer and the only physical connection to it is the serial link to the gateway simulator computer, all commands from and to the MHS controller are received through the gateway computer connection. Part of the data that would normally be present and readily accessible on the factory-wide database will be read from the disk, or a database query to the cell controller will be made through the gateway computer. Since it is fast, reading from the disk is totally acceptable for simulation purposes. Database queries to the cell controller's database are possible but will not be used unless absolutely necessary, because the serial link is relatively slow even at high transmission rates.

The program is implemented such that there are two parts: simulation of the MHS, and simulation of the MHS controller. Simulation of the MHS is done such that the travelling distances which are available from the facilities database are used to calculate the travel time once the loading is made. Allowances are made for loading and unloading the workpiece.

The MHS controller is implemented such that each request to transport a workpiece is queued. Requests are fulfilled in the order received: available resources are committed to tasks as they become available. Each command is analyzed when received. If the command requests transport to a machine tool that is occupied, and work on the workpiece has been completed, a request for transport of the workpiece to an available buffer is generated and queued preceding the current order. This self-generated transport order can be overridden by a subsequent command for transport of the same workpiece, but will have to be executed before the command to transport the second workpiece is made.

Although the data structures for tracking the orders and their precedence relationships have been created, no attempt has been made to do planning at this point. It can, however, be implemented by replacing the controller simulation module with a module that does planning. This program has been viewed as a necessity for the expansion of the current simulation of the environment, and serves the purpose of the whole project by providing a testground for implementing algorithms to be used in a computer integrated manufacturing environment.

OPERATION

Operation of the material handling system simulation program can be explained in two contexts: decision-making through the MHS controller, and the simulation of the MHS. Both parts are asynchronous programs, that is their inputs and outputs occur at random times. Inputs to the controller are through the serial link. Commands are received from the cell controller(s) and queued. Input queues are checked periodically and assignment of the resources are made as a first-come-first-serve basis, except for the precedence relationships described above. Precedence relationships of the orders, implied by the sequence of orders or generated by the MHS controller, must be satisfied before a commitment is made so that deadlock does not occur. A typical deadlock situation would be the case of a cart waiting for the buffer to be cleared so that it can unload a workpiece, and the workpiece in the buffer waiting for the same cart so that it can be moved and the buffer cleared. Since resource commitments are not made in advance, deadlock situations are not expected to occur in the current implementation.

The material handling system controller keeps track of all workpieces in the shop floor, from the time they enter the system until they are completed and delivered to the shop. Fixturing of the workpieces is required so that they can be transported. Material handling system keeps track of available fixtures and instructs the automated storage and retrieval system to fixture the device so that it can be transported. Since an automated warehousing system simulation is currently not implemented, responses from that unit are simulated in the MHS simulator itself. When that unit is simulated externally, commitment of the transporters would be delayed until the fixtures were applied to the workpieces, or the operation simulated by the external system and messages indicating completion of the task were received.

CHAPTER 7

DISCUSSION AND CONCLUSIONS

The objective of this research was to investigate the feasibility of design and demonstration of the operation of a shop floor control structure to function in the fully integrated manufacturing environment independent of the parts produced or the manufacturing processes involved. The desired properties of the control system included modularity and expandability to accommodate future changes in control algorithms and configuration of the system. It was also desired to take advantage of the databases existing in the CIM environment.

This research has defined the conceptual design of a full CIM environment. This design was successfully tested in a hierarchical control structure implemented on two interconnected microcomputers. The resulting simulation involves the full spectrum of a CIM environment. The simulation was, however, specifically directed at testing the cell control system. It included a free standing cell controller, a workstation controller, and programmable device interfaces. To be able to demonstrate the operation of the cell controller in the CIM environment, responses of the systems that interact with the cell controller were emulated on a microcomputer connected to the cell controller (called the gateway). Requests from the higher level controllers, responses of the workstation controllers, material handling system controller and the factory-wide network were simulated on that computer. This gateway computer can be dynamically re-configured, so that the configuration of the cell can be changed. It is also possible to use free standing implementations of workstation controllers or material handling system controller on other computers that are attached to the gateway via communication links.

A manipulator arm representing a generic programmable equipment was implemented using a microcomputer as the device controller. The control requirements and types of feedback from this model were observed. Also, a programmable logic controller (PLC) with communications capability was connected to a microcomputer representing another type of device interface to the control system. The variety of control requirements and types of feedback by these widely differing systems helped us determine the operational requirements of the workstation and workcell level controllers.

The design of the control system started with an examination of the type of data that would be naturally present in the CIM environment before the part would be released to the shop floor. The design was then built around this available data so that the databases that were present in the system could be utilized directly by the controllers, rather than processing the available data to generate code to drive the shop floor controllers. This led to a data driven structure, rather than a programmable one.

Functions of the cell controller were implemented as functional units that are executed at each cycle of operation. Each functional unit operates on the databases and updates them if necessary. The operation cycle time needs to be short so that each functional unit can take a turn without degrading the real-time response of the system. It is therefore possible to implement a functional unit such that it will complete its function in a number of turns. It is also possible to implement a functional unit as an interface so that it will invoke the function implemented in another computer through the messaging system. It is expected that operations such as optimization whose long execution times warrant a stand alone implementation on a microcomputer will be implemented using this approach.

The capability of the cell controller to function in the presence of many asynchronous events was essential to the operation of the cell controller in real-time. To accommodate all events without tying-up other controllers and yet be able to process all inputs in some sequence, an input output (I/O) queue scheme was designed. All inputs are queued as soon as received, and so are the outputs. To preserve the real-time response of the system, the inputs to the cell controller and outputs from it are processed at every cycle of operation between the operation of each functional unit.

This asynchronous, modular design of the cell controller also accounts for easy modification or expansion of the controller by providing a uniform, easy to access input output interface for all functional modules. The implemented functional modules can be replaced, removed or modified without any changes in the rest of the functional modules.

The approach developed and tested at the cell controller was used to design the workstation controller and the programmable device interfaces. They use essentially the same approach to dealing with the problem of asynchronous events.

The workstation controller designed for the model workstation has a limited command set, yet it is capable of handling a wide variety of devices. The implementation which defers the binding of the devices to the point of program loading provides this flexibility.

The generic cell control system concepts proposed in this work utilize the high level of standardization required in the CIM environment and emphasize modularity and expandability. Unlike most of the designs discussed in the background section, the proposed system design does not address the interim needs of the market, nor does it provide solutions for existing manufacturing facilities. This work is however, relevant to the needs of full CIM implementations to be made in the future. This research has tested the concepts developed by demonstrating a simple and structured design for shop floor control, and provides a testbed to aid in future extensions of algorithms for that environment.

FURTHER RESEARCH

One of the contributions of this project has been the creation of a CIM environment which can be used for development and testing of algorithms for dynamic planning and real-time control of manufacturing in the integrated environment. For example, the rules used for selection of jobs to be assigned to available workstations is currently made in a simple way. A look-ahead type of algorithm could be implemented by replacing the present one without difficulty. Effects of this change may then be measured in the simulated cell operation in terms of machine utilization or reduction in work-in-process inventory.

Similar replacements may also be made in the material handling system controller and new methods for allocating transport vehicles to requests for transportation of workpieces, or allocation of work-in-process inventory to buffer storages may be made. The data structures for the required information has been implemented, even if not fully utilized. An example is the material handling system interface type. Data structures for this information has been provided so that a more sophisticated model of the material handling system where more than one type of transport system is present may be simulated. The simulated base model however has used one type of transport system.

The emulator developed for this project is designed to serve as a laboratory for further research in this area. It provides a structure or frame that can be filled to the extent desired. Configuration of the CIM environment can be made to fit the requirements of the experiment desired. The allocation of ports to the connected microcomputers (on which MHS simulation or workstation controller may run) is made using configuration files which indicate presence of each connection, and hence the configuration of the experiment. It may be desirable to test cell controller algorithms in the two-computer base model, on one of which the cell controller runs and the other is the simulation of the CIM environment. Other cases may require better simulation of the material handling system, hence three computers for running the experiment. Still another configuration may be physical simulation of one or more workstations, hence more connections.

Only one implementation of the developed workstation controller was made because the amount of programmable equipment at hand was limited and the contribution of the additional workstations would be marginal. As stated earlier, however, this workstation model is general enough so that it can be duplicated when additional equipment is available, and construction of additional workstations is desirable.

Another addition to the model may be the tooling system, which is more or less similar to the material handling system as far as the relationship to the manufacturing cell is concerned. The data structures for future addition of this system are also implemented.

The improvements that can be made to this system can be grouped in two separate areas: hardware and software. As far as the hardware is concerned, one critical improvement would be the inclusion of a network and communications controller which would provide high speed reliable transmission and access to data bases. The serial transmission methods used are limited to lower speeds of transmission. The absence of good buffering and support of operating system in some cases has necessitated the use of fully interlocked protocols for transfer of information, hence unnecessarily slowing down the communications.

On the software side, an operating system that supports multitasking and good communications support between tasks would improve the performance of programs written to take advantage of them.

The next step to be taken in the direction of this research is the addition of the shop control level, which would be the supervisor of cell controllers. Development of the planning modules at several levels should however be done before expanding the model. Implementation of decision makers using expert systems or artificial intelligence methods should also be considered at each level.

REFERENCES

1. Albus, J. S., Barbera, A. J., Fitzgerald, M. L., Kent, E., McLean, C., McCain, H., Bloom, H., Haynes, L., Furlani, C., Barkmeyer, E., Mitchell, M., Scott, H., Bloomquist, D., Kilmer, R., "A Control System for an Automated Manufacturing Research Facility," Robots 8 Conference Proceedings, Detroit, Michigan, June 4, 1984

2. Albus, J. S., Barbera, A. J., and Nagel, R. N., "Theory and Practice of Hierarchical Control," Twenty-third IEEE Computer Society International Conference, Sept. 1981, pp 18-39

3. Barbera, A. J., Fitzgerald, M. L., Albus, J. S., Haynes, L., S., "RCS: the NBS Real-time Control System" Robots 8 Conference Proceedings, Detroit, Michigan, June 4, 1984. pp 19.1-19.33

4. Bjorke, O., "Towards Integrated Manufacturing Systems - Manufacturing Cells and Their Subsystems," Robotics and Computer-Integrated Manufacturing Journal, Vol.1, No.1, pp 3-19

5. Brondolese, A., Garetti, M., "FMS Control Systems: Design Criteria and Performance Analysis," Proceedings of the 2nd International Confereence on Flexible Manufacturing Systems, Oct 1983, pp 365-381

6. Bullers, W. I. Jr., "Logic Programming For Manufacturing System Specification," Proceedings of 1986 IEEE International Conference on Robotics and Automation, San Francisco, California, April, 1986, pp 1831-1836

7. Chang, P. S., "Developing a Prototype Microcomputer Network for Implementing a Manufacturing Planning and Control System in Small Manufacturing Companies," Conference Proceedings of 11th Conference on Production Research and Technology, May, 1984

8. Chochon, H. and Alami, R., "NNS, A Knowledge Based On-Line System For An Assembly Workcell," Proceedings of 1986 IEEE International Conference on Robotics and Automation, San Francisco, California, April, 1986, pp 603-609

9. Dato, M. A., "Control Systems for Integrated Manufacturing - The CAM Solution," Proceedings of 1986 IEEE International Conference on Robotics and Automation, San Francisco, California, April, 1986, pp 791-795

10. DiCesare, F., Goldbogen, G., Langan, D., Suresh, R., Desrochers, A., "Functions of a Manufacturing Workstation Controller," Proceedings of 1986 IEEE International Conference on Robotics and Automation, San Francisco, California, April, 1986, pp 1470-1475

11. Elgabry, K. A., "Communicating Product Definition and Support Data in a CAE/CAD/CAM Environment", Proceedings of CASA/SME Autofact'85 Conference, Detroit, Michigan, November 1985, pp 9-11

12. Gatelmand, C. D., and Vignaud, J. P., "FMS Design and Specifications", Autofact-Europe, September 1983.

13. General Motors' Manufacturing Automation Protocol, General Motors Corporation, Warren, Michigan, 1985

14. Harned, J., and Holcman, S. B., "An Approach for Solving The CIM Gap Problem", Proceedings of CASA/SME Autofact'85 Conference, Detroit, Michigan, November 1985, p 3-7

15. Haynes, L. S., Wavering, A. J., "Real-Time Control System Software: Some Problems and Approach," Proceedings of 1986 IEEE International Conference on Robotics and Automation, San Francisco, California, April, 1986. pp 1470-1475

16. Haynes, L. S., Barbera, A. J., Albus, J. S., Fitzgerald, M. L., McCain, H. G., "An Application Example of the NBS Robot Control System," Robotics and Computer-Integrated Manufacturing Journal, Vol.1, No.1, 1984, pp 81-95

17. Jones, A. T., McLean, C. R., "A Proposed Hierarchical Control Model for Automated Manufacturing Systems," Journal of Manufacturing Systems, Vol.5, No.1, 1986, pp 15-25

18. Krizsan, A., Haidegger, G., and Nagy, S. S., "Manufacturing Systems Based on Distributed Intelligence Cell Concept," Integration of CAD/CAM, Editor: D. Kochan, IFIP, 1984, pp 253-263

19. Li, C., H., "Computer Integrated Self Optimizing Factory," Proceedings of CASA/SME Autofact'85 Conference, Detroit, Michigan, November, 1985, pp 3.54-3.74

20. Liu, T., Simon, M., et al. "Machine Control Software Design," Proceedings of 1986 IEEE International Conference on Robotics and Automation, San Francisco, California, April, 1986, pp 1470-1475

21. Maimon, O., "FMS Real-Time Operational Control," Proceedings of CASA/SME Autofact'85 Conference, Detroit, Michigan, November 1985, pp 6-31

22. Microbot, Inc., Minimover-5 Robotics Reference and Applications Manual, Microbot, Inc., Mountain View, California, 1982.

23. Neal, R. E., DeMint, P. D., Klages, E. J., et.al, "Integrated Manufacturing Information and Control System," Proceedings of CASA/SME Autofact'85 Conference, Detroit, Michigan, November 1985, pp 17-16

24. Pamukcu, D.,"Computer Programs for a Generic Manufacturing Cell Control System," IE 87-2 Working Paper Series, Industrial Engineering Department, Louisiana State University, Baton Rouge, Louisiana, May, 1987.

25. Ranky, P. G., The Design and Operation of FMS, North-Holland Publishing Co., 1983.

26. Singh, M. G., Dynamical Hierarchical Control, North-Holland Publishing Co., 1980

27. Solberg, J. S., Paul, R. P., and Anderson, D. C., "The Factory of The Future: A framework for Research," Conference Proceedings of 11th Conference on Production research and Technology, May, 1984, pp 53-58

28. Villa, A., Mosca, A., Murari, G., "Expert Control Theory: A Key For Solving Production Planning and Control Problems in Flexible Manufacturing," Proceedings of 1986 IEEE International Conference on Robotics and Automation, San Francisco, California, April, 1986, pp 466-471

APPENDIX A

ROBOT-5 INSTRUCTION SET

Robot-5 is the software designed to program and control a Minimover-5 robotic arm. Detailed information on this system is given in Chapter 5. This appendix contains complete information on the instruction set and structure of each instruction for programming the robotic arm.

LIST OF INSTRUCTIONS

Following is a list of the implemented instructions by opcode:

0 - **Stop Execution.**

No Operands.

1 - **No Operation.**

This instruction causes the execution to continue with the next instruction. No operands.

2 - **Absolute Move.**

This instruction will step all motors until the step counter of each motor is the same as a given value (absolute position).

Operands:

A delay value (optional) to specify the speed of the stepper motors.

Positions of each motor (step counter values). All motors need not be specified, in which case they shall not be moved (a step count value of -9999 means not specified).

3 - **Move Relative**

This instruction will step all motors with the number of steps specified for each motor. Operands are the same as (2) but if a motor need not be moved, a zero value may be specified for the number of steps for that motor.

4 - **Close Gripper (GRAB).**

This instruction will pulse the gripper motor (6) until the gripper switch is closed. No operands.

5 - **Return To Home Position.**

All motors will be pulsed until the step counters are zero. No operands.

6 - **Move Motor Until Switch is Hit.**

This instruction will move the specified motor in the indicated direction until the designatedswitch is in the position specified.

Operands:

Delay value (optional, if change of speed is desired),

Motor number to be moved,

Switch number to be checked,

Condition to stop (ON/OFF), Direction of movement (Forward/Backward).

7 - **Wait (idle) UNTIL Switch is ON/OFF.**

This instruction will hold the execution of the program until a switch is at the indicated position (ON/OFF).

Operands:

Switch Number to check,

Triggering position of the switch.

8 - **Move motor Until a Key is Hit.**

This instruction will move a motor until a key (at the keyboard) is struck.

Operands are the same as (6) except for switch number

9 - **Set Delay Value (Speed of motors).**

This instruction changes the delay value (explained above) resulting the change in the speed of the motors.

Operand is the delay value.

10 - **Unconditional Jump.**

This instruction will cause the execution to continue with the instruction specified.

Operand is the instruction number.

11 - **Jump on Switch Condition.**

This instruction will cause the execution to continue with the given instruction if a switch is in the indicated condition. If the condition is not met, execution will continue with the next instruction.

Operands:

Instruction number for jump,

Switch number to check,

Condition of the switch (ON/OFF) which will result in Jump.

12 - **Jump if Switch Not in the Given Position.**

This instruction is the same as (11) except that it will cause a jump if the switch is not in the specified position.

Operands are the same as (11).

13 - **Jump on the Position of the Motor.**

This instruction will cause execution to continue at the given instruction if the specified motor's step counter, compared to the given value meets the condition for jump. Otherwise, execution will continue with the next instruction.

Operands:

Instruction number for jump,

Motor number,

Value for comparison,

Condition (equal to, less than, greater than, not equal, less than or equal to, greater than or equal to).

14 - **Jump on the Contents of the Register.**

This instruction will cause the execution to continue at the given instruction number, if the specified value compared to the contents of the specified register meets the condition for jump. Otherwise execution will continue with the next instruction.

Operands:

Instruction number for jump,

Register number,

Value for comparison,

Condition (equal to, less than, greater than, not equal, less than or equal to, greater than or equal to).

15 - **Set all Registers to a Value.**

This instruction will set the contents of all registers to the value given. Previous contents of all registers will be lost. All registers will be marked as unused (since reset).

Operands:

Value to store in all registers.

16 - **Store a Value in a Register.**

This instruction will set the contents of a register to the value given. Previous contents of the register will be lost. The register will be marked as used.

Operands:

Value to store in the register.

17 - **Add To Register.**

This instruction will add a value to the specified register. The register will be marked as used.

Operands:

Register number,

Value to be added to the register.

18 - **Increment Register Contents.**

This instruction will add one to the contents of the specified register. A useful instruction for use in collecting event counts etc.

Operand is the register number.

19 - **Decrement Register Contents.**

Similar to (18), will subtract one from the contents of the specified register.

Operand is the register number.

20 - **Add Two Registers.**

This instruction will add the contents of two specified registers, and the result will be stored in the third register. Register numbers of any or all the operands may be the same. Result register's previous contents will be lost. All three registers will be marked as used.

Operands are three register numbers.

21 - **Subtract Two Registers.**

Contents of the second specified register will be subtracted from the contents of the first register. Result will replace the contents of the third register.

Operands are three register numbers. Any or all of the register numbers may be same.

22 - **Declare Current Motor Positions "Home".**

This instruction will zero the step counters of all motors, thus declaring the current position of the arm as the "Home" position. Contents of the step counters of motors will be lost.

No operands.

23 - **Set Step Counter of a Motor to a Value.**

This instruction will replace the current step counter of the motor specified with the given value. It may be used to declare the current position of the motor as home (value = 0). Previous value of the step counter of the motor will be lost, therefore effectively altering the home position of the motor.

Operands:

Motor number whose step counter will be involved,

Value to initialize the counter.

24 - **Copy Value in Register to Step Counter of a Motor.**

This instruction will set the step counter of a motor to the value specified in the specified register. Previous contents of the step counter of the motor will be lost, effectively altering the Home position for the motor. Contents of the register will be unchanged.

Operands:

Motor number whose step counter will be changed,

Register number whose contents will be used.

25 - **Copy Step Counter of a Motor to a Register.**

This instruction will copy the current value of the step counter of the motor to a Register. Contents of the step counter will not be changed. Previous contents of the register will be lost.

Operands are:

Motor number whose step counter value will be copied,

Register number to be used for copy.

26 - **Suspend Execution (in remote mode only)**

This instruction is equivalent to stop execution instruction (opcode = 0) except for the message that is immediately transmitted to the remote operator, and containing the number of the last executed instruction. The "End of Execution" message is also sent to the remote operator. Execution can be resumed with the next instruction in the program when the "Continue Execution" instruction is received from the remote operator. This instruction will be skipped during execution if opertion is local (not remote mode).

No Operands

27 - **Transmit Contents of a Register to Remote Operator** (in remote mode only)

This instruction causes the robot controller to transmit a message to the remote operator. Operation code of the message will be (2) similar to that of a request by the remote operator, and the content of the register specified by the operand will be transmitted as the second parameter of the message.

This instruction enables the programmer to send piece counts, repeat values etc., to the remote operator during execution, therefore generating feedback to the remote operator.

This instruction will be skipped during execution if opertion is local (not remote mode).

The only operand is the register number whose contents (value) will be transmitted.

28 - **Transmit a Four Byte Message to Remote Operator** (in remote mode only)

This instruction enables the programmer to send a signal to the remote operator. It may be a synchronization signal, a flag to inform the operator of start or end of a move, or any other informative message. The message is coded into the instruction, and therefore can not be changed during execution.

This instruction will be skipped during execution if operation is local (not remote mode).
Coding of the message is such that four bytes will be transmitted. Operation code of the message will be (9), and the two parameters of the message are contained in the first two operands of the instruction (two bytes per operand).

INSTRUCTION STRUCTURES BY OPCODE

0 - **Stop Execution**

No Operands (Blank instruction)

1 - **No Operation**

No Operands.

2 - **Absolute Move**

Operands:

1) Delay Value (or zero for no change in speed),

2-7) Position to step to (for each motor, in order).

Value -9999 means no movement for that motor.

3 - **Move**

Operands:

1) Delay Value (Speed),

2-7) Number of steps for each motor, in order.

Value of zero means no movement for that motor.

4 - **Close Gripper**

No Operands.

5 - **Return to Home Position**

No Operands.

6 - **Move Motor Until Switch is ON/OFF**

Operands:

1) Delay Value (zero for no change),

2) Motor number (1-6)

3) Switch Number,

4) Switch Position (1=ON, 0=OFF),

5) Sense of Movement (Forward=1, Reverse= -1).

7 - **Wait (idle) Until Switch is ON/OFF**

Operands:

1) Delay Value (zero for no change),

2) Motor number (1-6)

3) Switch Number,

4) Switch Position (1=ON, 0=OFF).

8 - **Move Motor UNTIL a Key is Struck at the Keyboard**

Operands:

1) Delay Value (zero for no change),

2) Motor number (1-6),

5) Sense of Movement (Forward=1, Reverse= -1).

9 - **Set Delay Value (speed of motors)**

Operands:

1) Delay Value (NONZERO).

10 - **Unconditional Jump**

Operands:

1) Instruction number.

11 - **Jump on Switch Condition**

Operands:

1) Instruction number,

2) Switch Number,

3) Condition of Switch for Jump (On=1, OFF=0).

12 - **Jump on NOT Switch Condition**

Operands:

1) Instruction number,

2) Switch Number,

3) Condition of Switch for Jump (On=1, OFF=0).

13 - **Jump on Step Counter of a Motor**

Operands:

1) Instruction number,

2) Motor Number,

3) Value for comparison (to step counter),

4) Condition for Jump:

   = 1 Equal to,

   = 2 Less Than,

   = 3 Greater than,

   = 4 Not equal,

   = 5 Less than or equal,

   = 6 Greater than or equal.

14 - **Jump on Register Value Comparison**

Operands:

1) Instruction number,

2) Register Number,

3) Value for comparison (to register contents),

4) Condition for Jump:

   = 1 Equal to,

   = 2 Less Than,

   = 3 Greater than,

   = 4 Not equal,

= 5 Less than or equal,

= 6 Greater than or equal.

15 - **Set All Registers to a Value (initialize)**

Operands:

1) Set Value,

2) Zero.

16 - **Set a Register to a Value (initialize)**

Operands:

1) Set Value,

2) Register Number.

17 - **Add a Value to a Register**

Operands:

1) Value to add,

2) Register Number.

18 - **Increment a Register (add one to contents)**

Operands:

1) One,

2) Register Number.

19 - **Decrement a Register** (subtract one from contents)

Operands:

1) Minus One,

2) Register Number.

20 - **Add Two Registers** (r1) + (r2) -> (r3)

Operands:

1) First Register Number,

2) Second Register Number,

3) Result Register Number.

21 - **Subtract Two Registers** (r1) - (r2) -> (r3)

Operands:

1) First Register Number,

2) Second Register Number,

3) Result Register Number.

22 - **Reset (to zero) step counters of all motors**

No Operands.

23 - **Set Step Counter of a Motor to a Value**

Operands:

1) Set Value,

2) Motor Number (=0 for all motors).

24 - **Copy Contents of a Register to a Step Counter**

Operands:

1) Register Number,

2) Motor Number.

25 - **Save Contents of a Step Counter to a Register**

Operands:

1) Register Number,

2) Motor Number.

26 - **Suspend Execution (in remote mode only)**

No Operands

27 - **Transmit Contents of a Register to Remote Operator**
(in remote mode only)

Operands:

1) Register Number,

28 - **Transmit a four byte message to remote operator**
(in remote mode only)

Operands:

1) First two bytes of message (an integer),

2) Next two bytes of message (an integer).

APPENDIX B

ABC PROTOCOL AND THE COMMUNICATIONS INTERFACE MODULE OF THE PROGRAMMABLE LOGIC CONTROLLER

GENERAL

The Honeywell 620 Programmable Logic Controller (PLC) has three parts that must be distinguished: the input/output (I/O) tables, sixteen bit registers, and program memory.

The I/O consists of two parts: external and internal. External I/O are the group of I/O bits that can be physically connected to the outside world through the modules in the PLC. Internal I/O are the group of bits that can be used in the same manner as the external I/O, but can not be physically connected to external devices.

All I/O is single bit (i.e. a one or a zero). Status of all I/O bits are kept in a table called the I/O table. PLC periodically checks the status of the inputs and updates the I/O table. Outputs are updated when referred by the PLC program.

Honeywell 620 PLC provides a number of sixteen-bit registers that can be used by the programmer. Among the possible uses of these registers are timer preset values, piece counts, etc.. These registers can be accessed by the PLC program.

The program memory of the PLC consists of 24-bit program words. Program instructions are stored sequentially. First few locations in the program memory are used for diagnostic, etc., however, locations starting with 11 can be used by the application program.

COMMUNICATIONS INTERFACE MODULE

One of the modules available for the Honeywell 620 series PLC is the Communications Interface Module (CIM) that can be used to connect the PLC to the external devices using serial connections. The CIM port may be used to connect the PLC to terminals, computers, or to networks, and may be programmed for different modes of communication.

CIM uses Asynchronous Bisync Communications (ABC) protocol. Commands received through the CIM port are checked for the correct opcodes and are responded to by the PLC (or the CIM itself). Detailed description of the operation and configuration of CIM is given in the PLC CIM manual.

ABC PROTOCOL

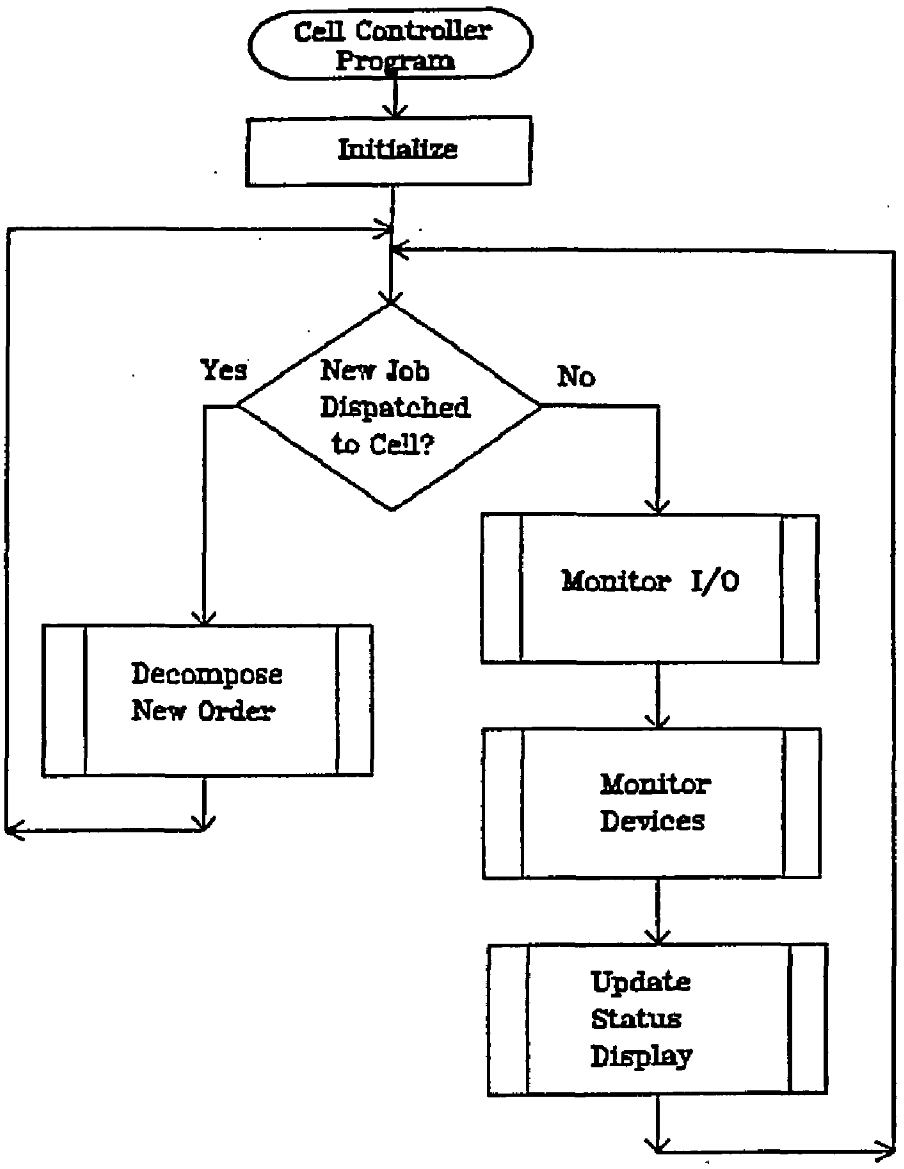
This protocol uses leader and follower bytes for each message. The header bytes are: SOH (ASCII 1), Node Address, a control character identifying the message type, ETB (ASCII 23), STX (ASCII 2). Following these five bytes is one byte opcode and two bytes of message length. The message length is the number of bytes following the eighth byte (the lo-byte of the length), excluding the follower bytes. The follower bytes are checksum (one byte) and ETX (ASCII 3). The checksum includes everything including nodal address to (excluding) checksum.

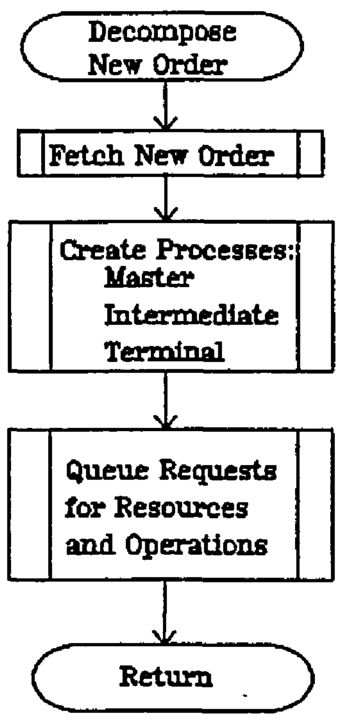
Detailed description of the ABC protocol is also given in the PLC CIM manual. This protocol was implemented on microcomputer using assembly language.

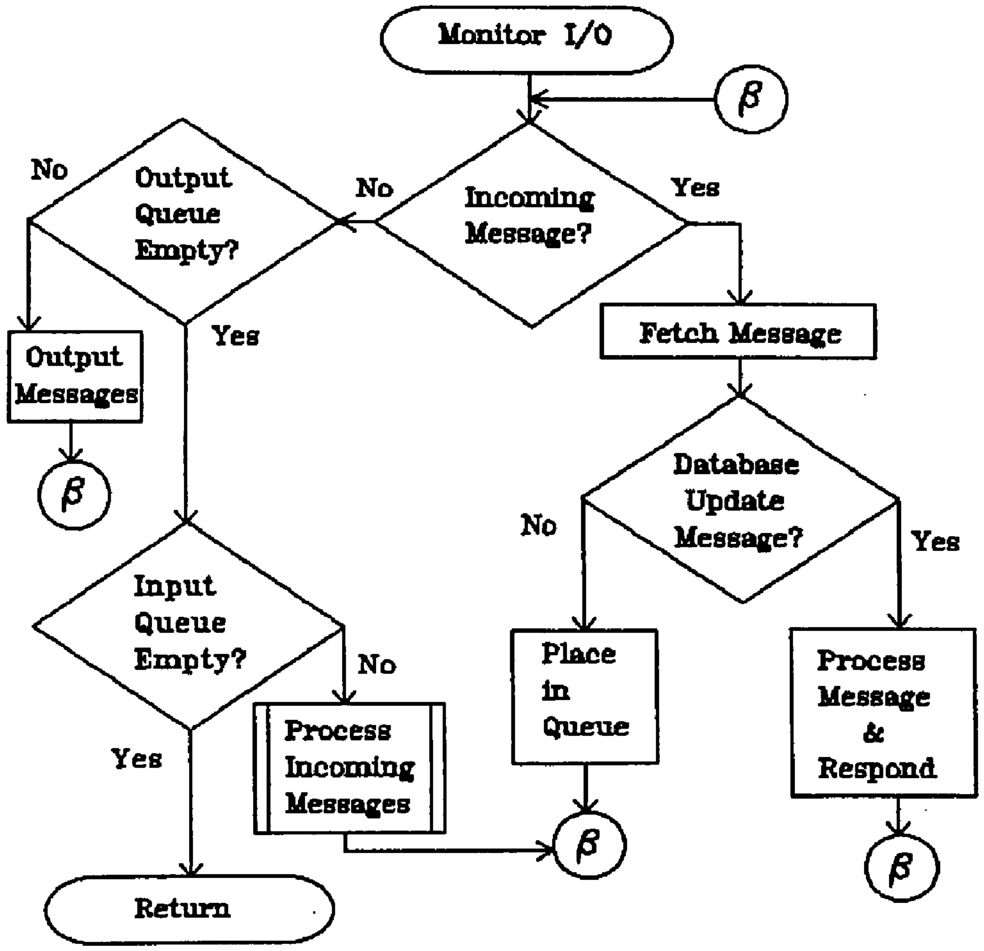
APPENDIX C

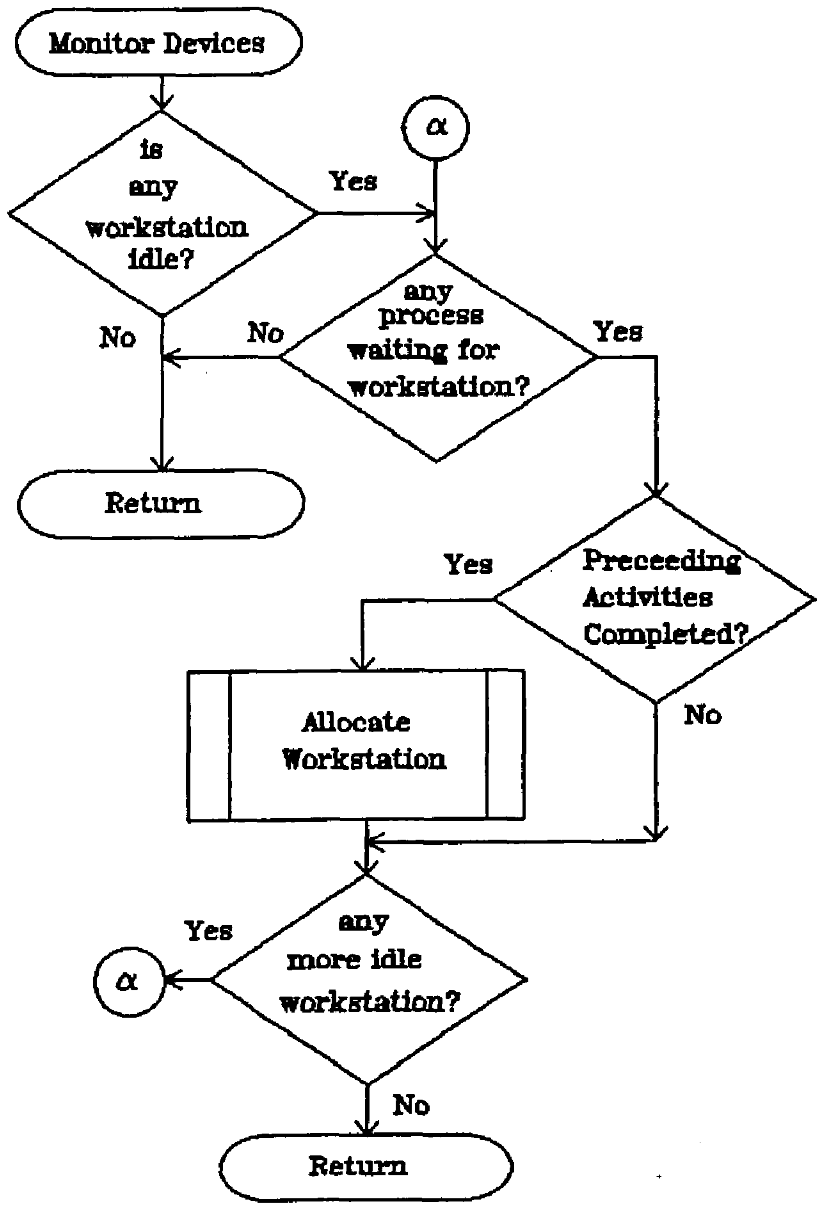
MAJOR FLOWCHARTS

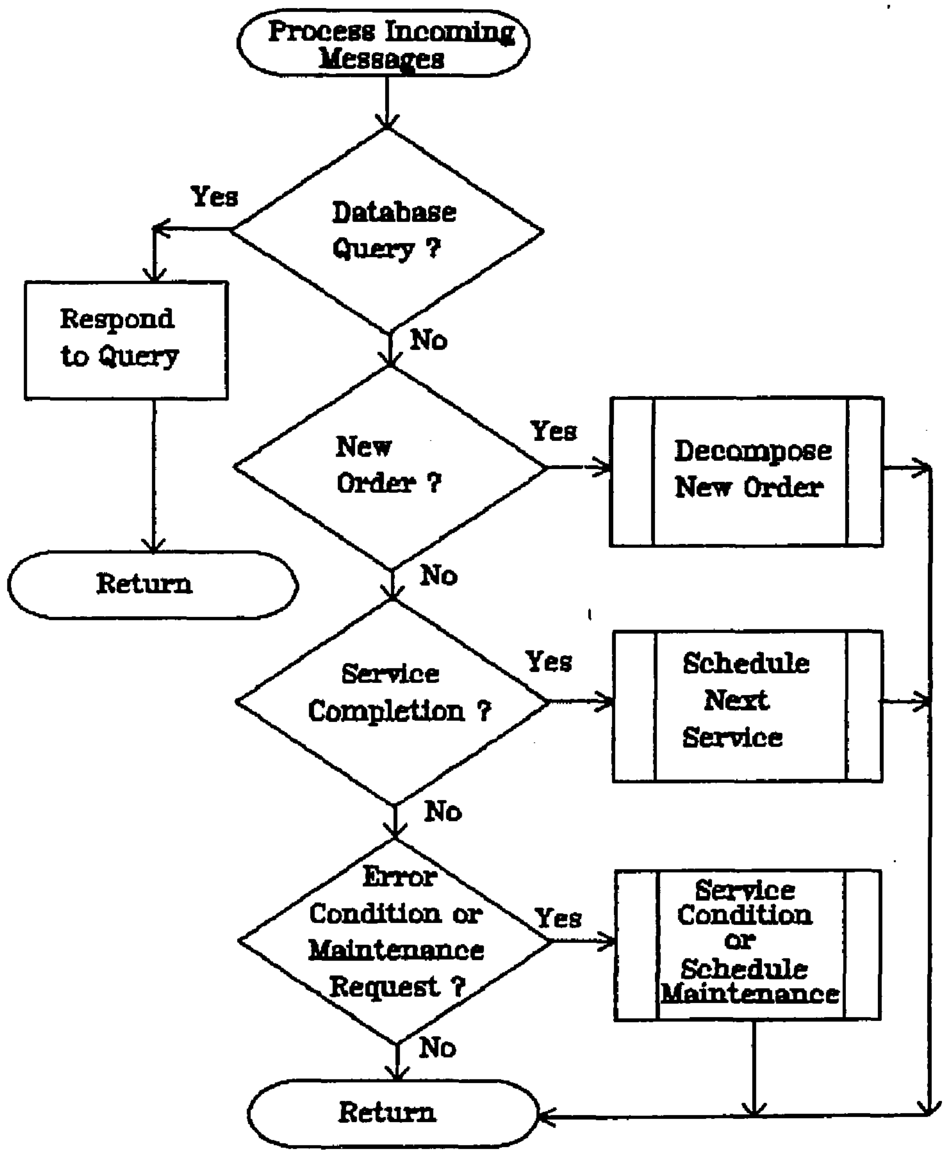
MAJOR FLOWCHARTS
FOR THE CELL CONTROLLER PROGRAM


Cell Controller Program
Initialize
New Job Dispatched to Cell?
Yes
No
Decompose New Order
Monitor I/O
Monitor Devices
Update Status Display


Decompose New Order
Fetch New Order
Create Processes:
Master
Intermediate
Terminal
Queue Requests for Resources and Operations
Return

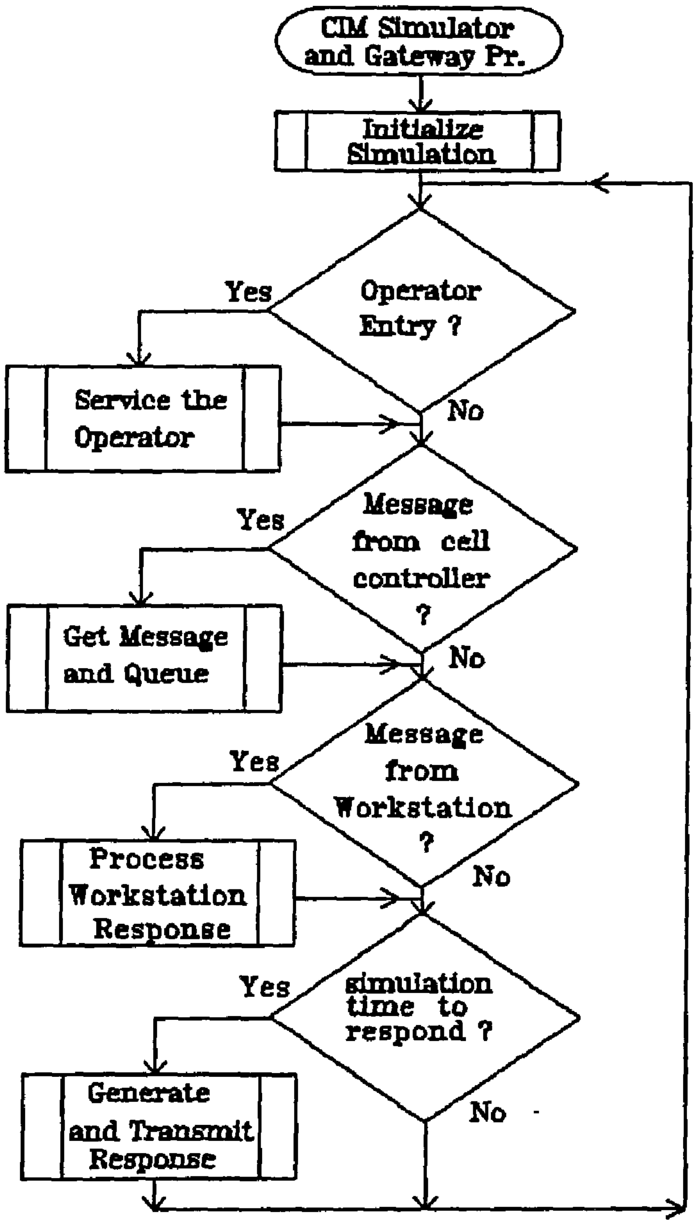

Monitor I/O
β
Incoming Message?
No
Yes
Output Queue Empty?
No
Yes
Output Messages
β
Fetch Message
Database Update Message?
No
Yes
Input Queue Empty?
No
Yes
Process Incoming Messages
Place in Queue
Process Message & Respond
β
β
Return

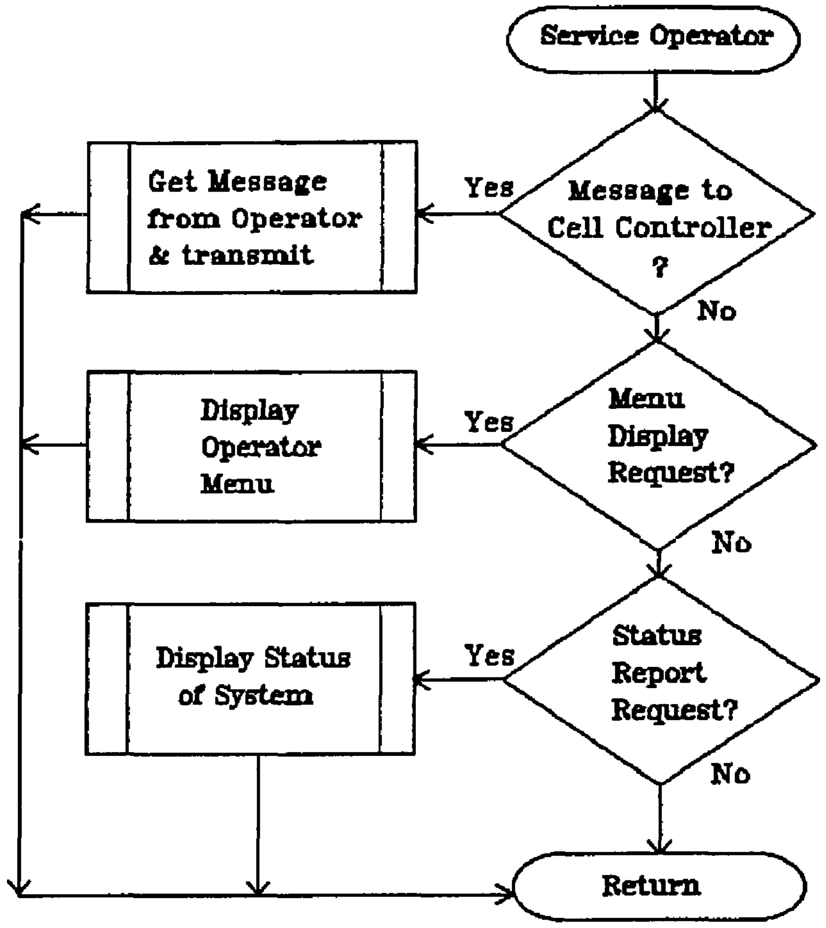

Monitor Devices
is any workstation idle?
Yes
No
α
any process waiting for workstation?
No
Yes
Return
Preceeding Activities Completed?
Yes
No
Allocate Workstation
any more idle workstation?
Yes
α
No
Return

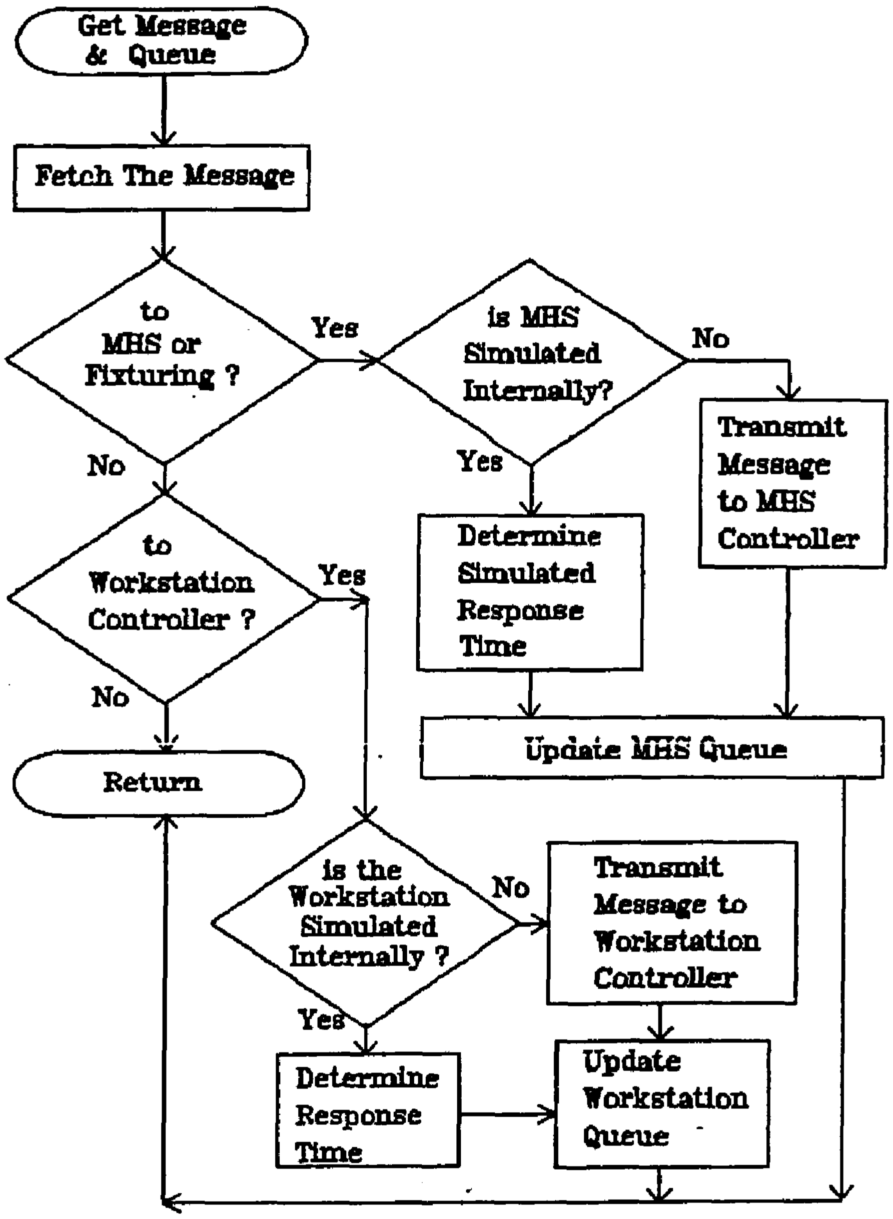

Process Incoming Messages
Database Query ?
Yes
Respond to Query
Return
No
New Order ?
Yes
Decompose New Order
No
Service Completion ?
Yes
Schedule Next Service
No
Error Condition or Maintenance Request ?
Yes
Service Condition or Schedule Maintenance
No
Return

MAJOR FLOWCHARTS

FOR THE GATEWAY CONTROLLER PROGRAM


CIM Simulator and Gateway Pr.
Initialize Simulation
Operator Entry ?
Yes
No
Service the Operator
Message from cell controller ?
Yes
No
Get Message and Queue
Message from Workstation ?
Yes
No
Process Workstation Response
simulation time to respond ?
Yes
No
Generate and Transmit Response


Service Operator
Message to Cell Controller ?
Yes
Get Message from Operator & transmit
No
Menu Display Request?
Yes
Display Operator Menu
No
Status Report Request?
Yes
Display Status of System
No
Return


Get Message & Queue
Fetch The Message
to MHS or Fixturing ?
Yes
No
is MHS Simulated Internally?
No
Yes
Transmit Message to MHS Controller
Determine Simulated Response Time
Update MHS Queue
to Workstation Controller ?
Yes
No
Return
is the Workstation Simulated Internally ?
No
Yes
Transmit Message to Workstation Controller
Determine Response Time
Update Workstation Queue

VITA

The author, Derya Pamukcu, was born in Ankara, Turkey on June 25, 1955. He completed a B.S. in Mechanical Engineering in September, 1977 at Bogazici University in Istanbul, Turkey. After working as a project engineer for two years at Pasabahce Glassworks of Istanbul, Turkey, Mr. Pamukcu enrolled at Louisiana State University in January, 1979.

At L.S.U. , Mr. Pamukcu has worked on a part time basis in the Energy Programs Office as a graduate research assistant and later as a graduate teaching assistant for the Industrial Engineering Department. After receiving his Ms. in Industrial Engineering in May, 1982, he worked as a full-time Instructor for the same department.

DOCTORAL EXAMINATION AND DISSERTATION REPORT

Candidate: Derya Pamukcu

Major Field: Engineering Science

Title of Dissertation: Design of a Generic Manufacturing Cell Control System

Approved:

[illegible]
Major Professor and Chairman

[illegible]
Dean of the Graduate School

EXAMINING COMMITTEE:

M. Zohdi

[illegible]

[illegible]

[illegible]

[illegible]

Date of Examination:

March 12, 1987