An Integrated Intelligent Shop Control System

by

Yaoen Lan Zhang

Thesis submitted to the Faculty of the

Virginia Polytechnic Institute and State University

in partial fulfillment of the requirements for the degree of

Master of Science

in

Industrial Engineering and Operations Research

APPROVED:

Michael P. Deisenroth, Ph.D., Chairman

H. JoAnne Freeman, Ph.D., P.E.

Subhash C. Sarin, Ph.D.

April, 1989

Blacksburg, Virginia

An Integrated Intelligent Shop Control System

By

Yaoen Lan Zhang

Michael P. Deisenroth, Chairman

Industrial Engineering and Operations Research

(ABSTRACT)

Presently there is a trend in manufacturing system design from stand-alone, automatic operations to Computer Integrated Manufacturing (CIM). The success of the integration depends largely on the performance of system control software. This document presents research in the design and implementation of a shop control system, for a CIM facility, using a new method called the three-layer integrated approach. Two basic techniques used in this research are expert systems and object-oriented programming.

The CIM system at VPI is an automated manufacturing and assembly system. In designing the control system for this CIM facility, the design of products, production facilities, and the overall system must be taken into account. In order to manage this complex system, a control system called the "shop controller" has been developed using C++, an object-oriented programming language. In addition, three real-time simulators for the cell controllers have been developed for testing and debugging the production rules of the expert system.

The basic approach taken for the shop control system has several advantages: applied intelligence, program efficiency, reusability of code, and ease of maintenance. Moreover, this approach has a new feature -- modularity, which is the result of combining expert systems and object-oriented programming.

Dedication

This work is dedicated to my parents and in gratitude for their many years of love and support.

Acknowledgements

The author would like to express his great thanks to his adviser Dr. M. P. Deisenroth for his endless guidance and support in this effort, especially in defining the thesis topic, and in directing the development of the methodology.

The author would also like to thank Dr. H. J. Freeman, and Dr. S. C. Sarin for serving on his thesis committee and providing ideas. Dr. Freeman deserves special thanks fot her dedication of time and effort towards this research.

The author would like to thank Dr. Rick Swindell for the proof-reading. Finally the author wants to thank his friends: , , , , , , and his roommates and .

Table of Contents

List of Illustrations

List of Tables

1. Introduction

1.1 Background

In the United States, the gross national product is composed of service industries (50%), manufacturing industries (40%), and extractive (e.g. agriculture) and construction industries (10%) [U.S. Dept. of Commerce, 1986]. Since "service" does not result in direct creation of wealth, it can be seen that manufacturing industries contribute a major portion of the gross national product. Obviously, any tool that increases manufacturing productivity will have a profound effect on the real wealth of the United States. In recent years, new technology, particularly computer integrated manufacturing (CIM) technology, has demonstrated great potential for improving the manufacturing productivity.

Most manufacturing components in the U.S. are produced in low-volume lots of 50 pieces or less. This means that a great deal of time is spent on altering part routes and re-programming machine operations, which results in slow and costly production sequences. At the present time, it is estimated that up to 95 percent of the production time of work-pieces in a batch-type metal-working shop is spent simply moving or waiting [Ham, 1985].

In order to improve the throughput of the production process, increase productivity, reduce manufacturing costs, and become more

competitive in the world market place, many small-lot batch manufacturing companies have been trying to implement CIM systems with capabilities that allow for the flexible and changing manufacturing requirement needs for product variants and for controlling material flow.

The objective of CIM is to develop a cohesive database that integrates manufacturing, design, and business functions. Information is sent on demands and as needed to the largest number of intra- and interdisciplinary groups as possible. The ideal CIM controlled system will design and manufacture without disruption from raw material to the finished products. The typical CIM functions are illustrated in Table 1.1 [Feicholz 1984].

In implementing such a CIM, an organizational structure and a communication mechanism are required which tie the several different processes together. To put it another way, a CIM system involves linking together numerically-controlled machines, robots, material handling equipment, and other automatic equipment. To integrate these machines under one control system is certainly not an easy job. A few companies have done this using a hierarchical control philosophy. The organizational structure involves partitioning the control problem into small, more manageable and less complex control modules and arranging them in a hierarchical fashion, so that the control system becomes easier to implement and modify.

Mesarovic, Macks, and Takahara (1976) prescribe the characteristics associated with a hierarchy. Each hierarchy is composed of a vertical

Table 1.1 Typical CIM functions [Feicholz, 1984].

BUSINESS PLANNING AND SUPPORT	MANUFACTURING CONTROL
Economics Simulation	Purchasing / Receiving
Long-term Forecasting	Shop Routing
Customer Order Servicing	Methods & Standards
Finished Goods Inventory	In-process Inventory Management
	Short-term Scheduling
ENGINEERING DESIGN	Shop Order Flow System
Computer-aided Drafting	SHOP FLOOR MONITORING
Computer-aided Tool Design	
Group Technology	Machine Performance Monitoring
CAD	Machine Load Monitoring
	Man-time Monitoring
MANUFACTURING PLANNING	Material Storage Monitoring
	Preventive Maintenance
Process Planning System	In-process Quality Testing
Parts Programming	
NC Graphics	PROCESS AUTOMATION
Tool & Material Catalog	
Material Requirements Planning	NC, DNC, CNC
Production Line Planning Simulation	Adaptive Control
Bill of Material Processors	Assembly Automation
Machinability Data System	Automatic Inspection
Computerized Cutter, Die Selection	Computerized Testing
Material / Parts Inventory Management	

arrangement of decision-making units. The higher level decision-making units have a right of intervention on the actions of the lower level decision-making units, but the performance of the higher level decision-making units depends on the actual performance of the lower levels.

The concept of hierarchical control is not new to those in industry. The organizational structures of most companies is organized hierarchically with the goals of each level contributing to meet the goals of the level above it. Not only can cases using hierarchical control be found in industries, but also in other areas such as retailing enterprises, local governments, etc.

1.2 Problem Statement

This research was directed at the control of a small-batch manufacturing system. This was accomplished through the design and implementation of a CIM control system which is also called the "shop controller." The shop controller has been developed to control the CIM facility at the Department of Industrial Engineering and Operations Research of Virginia Polytechnic Institute and State University. The CIM system consists of a machining cell, a material handling subsystem, and an assembly cell. This system has a network of computers to serve as control devices. The objective of the shop control system is to coordinate and supervise this complex system intelligently. The design of the shop control system is the main focus of this research.

The CIM system at Virginia Tech makes a family of products of similar parts and process operations. Orders are introduced into the system by demand. The first function included in the shop controller is customer order management, which takes new orders, releases orders into the manufacturing system, and keeps track of all orders in the system. The manufacturing process explosion follows the customer order management function and extends the ordered products from raw material processing through finished parts assembly. Next comes the routing problem. After orders are introduced to the production system, the material handling system moves the parts from the storage to the machining area. After being processed, parts are transported to the assembly cell. A database management module is included in the shop controller which manages the parts inventory and parts programs for all automated equipment. Additionally, the shop controller must be able to handle a set of typical malfunctions such as machine break-down or work-piece failure. Finally the shop controller can generate periodic production reports for management review.

1.3 Proposed Methodology

Because of the inflexibility in controlling production systems with the "conventional" methods, the shop control system in this research was developed using a new method called the three-layer integrated approach. This approach employs two basic techniques: expert systems and object-oriented programming.

The control problem of a production facility has two main parts: the regular service functions and the decision-making process. The main part of the decision-making process is scheduling and routing. It also performs some associated functions which includes in-process inventory management, machine performance monitoring, material handling monitoring, etc. It is shown that, by incorporating human decision-making capabilities into the system control, many tasks can be executed more easily and efficiently. Thus the decision-making process is to be implemented through a rule-based expert system. The expert system has a rule interpreter and three types of databases: the static knowledge base about the products and the processes, the dynamic database about the current state of the system, and the production rule base which is the logical part to perform actions.

The regular service functions include those which are a necessary part of the control system but do not have any direct influence on the decision-making process. Usually some business support functions and manufacturing planning functions are categorized in this group. These functions are incorporated into the control system through "objects." The objects are relatively independent function modules which are implemented in an object-oriented programming language.

2. Literature Review

2.1 General Control Methods for CIM

The control of a CIM system can be analyzed from various points of view. Most of the control mechanisms decompose the CIM into hierarchical levels [McLean, et al 1983]. McLean and his colleagues have designed a test-bed control hierarchy on the Automated Manufacturing Research Facility (AMRF) at the National Bureau of Standards and they are striving to develop standards in this area. The test-bed hierarchical control model is depicted in Figure 2.1. This test-bed control hierarchy is composed of five major levels: facility, shop, cell, work station, and equipment. Each level has one or more control modules that are further broken down into sub-levels or modules.

McLean, Mitchell, and Barkmeyer [1983] outline some of the control functions for each level. The facility level comprises three major subsystems: manufacturing engineering, information management, and production management. Manufacturing engineering provides user interfaces for the computer-aided design of parts, tools, and fixtures, as well as for the planning of production processes. Information management provides interfaces and supports for the administrative functions of cost and inventory accounting, order handing, and procurement. Production management tracks major projects, generates long-term schedules, and

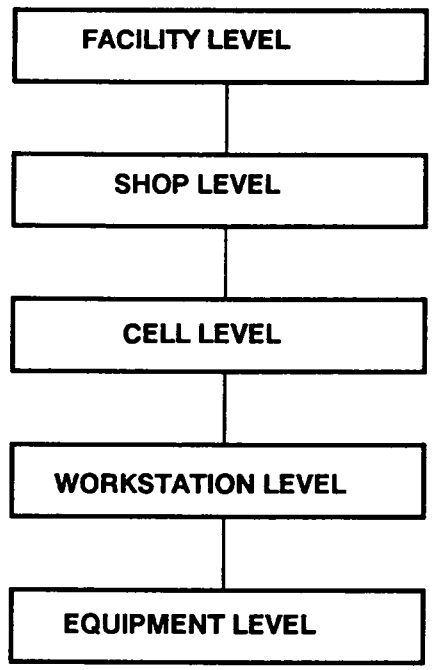

Figure 2.1 NBS hierarchical control architechure [McLean, at al 1983].

identifies production resource requirements and excess production capacity. The production planning data generated at this level are used to direct the shop control system at the next lower level.

The shop level is responsible for the real-time management of jobs and resources on the shop floor through two major modules: task management and resource requirement. The first schedules job orders, equipment maintenance, and shop support services. The latter allocates work stations, storage buffers, tools, and materials to all-level control systems and to particular production jobs.

The cell level controller manages the sequencing of batch jobs of similar parts or subassemblies, such as material handing or calibration. Models, within the cell, control the performance of system tasks, analyze availability of resources, report on job progress, route batches, schedule activities, requisition resources, and keep track of tasks being done at work stations.

The work station level controller directs and coordinates the grouping of equipment on the shop floor. It sequences the equipment through job setup, parts fixturing, cutting processes, in-process inspections, job take-down, and cleaning-up operations.

The equipment level controllers are tied directly to all automated pieces of equipment on the shop floor; be they robots, numerically controlled machines, tools, delivery systems, or various storage-retrieval devices. These systems perform the basic low-level functions of mate-

rial storage, transportation, material handling, material processing, cleaning, and inspection.

A similar approach can be seen in the document, "Reference Model of Production Systems" (1987), jointly prepared by Digital Equipment Corporation (DEC) and Philips. In this document, the NBS hierarchy is expanded to seven levels which is depicted in Figure 2.2. In this model, one more level, production facility, is added to the top of the factory level. This allows different factories on the same site to be under the domain of a dominant centralized controller. The DEC/Philips model divided the equipment level on the NBS model into two levels: Automation Module and Device. A device level controller commands actuators and receives status information from its sensors. The functions performed at the other levels are quite similar to those performed at similar levels in the NBS hierarchy.

It is important to know that the DEC/Philips model uses the concept of aggregation to describe the operation of the control hierarchy. Aggregation here means that a superior controller sees only its immediate subordinates, not the subordinates of its subordinate, and is only concerned with the subordinates fulfilling that command, not how the subordinate handles the execution. (This idea is quite similar to the basic idea used in object-oriented programming.) However, each controller also has access to global data, which may be of concern to each controller in the system.

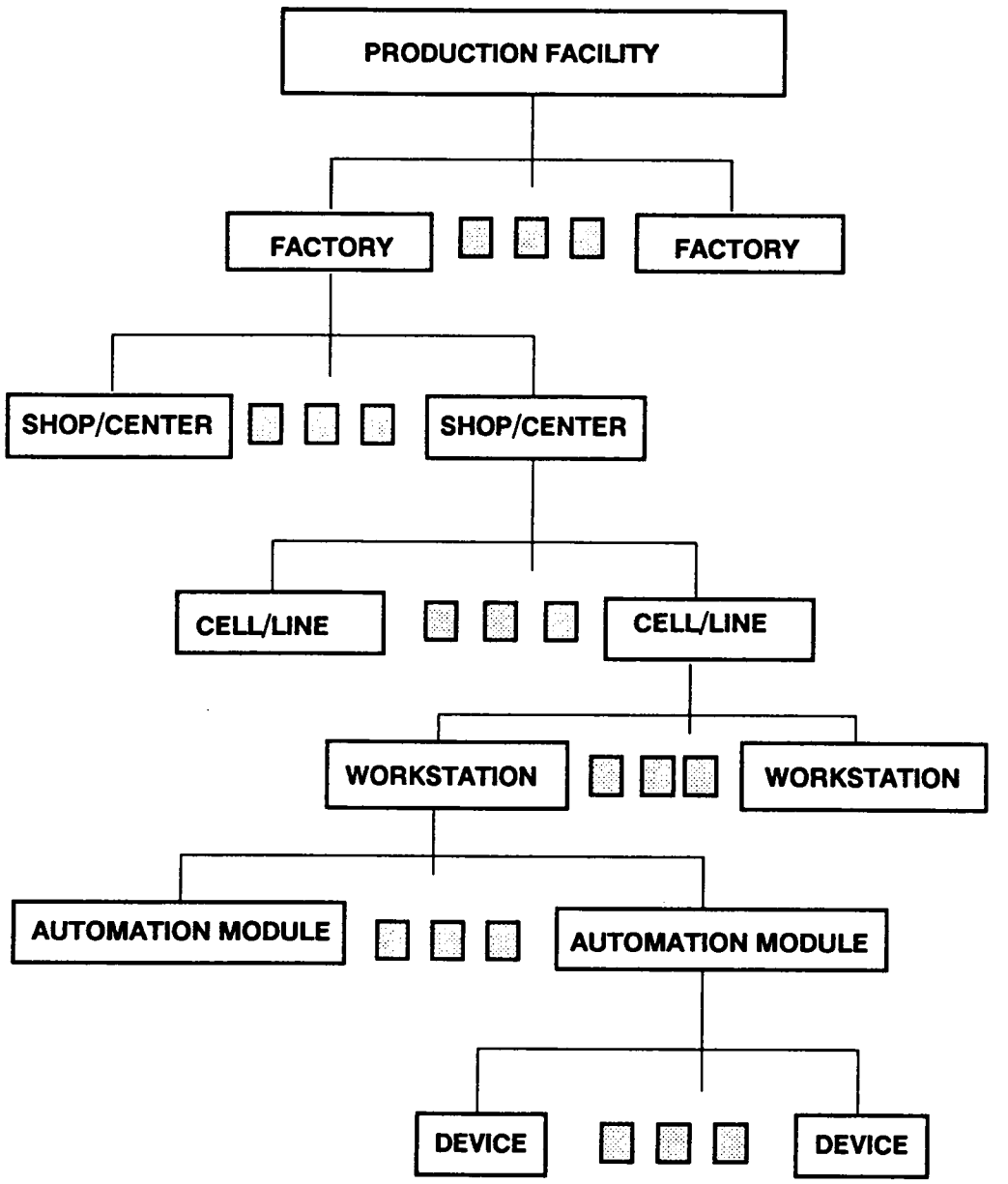

Figure 2.2 DEC/Philips model of production system [DEC/Philips 1987].

Kimesnia and Gershwin [1983] developed a different structure for the control system of CIM. In their work, a multi-level hierarchical control algorithm is proposed which involves a stochastic optimal control problem at the first level. The task performed at each level are not mentioned, but the lowest level deals with routing of the parts with considerations of resource availability and machine failure.

2.2 Shop and Cell Control Methods

One cell controller model can be found in the DEC/Philips "Reference Model of Production Systems" (1987). The general model of a controller is illustrated in Figure 2.3.

The DEC/Philips cell control model consists of three major functions: 1) goal or task decomposition (H); 2) sensory data processing (G); and 3) world model data representation, storage, and retrieval (M). The H function receives a command, for example, C2, to perform a certain task. This cell control function decomposes C2 into subcommands, C1O-C1N, based on the current state of the system under the domain of the controller. C2 has been fulfilled when all subcommands, C1O-C1N, have been executed by subordinate controllers. Status information concerning the execution of the subcommands, S1O-S1N, is returned to H. Sensory data, E1O-E1N, concerning the conditions of the environment under the domain of the controller is processed by G. Relevant information is passed to H to aid in decision making.

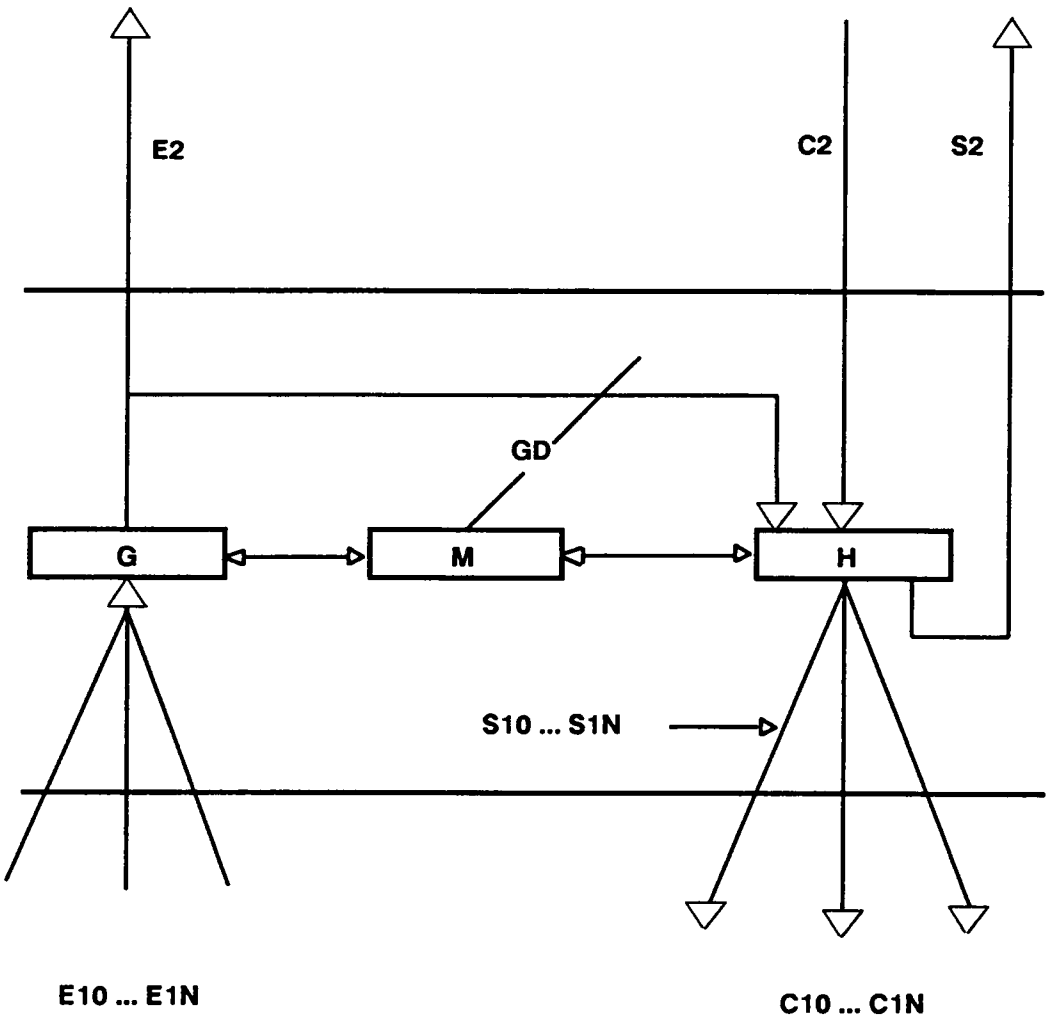

Figure 2.3 DEC/Philips general controller model [DEC/Philips 1987].

The M function is the controller's knowledge of its environment. It contains such information as the capabilities of subordinate controllers. It does not, however, concern itself with the actual workings of the subordinates. M has access to both local data and global data (GD). GD is information about parts of the system which are outside the domain of a particular controller but which may influence the decision making in some way.

The "Reference Model of Production Systems" stresses that the H, G, and M components of the general model are abstract concepts and do not necessarily represent three separate physical systems. In fact, the implementation of the H, G, and M components at one level of the hierarchy may differ from implementation of the H, G, and M components at another level [DEC/Philips, 1987, Dutton, 1987].

Another approach for the cell control system can be found in the research on AMRF at National Bureau of Standards [Albus, McLean, Barbera, and Fitzgerald. 1982]. In this research, state tables were utilized for the control of a manufacturing cell at the AMRF. An example of a portion of the Material Handling System state table is provided in Figure 2.4.

The left side of the state table is the input table which consists of a list of internal states, all commands and feed back inputs that are possible at the given level. The right side of the state table is the action table which specifies the appropriate action to be taken by the controller at a particular given state. At each clock cycle, all input

Internal States						Actions		
CURS	MHS_DISP_CMD	MHS_OPER_S	MHS_TASK_S	MEG_ERR	DB_WAIT	PROCEDURE	MHS_DIPS_S	NEXTS
INIT	*	*	*	*	FALSE	SHUTDOWN	NOT READY	NOT READY
NOTREADY	END DISP	*	*	OK	FALSE	SHUTDOWN	NOT READY	NOT READY
NOT READY	BEG DISP	OFF	*	OK	FALSE	STARTUP	NOT READY	NOT READY
NOT READY	BEG DISP	LOCAL	*	OK	FALSE	STARTUP	NOT READY	NOT READY
NOT READY	BEG DISP	REMOTE	NOT READY	OK	FALSE	STARTUP	NOT READY	NOT READY
NOT READY	BEG DISP	REMOTE	READY	OK	FALSE	STARTUP	NOTREADY	READY
READY	BEG DISP	REMOTE	READY	OK	FALSE	STARTUP	READY	READY
READY	HANDREG	REMOTE	READY	OK	FALSE		READY	MONITOR
READY	WAIT	REMOTE	READY	OK	FALSE			READY
READY	ENDDISP	*	*	OK	FALSE	SHUTDOWN	READY	NOTREADY
MONITOR	*	REMOTE	READY	OK	FALSE	REPLENIN	MONITOR	MONITOR
MONITOR	*	REMOTE	KITTING	OK	FALSE		MONITOR	MONITOR
MONITOR	*	REMOTE	PICKUP	OK	FALSE		MONITOR	MONITOR

Figure 2.4 A state table used in material handling system at NBS [Albus, at al 1982]

values are read from common memory, the matching line in the table is determined and the corresponding action taken.

As described in the paper written by Jones and McLean [1984], the state table approach has several advantages. First, the system can be decomposed into well-defined modules with clearly specified inputs, outputs, internal states, and rules for state transitions. Second, additional internal states can be defined to handle recoverable error conditions, new feedback data, or commands, and the corresponding lines inserted into the table. Third, since it clearly defines the problem in an orderly structure, new states and actions can be incorporated easily. However, the utilization of state tables for control becomes clumsy as control problem gets more complex, and the number of values that each variable can take increases.

Dutton [1987] designed and implemented a cell control system using DEC/Philips philosophy. By specifying and examining the functions required for system operations, a general model of a cell controller is developed. First, all the functions that are necessary for the management of a manufacturing system are reviewed and grouped under four categories: Information Management, Equipment Management, Production Management, and Quality Management. Second, the overall system functions are decomposed as the tasks move down the hierarchy from the cell level controller to the individual devices in the facility.

Another model for system control was suggested by Bertrand [1984]. He sought to expand the MRP-II philosophy of production control system

design by developing a methodology for designing hierarchically structured control systems. He divided production control problem into three hierarchically arranged sub-problems:

1. Periodically generate a Master Plan based on the actual state and predicted future state of the system.
2. During a period, generate inputs for the various production units which will coordinate in the fulfillment of the Master Plan.
3. Control the actual inputs to the production units, based on the state of each production unit, the coordination inputs, and the requirement of the systems.

Arai, Hata, Imakubo, and Kikachi [1982] suggest a standard two-level production control system. The lower level controller is responsible for the actions of up to four NC machines. The upper level controller contains all production information and transmits the required NC programs to the lower level controller. Eight functions are performed by this system: data management, schedule running, machine control, production record gathering, interactive quality control, machine monitoring, interactive failure diagnosis, and material handling control.

2.3 Expert Systems Review

Expert systems is a research field in Artificial Intelligence. Expert systems, often called rule-based systems, are programs for performing tasks requiring expertise but no great insight. The objective

of expert systems is to do jobs in a domain where it is hard to break the job down into an algorithmic procedure.

There are four phrases in expert systems design:

1. Knowledge acquisition.
2. Rule set development.
3. Prototype implementation.
4. Refinement.

The nature of rule-based systems affects all stages of the process and places fundamental limitations on the control programs. It is important to notice the last stage, Refinement. In this case, refinement does not necessarily mean making code changes to make the program run more efficiently. It means coming back to the experts for criticism and extension to the problem, until it begins to solve a significant fraction of problems in the domain. A program like this must be open-ended; it is never finished. There are three fundamental structures in an expert system:

1. A rule base is a set of rules used to determine the system behavior. These rules constitute the "expert knowledge" of the system.
2. Working memory is the variable store in which the particular problem situation (as opposed to knowledge of the problem domain) resides.
3. The rule interpreter is the procedure to determine which rule to apply. This procedure determines which rules have their conditions satisfied and triggers them. If more than one rule is trig-

gered, the inference engine resolves conflicts to select a rule for activation.

2.3.1 Knowledge Representations

The knowledge representation is the crucial part in expert systems implementation. The representation can be embodied in a variety of different data structures, to make different operations efficient. There are many different ways knowledge can be represented. Examples of currently-used methods of knowledge representation are:

1. Rule-based Representations: This method has two main parts: pattern-action rules and working memory. It is efficient but brittle, and unmodular.
2. Graph-based Representations: In this category, there are several different approaches depending on the problem, like game tree, AND/OR tree, associative networks, inheritance hierarchies, etc.
3. Numeric Modules: This method implies representing the knowledge with numerical values, and is ideal for low level image processing, and speech recognition.
4. Language/Logic Based Systems: In this approach, the propositional and predicate calculus are the representations of the process of inferring new information from existing facts. Shortcoming with this approach is that the knowledge will change as time goes on.
5. Frame-based Systems: A frame in this system is an object template which stands for each basic kind of object in the system. Each frame has several named slots which receive attribute values.

6. Hybrid Systems: This method uses two or more of the other standard representation schemes. Most often the knowledge system is segregated into distinct knowledge subsets. It attempts to exploit the advantages of different representational schemes for different aspects of the problem at hand.

2.3.2 Rule Interpreter

The rule interpreter is the control procedure in the expert systems. The strategy used by the control procedure is called the "control strategy." Several strategies have been used in various expert systems. Two commonly-used ones are data-driven control and goal-driven control. In the first one, the conditional parts of the set of rules are examined to determine which rules are satisfied by the contents of the working memory. In goal-driven control, it starts with examining the rules to determine if a desired rule can be matched by the current contents of the working memory.

By choosing the rule to be activated on the basis of the contents of the working memory, a complete re-evaluation of the state of the system and the corresponding updating of the working memory are performed every computational cycle. So the rule-based system is sensitive to the changes in the environment and responds to these changes in a single cycle [Fischler and Firschein, 1987].

2.3.3 Conflict Resolution Strategies

As the rule interpreter goes through all the rules, it may find several rules have their conditions satisfied. Therefore, conflict resolution must be incorporated into the system to select a particular rule to fire (to perform the action). A list of strategies for the conflict resolution is shown here.

1. Specificity: If the conditions of R1 contain the conditions of R2, use R1.
2. Rule ordering: Fire the one which occurs earliest in the rule base.
3. Data ordering: Compute priorities for competing rules based on priorities for their conditions.
4. Size ordering: Use the rule with the longest list of restraining conditions (toughest rule to match).
5. Recency ordering: Use rule most (or least) recently used.
6. Context limiting: Separate the rules into groups and provide a procedure for activating and deactivating groups.

2.3.4 Expert Systems for CIM Control

In the CIM environment, the expert system can be incorporated into system software to provide expertise for managing the system. The main function of the expert system is trying to simulate an expert shop floor manager (or several managers), who knows from his/her experience how the machines and jobs interact.

Limited research has been done on using expert systems in shop control software. In the scheduling domain, an example on using expert systems in a job-shop environment is ISIS [Fox, 1982, 1983]. This system which is written in SRL (a knowledge representation language) uses a heuristic search approach in a constraint environment to achieve its objective. The system is used to schedule orders that arrive at the Westinghouse Electric's Turbine Component Plant. The objective is to meet due dates while satisfying the constraints in the plant. The constraints are divided into three main groups. The first group has "organizational goals" like in-process inventory, resources level, production level and shop stability. The second group contains physical constraints (ability of machines, etc.). And in the third group, there are precedence relations and resources requirements. The scheduling decisions are made on the basis of current and future costs, and profits.

Another example on using expert systems for shop floor scheduling is "Knowledge Based Control System (KBRS)" developed by Ben-Arich [1985]. Its main aim is to route jobs in an automated production and assembly system. It is written in Prolog and implemented in simulated environment. The simulated manufacturing system is composed of five computerized manufacturing cells and an automated assembly station. The assumptions are that each machine has a small buffer and parts operations can be done on any one of the five machines. This model is very limited in the ability to control a real-time CIM system. Some of the assumptions like a given machine failure rate or parts programs changing instantly is no longer valid in the real environment. In a real-time CIM system,

information is updated every clock cycle, the dynamic environment would make the static expert system (KBRS) lose its validity.

2.4 Object-Oriented Programming

Object-oriented problem solving and object-oriented programming represent a way of thinking and a methodology for computer programming that are quite different from the usual approaches supported by structured programming languages [Weiner, Pinston 1988]. Object-oriented programming is a relatively new method for designing and implementing software systems. In the past twenty years, computer hardware has undergone fundamental revolutions that repeatedly changed its entire way of thinking. The earlier engineers developed computers using discrete components such as capacitors, resistors, and transistors. This was quickly replaced by board-level integration, and then integrated circuits, LSI, VLSI. This change resulted in great improvements in hardware capacity.

In recent years, some researchers have been seeking similar improvement in computer softwares. Their main aim is to improve software reusability and extensibility. One result is the object-oriented programming. The object-oriented programming defines the unit of modularity so that programmers produce sub-components, instead of complete solutions. The sub-components are controlled by standards and can be interchanged across different products. Then software engineers build the desired software system by assembling software components of other programmers instead of developing the entire software from basic statements and expressions of a programming language.

2.4.1 Characteristics of Object-Oriented Programming

As discussed earlier, object-oriented programming is a relatively new way of computer programming. It has brought along some new tools and concepts that can help to develop software.

1. Abstract data type, class, and object: Abstract data type is a module that encompasses a data type and an associated set of operations. These operations are defined for and characterize the behavior of the underlying type. Weiner and Pinston [1988] give a detail discussion on these concepts.

   A class definition describes the behavior of the underlying abstract data type by defining the interface to all the operations that can be performed on the underlying type. Each operation is implemented as a method. The class definition also specifies the implementation detail or data structures of the type.

   An object is a variable declared to be of a specific class. Such an object encapsulates state by duplicating all the fields of data that are defined in the class definition. Actions may be performed on such an object by invoking one or more methods defined in the class definition. The process of invoking a method is called sending a message to the object.

2. Encapsulation: Encapsulation is the foundation of the whole approach. It defines a complete boundary that encompasses the object's internal data and procedures so as to restrict the effect of changes. All access to internal data is handled by methods defined in the object. Encapsulation also gives interface which shows how the underlying data object interface with other objects or other programs. With conventional programming, programmers specify how operations should be performed by naming a piece of data, while with object-oriented programming, they only have to specify what is to be done and are no longer concern about how operations are done.

3. Inheritance: Inheritance is the more innovative part of the approach because it is not provided by conventional languages. In object-oriented programming environment, some classes can be declared to be subordinate to other class (or parent class). Subclasses are considered special cases of the parent class and have some additional features.

4. Polymorphism: Polymorphism is a key feature of object-oriented programming. As discussed before, in order to make an object to perform an action, message must be passed to the object. In most object-oriented languages, a common set of messages can be sent to the objects of both parent class and subclasses; this is called polymorphism. Polymorphism allows each object to respond to a common message in manner appropriate to the class from which the object is taken.

2.5 C++ and Object-Oriented Programming

C++ is a superset of the C language developed at Bell Laboratories. C++ retains C's power and flexibility to deal with hardware-software interface and low-level system programming. Its significant enhancement is the support of the object-oriented paradigm.

C++ was developed by Bjarne Stroustrup in the early 1980s. Dr. Stroustrup developed C++ to support the writing of some complex event-driven simulations for which considerations of efficiency precluded the use of Simula67.

Some important aspects of C++ language are listed below:

1. Retaining the extremely high machine efficiency and portability of the C language.
2. Retaining compatible with C. This preserves integrity of millions of lines of C code, the expensive C libraries, and tools that have been developed.
3. Providing increased type checking, compared to C. This has repaired the C language's long-standing flaw, the weak type checking when using functions.
4. Updating C to support object-oriented paradigm.

3. System Control Structure

Managing and controlling a computer integrated manufacturing (CIM) facility which is composed of machining, assembly, and material handling systems is a complicated problem. The main issue is the coordination of all sub-systems within the facility. The production environment is complex and dynamic, hence a short response time is required. For a particular manufacturing shop, with the capabilities of all machines or production cells and the product structures being given, a high performance of the system is of course desirable. Furthermore, business functions and human operator interface must be addressed in the control model. Therefore, these requirements result in the need of a system control software which is intelligent, fast, and adaptive in order to deal with all possible situations that may occur in the system.

In this chapter, the Computer Integrated Manufacturing facility is described. Next, the functions which need to be included in the control model are analyzed. Then, the design of the whole system control hierarchy is discussed, and the communications devices used for inter-level communication are described. Finally, using a three-layer integrated methodology, a general control structure of the shop controller is presented.

3.1 VPI CIM System Description

A CIM facility is being created to support instruction and research in the integration and control aspects of computer-based manufacturing technology. This facility is currently located at the CIM Laboratory in Whittemore Hall at Virginia Tech and consists of three workcells. A schematic description of the CIM facility is illustrated in Figure 3.1.

The first workcell is a machining center which has two CNC milling machines, a robot and its controller, a local storage buffer, and an AT&T 6300 PC which serves as its workcell controller. The main function of this cell is to perform all machining of individual work pieces. The second workcell has an IBM robot, a local storage buffer, a few fixtures, and an AT&T 6300 PC to serve as the workcell controller. It performs two system functions. First it places all the parts that are necessary to make an individual product on an empty pallet. This is referred to as "kitting." Additionally this workcell is used for assembling the parts after they have been machined into a finished product. The AT&T personal computer serves to coordinate and control all the work within the workcell. The last workcell is a material storage and handling sub-system which consists of an automatic storage and retrieval system (AS/RS), an automatic conveyor system, a machine vision system and an AT&T 6300 PC workcell controller. A shop system host computer is connected to the three cell computers through a local area network (LAN) to coordinate and control the whole system. The shop level is the highest level of control considered in this research.

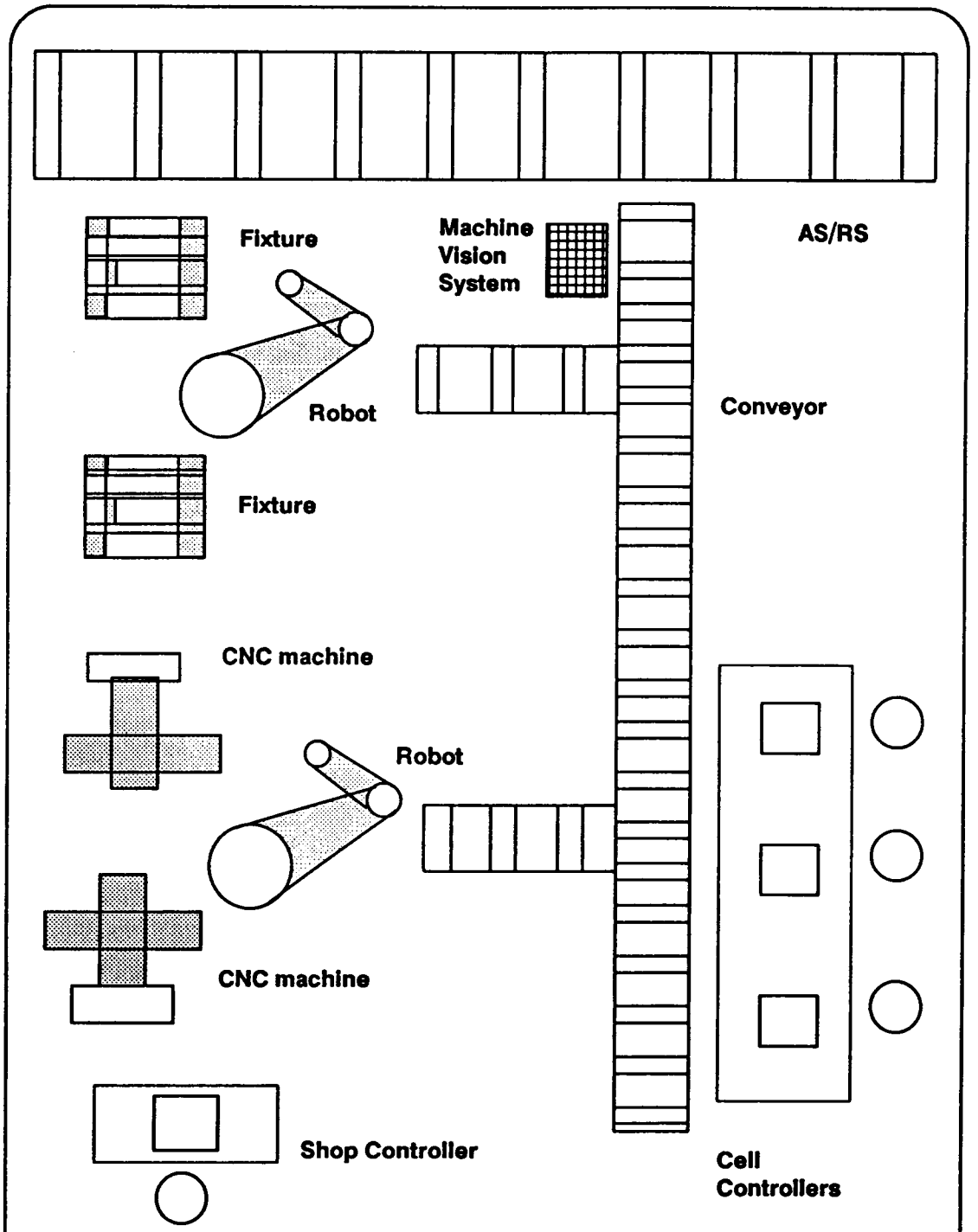

Figure 3.1 A schematic description of the CIM facility at Virginia Tech.

3.2 Shop Control System Functions

Several functions are required to manage a manufacturing system. Teicholz [1984] presents a table which lists all typical functions that are required to control a CIM system. These functions were illustrated in Table 1.1. He groups all functions into six categories: business planning, engineering design, manufacturing planning, manufacturing control, shop floor monitoring, and process automation. For a particular case, it is not necessary or feasible to include all functions listed in Table 1.1. Different manufacturing systems require a different set of functions depending on the system configuration and the product structures. Alternative control models may arrange the set of functions in a different way depending on the methodology used. Since the CIM facility at VPI is just a make-to-order production model and not a full-blown manufacturing system, only functions which are necessary for the system are included. The functions selected for the VPI CIM system is shown in Table 3.1. These functions can be grouped into three categories: business planning and support, manufacturing planning, and production control.

3.2.1 Business Planning and Support

The business functions are those which handle business procedures beyond the shop floor and provide information for the production shop. In this control system only two functions are supported: customer order management and production reporting.

Table 3.1 Function list of the shop controller.

Business Planning and Support.

- **Customer order management.**
- **Production reporting.**

Manufacturing Planning.

- **Parts programming management.**
- **Bill of material processors.**
- **Machinability data system.**

Production Control.

- **In-process inventory management.**
- **Shop scheduling and routing.**
- **Shop order flow system.**
- **Machine performance monitoring.**
- **Material storage monitoring (AS/RS).**
- **Machine vision testing.**

Customer order management is the first function of the CIM system and its main responsibility is to process customer orders. Only after orders come into the system does the production system begin working towards the fulfillment of the orders. This function maintains a prioritized list of all active customer orders for input to the production control functions.

The production reporting function maintains records of all orders produced and generates a production report periodically. This function also provides information for order tracking.

3.2.2 Manufacturing Planning

Manufacturing planning functions provide the basic information and support for the production system. There are three manufacturing planning functions: parts program management, bill of material preprocessors, and machinability data system.

In a fully automatic manufacturing system, no human being is involved at the lower levels of the production system. Hence the controllers at the shop and cell levels have the responsibility of maintaining and manipulating all parts programs for the various machines. This responsibility includes maintaining a database of all parts programs, renewing old programs, and downloading programs to the machines as needed and is referred to as the parts program management function.

The bill of material processors function provides all information related to the production of each part. Basically this function specifies the type of materials and quantities for the production of each product. This information is also needed to determine whether the system has enough raw materials to produce an order.

The machinability data system function is a necessary part of the control system of a manufacturing system. As for the shop controller, this function provides information about the capabilities of each cell. This information is necessary for determining the production routing.

3.2.3 Production Control

The production control functions include those which monitor all machine performance on the shop floor and manage the material flow in the process. This category has six functions: in-process inventory management, shop scheduling and routing, shop order flow system, machine performance monitoring, material storage monitoring, and machine vision testing.

The in-process inventory management function is required to keep track of all the parts which are either being processed or in the storage sub-system. This function also provides information for accessing inventory cost and determining the part flow route.

To produce a product, the raw material must visit several cells for a series of processes. So the control system must select a route

through the system for each part. The route selection function is referred to as scheduling and routing. This function is the most difficult part of the production management. And it is where expertise is required.

The shop order flow system function is the main force that drives the production system. "No order" means no production. The order flow system is a necessary part of the production control system. This function serves as a buffer between the customer order queue and the finished goods inventory list.

Information about the status of the equipment must be known in order to operate a production system. The machine performance monitoring function monitors machine operations and provides information for production scheduling. This function also provides information for evaluating machine utilization and failure rate.

The material storage monitoring function handles the Automatic Storage and Retrieval System (AS/RS). Usually raw material, unfinished parts, and finished goods are stored on the storage subsystem. Thus the position of each storage cell on the rack and the part type in all occupied storage cells must be kept in memory. In this production system, parts are carried by pallets. When a pallet comes in, this function finds an appropriate empty storage cell for that pallet; when a pallet needs to be released, it finds the correct storage cell which holds the required pallet.

In the CIM system, whenever the control system removes a pallet from the storage, it is necessary to make sure that the pallet contains the parts that are expected and that they are in the right location. During the transportation process, the positions of the parts on the pallet may shift and this may cause a problem for the next cell. The machine vision system can be used to prevent this kind of problem. This function will be implemented in the material handling sub-system.

3.3 System Control Structure

The hierarchical approach is used to design the system control structure. The architecture of the control hierarchy is illustrated in Figure 3.2. As can be seen in the "Literature Review," a hierarchical approach is a commonly used method in organizing and controlling CIM systems. The hierarchical philosophy is nothing new. Before the advent of the computer age, most industrial companies were organized hierarchically. Since the hierarchical approach appears to be a natural philosophy to follow, it is still widely used.

In this model, the shop controller is the highest control level. The cell level consists of five cell controllers. The first machining cell and the assembly cell are presently being implemented. The second machining cell and the inspection cell are planned for the future. The last cell is a material handling sub-system which links the other workcells into an integrated production system.

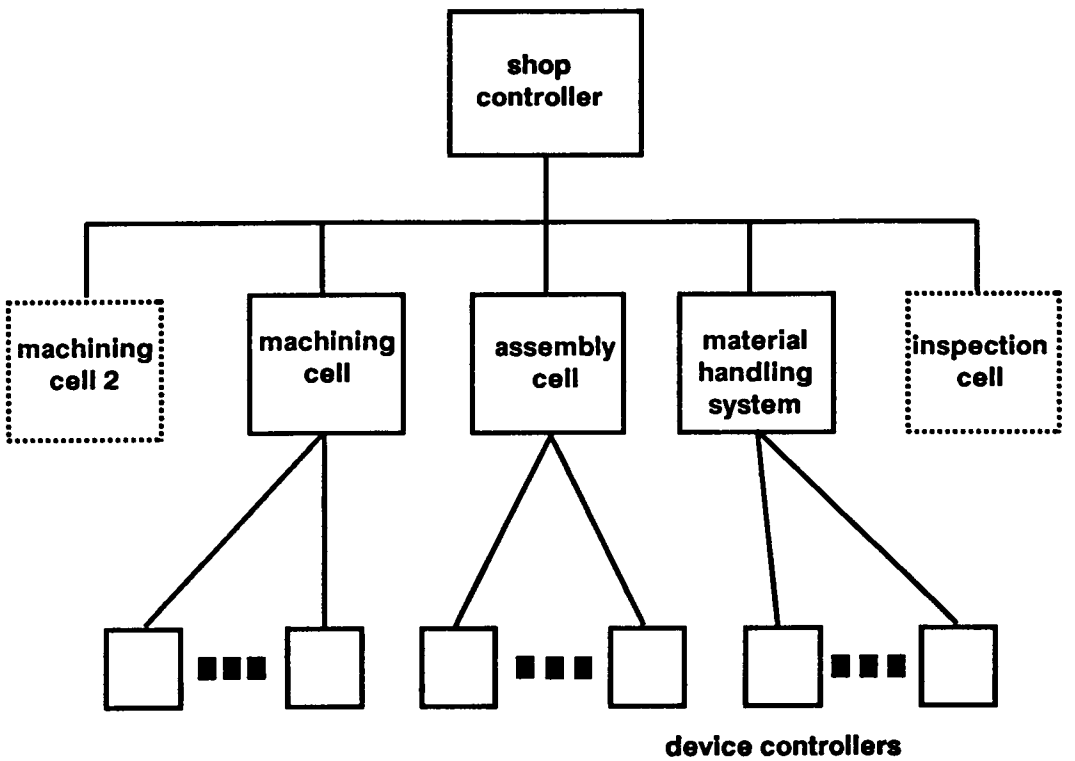

Figure 3.2 The control structure of the CIM facility.

The shop control system and the three cell control systems are implemented on four AT&T 6300 computers, each with a 640-kbyte memory and a 20-megabyte disk drive. The communication between the shop controller and the cell controllers is carried on through an AT&T STARLAN network with 1-megabit transmission speed. An AT&T Unix PC is used to serve as the network host and the file server. For the communication between the cell controllers and the devices, several kinds of communication devices are used. To communicate, the machining cell controller uses a built-in serial port for the robot, a four-port serial card for the two CNC milling machines, and a parallel I/O card for discrete input/output to both the robot and the two CNC milling machines. In the assembly cell, the built-in serial port and a parallel I/O card are used for communication with the robot. To communicate, the material handling controller uses the built-in serial port for the programmable controller, a second serial port for the machine vision system, and a 64-pin I/O parallel card for the programmable controller and the machine vision system.

3.4 Shop Control Structure

3.4.1 Description of the Three-Layer Integrated Method

As discussed earlier, the basic requirements for managing an automatic manufacturing facility are intelligence, high efficiency, and flexibility. A three-layer integrated approach was employed in order to satisfy these requirements. This approach can be used to design the control system for any level such as the shop level or the cell level

in the system control hierarchy. Figure 3.3 gives a graphical illustration of this approach.

As can be seen in Figure 3.3, the system has three concentric layers. The innermost layer performs the decision-making functions and is called "Decision Layer". An expert system is used in this layer. The objective of the expert system is to use all available data from the automatic production facility (environmental knowledge) and human expertise and experience to create an intelligent control mechanism. The intelligent control mechanism manages the production system and generates real-time responses to problems that may occur in the production system.

The middle part is the "Action Layer" which is composed of several "action objects." The algorithmic knowledge and part of the static database reside in this layer. This layer is responsible for generating commands for the production system and for interacting with the controllers of the next level to carry out the commands. This layer also gathers information necessary for the expert system to make decisions from the controllers or machines of the next lower level.

The "Service Layer" is the outermost part of the control system. This layer has some generic objects which help the action objects in the middle layer to do some calculations and operations. The term "generic" means that each service object is used by several action objects. The service objects usually don't interact directly with the decision layer.

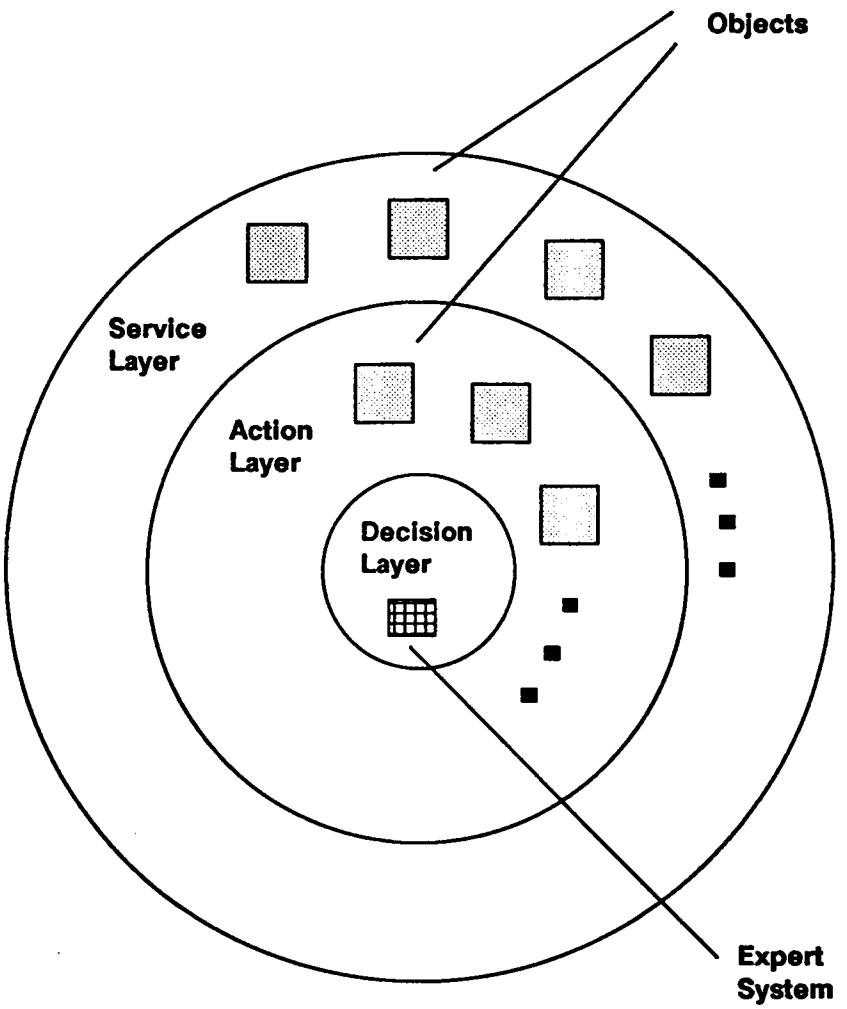

Figure 3.3 Graphical illustration of the three-layer integrated method.

3.4.2 The Conceptual Design of the Shop Control System

To design the shop control system for the CIM facility, the three-layer integrated approach was used. First the functions discussed in Section 3.2, the communication functions, and other related functions were analyzed. Those functions which were associated with the decision-making process were incorporated into the expert system, and others were implemented as independent objects. Then the conceptual form of the shop control structure was developed and is given in Figure 3.4.

In this shop control system, the central layer consists of a main driver and an expert system. The function of the main driver is to initialize the control system. The expert system performs the following functions: shop scheduling and routing, bill of material processors, machinability data system, shop order flow system, machine performance monitoring, in-process inventory management, and material storage monitoring.

Using object-oriented problem solving methodology to solve a problem requires that the problem space be mapped onto a set of objects and modules in the solutions space. The problem space for the shop controller includes the functions discussed in section 3.2 and the functions which handle the interfaces between the shop controller and cell controllers and between the shop controller and the operator. Since some of the production control functions from Table 3.1 are incorporated into the expert system, the problem space consists of the production

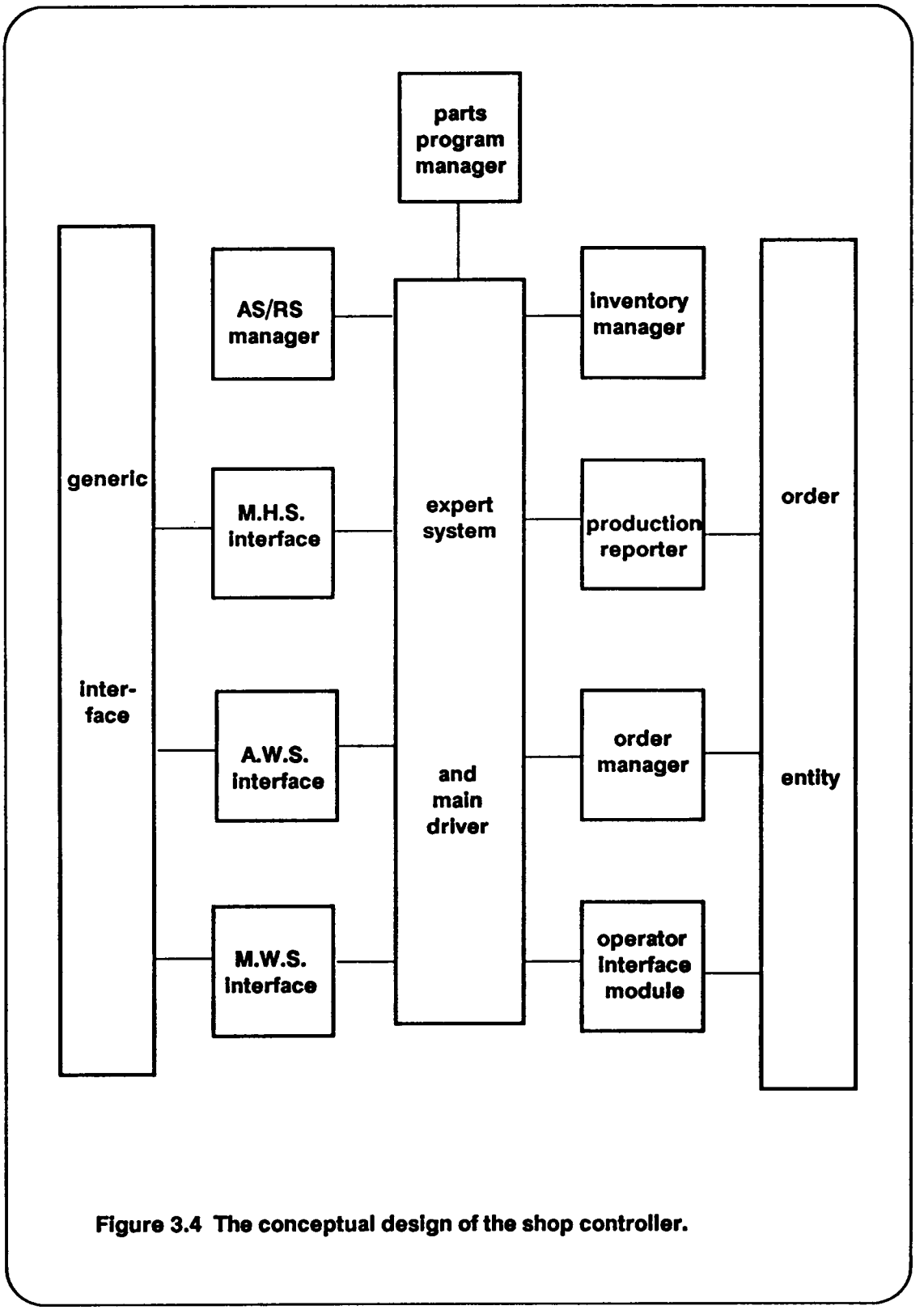

Figure 3.4 The conceptual design of the shop controller.

functions from Table 3.1 which were not included in the expert system, the four interface functions, and the AS/RS management function. The system objects are the following:

1. inventory management;
2. order management;
3. production report;
4. parts program management;
5. AS/RS management;
6. interface to the machining cell (MWS);
7. interface to the assembly cell (AWS);
8. interface to the material handling system (MHS); and
9. operator interface.

Each entity, except the operator interface, is represented by a class as defined earlier. The operator interface was implemented as an independent module and was coded in a functional form. All the other eight major entities are transformed into objects which constitute the middle layer.

Only two major entities, order form and generic interface function, can be put into the outer layer. Two classes are declared for these two entities, and the objects of these two classes compose the outer layer.

4. Lower Level Design of the Shop Controller

The control of a CIM system is a complicated problem. This problem has a dynamic nature to which there is no closed form analytical solution. Besides the system dynamics, other important issues to be addressed in the system control include the objective function, state and control variables and constraints. All these issues lead to a control system which is intelligent, adaptive, and able to manage all foreseeable events. Since the system works in a real-time mode, it must be fast to respond to the changes of the system.

4.1 Some Important Issues in the Shop Control System Design

The shop level control is designed to ensure that the operation of the CIM facility performs as desired. The primary objective of the control system is to meet the predefined master production schedule. Several important issues of the control problem have to be analyzed before the design of the control system. First, the allocation of operations to the system resources must be made in an efficient way. These resources include the machining cell, the assembly cell, the material handling system, and storage space. To assign tasks to machines requires careful consideration. For example, assume that the machining cell and the assembly cell are both ready. The assembly cell can do one of the two tasks: make a product kit for machining or

assemble the parts of a machined kit. The machining cell has nothing to work on. The assembly cell should make the product kit before the assembly is done because the machining cell would have more idle time if the assembly cell does the assembly first.

Reducing setup time is also a crucial element. After a cell completes a set of similar parts, it is often necessary to change the cell configuration somewhat to work on another set of parts. This changing period is setup time. It is beneficial to limit the system changes while still meeting the specified production requirements.

Another important function of the shop control system is to limit the effect of interruptions to the system operation. Interruptions are due to machine failure, material failure, and other elements. The effect of an interruption should be localized so that it has a minimum effect on the availability of other system resources.

4.2 Decision-Making Part----Expert System

The main function of the shop control system is to schedule and route parts through the production system according to the available information about the state of the system. This part was implemented through an expert system. The expert system was meant to put human expertise into a series of production rules, then to utilize these rules and knowledge of the state of the system to make decisions. This kind of expert system is often called a rule-based system. Basically this expert system has three main parts:

1. working memory;
2. production rules;
3. rule interpreter;

A rule-based system has several advantages. First, it is easy to understand, second, it is uniform in structure, and third, it is efficient.

4.2.1 Knowledge Representation and Knowledge Acquisition

A knowledge representation is a stylized version of human expertise. The representation has a great impact on operation efficiency. In this real-time control system, time efficiency cannot be overlooked. Therefore, all knowledge was represented in a simple but efficient IF-THEN form, using C language. In this system, knowledge acquisition has three phases:

1. Rule set development. This is the first step in building an expert system. At this stage, Dr. Deisenroth was consulted. Then his expertise was translated into the IF-THEN conditional form which will be the initial rule set.
2. Prototype implementation. After the initial system was built, several trial runs were made in a simulated environment. Some suboptimal functioning were observed. More knowledge was required to improve the performance.
3. Refinement. Some original rules were changed and some new rules were added to the knowledge base. Several refinement cycles were made before the expert system performed properly.

As an example, it can be seen that changing one rule can change the system behavior significantly. At first trial run, it was observed that the system did not shift the finished products to the operator until there was no customer order in the queue. The AS/RS put all pallets (carriers for parts and products) at the first available position near the entrance point, resulting in a large accumulation of finished goods at the entrance point. The efficiency of the AS/RS was affected. Later a new rule was added which stated that after an order had been processed, it would release all the products of that order to the operator. This change solved the problem.

4.2.2 Production Rules

The production rules are the system representation of the human expertise. The rule base is represented in a simple IF-THEN form, and it is written in C language. All rules in the knowledge base can be put into three main groups: operator interface rules, system maintenance rules, and system operation rules.

Operator interface rules relate to the events which require an operator interface. Whenever an action requires an operator interface, one of these rules will be fired, and the control system is put into a waiting state. These rules can also be called interrupt rules.

Example

IF loading request;
conveyor ready;
THEN load a pallet into the system.

The representation of the rule in C language:

```
if (manreq==2 && mhswm==1)
rawmat();
```

System maintenance rules are used to maintain the control system. For example, the system has to save status periodically to prevent accidental power off. A product report should be generated at the end of the day. If the order buffer is empty, the control system may go to the order queue to get an order. The rules in this group and the group above have one thing in common. If one of these rules is fired, the system does not end the control cycle and it continues to search for matching rules in the third group.

Example

IF order buffer empty.

THEN get order;
check product type;
initialize order quantity.

System operation rules are associated with machine operations like milling or assembling on shop floor. These rules are the part of the knowledge base which do the scheduling and routing.

Example

IF product type robot;
machining cell ready;
local storage no pallet;
AS/RS robot kit;
MHS ready;

```
THEN        find robot kit position;
            MHS as/rs to machining cell;
            set MHS busy;
            subtract robot kit number.
```

4.2.3 Working Memory

Working memory is the part of the expert system which represents a particular problem situation (system state). This part provides information about machine availability, local storage, AS/RS status, and order queue and is represented by integers and integer arrays.

Example

```
//Working cell status.
int mhswm, mwswm, awswm;
//Pallet at local storage.
int mpkc, mpkr;
int apmr, apmc, apbr, apbc, apbl, apem;
```

4.2.4 Rule Interpreter

In this system, the rule interpreter has three operations to perform. First, it interprets the feedback messages from cells, and updates the working memory. Second, it selects rules to apply by the matching technique. If several rules have their conditions satisfied, these rules are all selected. Third, if more than one rule is selected, the rule interpreter resolves conflicts by selecting one rule for activation. Two strategies are used to resolve conflicts. Rule

ordering is the strategy in which the rule that occurs earliest in the rule base has highest priority. The other is context limiting in which the rules are put into groups and a procedure for activating and deactivating groups is provided.

4.3 Action Objects in the Middle Layer

The middle layer of the shop controller is the action layer which contains objects which serve the decision layer directly. The objects in the middle layer provide interface between the expert system and the shop floor. And they also do bookkeeping and computations. If algorithmic knowledge is added to this system, it can be translated into objects which reside in this layer. These action objects can use objects in the outer layer to do some specific operations. Each class will be described separately in the following section.

4.3.1 Class Inventory

The inventory class contains all the information about raw materials. This includes component type, quantity, minimum ordering level, and maximum storage level.

The class inventory has the following private data:

1. An array of component nodes. Each node contains information about component type, quantity, and minimum order level.
2. An integer pointer. This pointer points to the current position on the array.

The class inventory has the following methods:

1. A constructor which sets the pointer to the first element of the array.
2. void replenish(char * name, int num) - used when adding components to the storage. The first parameter is the type name which serves as an identifier, the second is the quantity of the components.
3. void insert(char * name, int q, int o) - used when putting a new kind of component into storage. The first parameter is the component name, the second is the initial quantity, and the third is the minimum order level for the component.
4. int withdraw(char * name, int mum) - used when releasing some components into the production system. The function returns 1 if the component left in storage has fallen below the order level.
5. int check(char * name) - used to check how many components are left in storage. It returns the number left.

This informal description of class inventory is translated into a formal class definition which is shown below.

```
class inventory
{
      private:
            component storage[50];
            int last;
      public:
            inventory()
```

```
        {
              last=max_types-1;
        };
        void insert(char* name, int q, int o);
        void replenish(char* name, int num);
        int check(char* name);
        int withdraw(char* name, int num);// this function
        //returns -1 if the inventory falls below the order level.
};
```

The implementation of class inventory and a test program are illustrated in Appendix A.

4.3.2 Class Order Manager

The order is the basic driving force of the production system. "No order" means no production. After being entered into the system, the order has to go through several stages before it leaves the system. So it is desirable to transform the order form into a class in order to get the orders through the system smoothly. The order class is called a "class entity." Thus an order is an object of class entity. The class entity is going to be discussed in detail in the section "4.4 Service Objects." The class order manager handles the orders before they are released to the production system.

The class order manager has only one datum, a pointer which points the root of the list of orders. The class order manager has the

following methods:

1. A constructor which initializes the root pointer.
2. void accept(entity* a) - used to add an order to the appropriate position in the order list according to the new order's priority.
3. entity* release() - used to release the order at the head of the list.
4. int present(int a) - used to check if a specific order is still in queue. The parameter "a" is the order identification number. This function returns 1 if the order is still in the queue; returns 0 otherwise.
5. int change-prior(int ord, int pri) - used to change the priority of an order. It takes two parameters, with the first being the order ID number, and the second new priority. Notice that this function is only effective for the orders still in the queue.
6. int remove(int a) - used to remove an order from the queue. The parameter is the order ID number. It returns 1 if successful.
7. void show() - used to print out a list of orders still in the queue.
8. void clear() - used to clear the order queue.
9. void save() - used to save the queue status.
10. void load() - used to load the queue status.

The formal definition of class order manager is shown below. The implementation of this class and a test program is listed in Appendix B.

```
class omanage
{
```

```
private:
        entity* root;
public:
        omanage() {root=0;}
        void accept(entity* a);// add to the back of the queue.
        entity* release();//release the head of the queue.
        int present(int a);// find if an order is still in the queue
        int change_prior(int ord, int pri);//change order priority.
        int remove(int a);//remove an order from the queue.
        void show();//show all orders in the queue.
        void clear();//clear all orders in the queue.
        void save();//save the order status.
        void load();//load the order status.
};
```

4.3.3 Class Production Report

The class production report contains all the information about the products produced. The main task of this class is generating a production report which shows the finished orders and the total numbers for each product.

The class report has the following private data:

1. A list of orders produced during the working day.
2. A list of product nodes. Each node contains a product name and the total number produced.

The public methods and parameters defined for class report are

illustrated below:

1. void append(entity* a) - used to accept an order entity and attach it to the back of the finished order list.
2. int present(int ordnum) - used to check whether an order is in the finished list. The parameter is order ID number.
3. int show() - used to generate a production report.
4. void save() - used to save the current status.
5. void load() - used to load previously saved status.
6. void clear() - used to initialize the class.

The formal definition is shown below, and the implementation and a test program is listed in Appendix C.

```
class  report
{      private:
            entity* root; // Head of the list.
            entot* top; // Head of the total number list.
       public:
            report() {root=0; top=0;}
            void append(entity* a);
            int present(int orderno);
            int show(entity* y=0, int first=2);
            void save();// save the current status.
            void load();// load the previous status.
            void clear(); // Remove all the entities.
};
```

4.3.4 Class Parts Program Management

This class has the responsibility of managing all part programs for the machines in the system. It may help to load new programs, remove the old ones, and to provide information about programs in storage. In this system only one object is used to manage all parts programs. An alternative approach is to declare an object of this class for each cell.

This class has only one private datum: a pointer which points to a list of parts programs.

This class has the following methods:

1. int inquiry(char* a) - used to check if a parts program is in storage. The parameter is file name.
2. int newpart(char* a) - used to load a new parts program.
3. int remove(char* a) - used to delete an old program.
4. int replace(char* a) - used to replace an old program with a new one.
5. void show() - used to print a list of programs in storage.
6. void save() - used to save the program list.
7. void load() - used to load the program list.

The formal definition is shown below, and the implementation and a test program are listed in Appendix D.

```
class pprog
{
      private:
            pnode* head;
      public:
            pprog() {head=0;};
            int inquiry(char* a);
            int newpart(char* a);
            int remove(char* a);
            int replace(char* a);
            void show();
            void save();
            void load();
};
```

4.3.5 Class AS/RS

The class AS/RS manages the automatic storage and retrieval system. There are 124 storage positions in the rack. Several different types of pallets can be stored on the rack. So the pallet name is used as an identifier for storing and retrieving. Two rules are used in the management. First, whenever a pallet enters the storage system, the AS/RS manager finds the first empty position which is near the entry point of the conveyor. Second, whenever there is a need to withdraw a pallet, the AS/RS manager finds one near the entry point.

The class AS/RS has only one datum which is an array of boxes.

Each box contains information about pallet type and box ID number. The six methods of this class are shown below.

1. A constructor which initializes the array of boxes.
2. int check(char* name) - used to check if a specific kind pallet is in storage.
3. int comein(char* name) - used for storage. It returns the position of the first empty box.
4. int goout(char* name) - used for retrieval. It returns the position of the first available pallet.
5. void save() - used to save the current status of AS/RS.
6. void load() - used to load the previous status.

The formal definition is shown below, and the implementation and a test program are listed in Appendix E.

```
class asrs
{
      private:
            asnode box[124];
      public:
            asrs();
            int check(char* name);
            int comein(char* name);//Return the position, otherwise 0.
            int goout(char* name);//Return the position, otherwise 0.
            void save();//save the as/rs status.
            void load();//load the as/rs status.
};
```

4.3.6 Class MWS, AWS, and MHS

These classes are interface modules which carry the message back and forth between the shop controller and the cell controllers. MWS stands for machining working system, which handles the Machining Cell interface, AWS stands for assembly working system, which handles the Assembly Cell interface, and MHS stands for material handling system, which handles the interface of the material handling system. Since these functions have something in common, that is, basic I/O function, a generic interface class was designed. The three cell interface classes are declared to be the subclasses of the generic interface class. Here only class MWS is explained in detail, the other two interface classes are listed in appendices without explanation.

The class MWS has the following private data:

1. Two character strings which specify the mail box names.
2. Several character strings which store the commands for the machining cell.

The MWS class has the following methods:

1. A constructor which initialize all the character strings in the private section.
2. void mgoon() - used to send the command "okgoon".
3. void millr() - used to send the command of milling a mini robot.
4. void millc() - used to send the command of milling a mini CNC. Notice that mini robot and mini CNC are two products.

5. void initia() - used to send the command of initializing the machining cell.

6. char* readmws() - used to get the feedback information from the cell.

The formal definition and implementation are shown below:

```
class mws : public geninter
{
      private:
            char outf[15];
            char inf[15];
            char mirob[15];
            char micnc[15];
            char initia[15];
            char goonm[15];
      public:
            mws()
            {
            strcpy(outf,"mwsout.dat");
            strcpy(inf,"mwsin.dat");
            strcpy(mirob,"mirobot   ");
            strcpy(micnc,"micnc     ");
            strcpy(initia,"initia    ");
            strcpy(goonm,"okgoon    ");
            };
            void mgoon()
```

```
        {geninter::iwrite(outf,goonm);      };
    void millr()
        {geninter::iwrite(outf,mirob); };
    void millc()
        {geninter::iwrite(outf,micnc); };
    void initm()
        {geninter::iwrite(outf,initia);     };
    char* readmws()
    {return  geninter::iread(inf);      }
};
```

4.3.7 Operator Interface Module

In this control system, the operator interface module is designed as a subroutine. Its main responsibility is to translate the messages between the shop controller and the operator. There are three types of interaction between the operator and the shop controller. One is supervising activity such as order input, another is maintaining activity, and the last is loading activity.

Normally the shop control system works continuously. When the operator wants an interface, he/she can type one of three words to bring up the interface screen. The three words are monitor, problem, and loading. The menus associated with each of these inputs are shown in Figures 4.1, 4.2 and 4.3 respectively.

MONITOR MENU

1. Input an order;
2. Add raw materials;
3. View an order's progress;
4. Change an order's priortity;
5. Delete an order from the queue;
6. Do some parts program manipulation;
7. Request a production report;
8. Stop the whole system operation;
9. Show orders still in the queue;
10. Show parts programs in storage;
11. Exit the operator interface and back to normal run.

Enter your choice please:

Figure 4.1 The monitor menu.

LOADING MENU

1. Load a robot base pallet;
2. Load a CNC base pallet;
3. Load a link pallet;
4. Load an empty pallet;

First, you should put your pallet on the conveyor,
then enter your choice.

Figure 4.2 The loading menu.

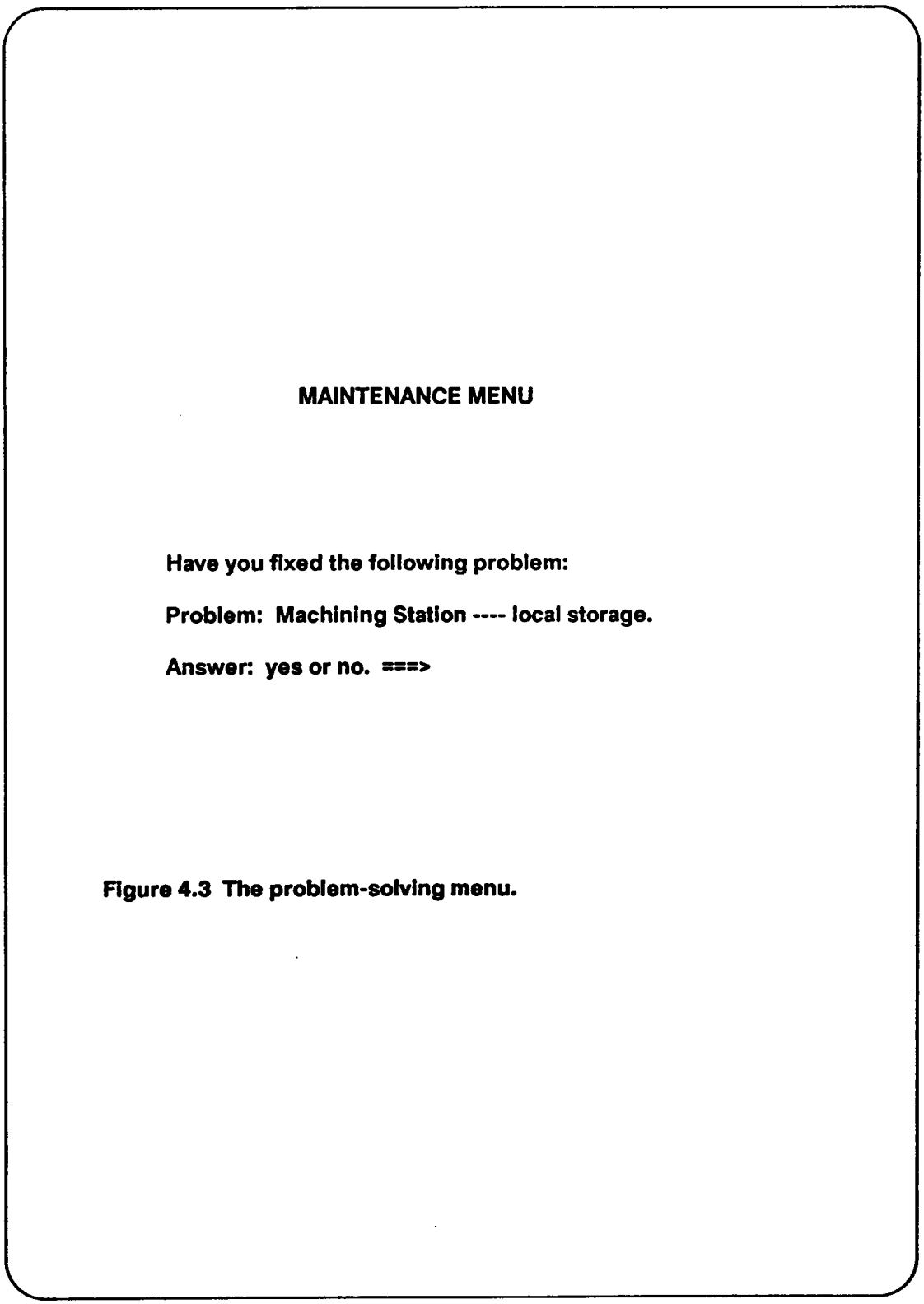

Figure 4.3 The problem-solving menu.

The menus are self explanatory, however one thing about the monitor interface needs to be mentioned. After the operator finishes the monitoring interface operation, the system does not return control automatically to the shop controller. The operator must enter the exit option on the menu to get out of the interface. Since the operator interface is an interrupt to the shop controller, the operator should do the interface operations quickly in order to get out of the interface as rapidly as possible.

The other two interfaces are somewhat different. When the system runs out of raw materials or needs more empty pallets, it will give a visible and an audible warning. When the operator has the correct material available, he or she should enter loading and the system will respond by presenting an empty pallet for the parts when the material handling system is ready. When a problem occurs in the system, the control system will give out an alarm. The operator should correct the problem and then tell the control system by using the problem interface.

4.4 Service Objects in the Outermost Layer

Service objects are those objects in the outer layer of the control system. In this shop controller, only two kinds of objects are grouped into this category. One group belongs to the class entity which represents customer order entities. The reason to declare the order entities as a class is that order entities are used by many other objects such as the order manager, the production report, and the operator interface module.

The class order entity has the following private data:

1. An integer order number which serves as an identifier.
2. A character string which stores the order type name.
3. An integer as the order priority.
4. A pointer which points to the next order entity.

The class entity has only one method which initializes the private data. The formal definition is shown below.

```
class entity
{
      public:
            int ord_num;
            char type[15];
            int quan;
            int prior;
            entity* next;
      entity()
      {
            ord_num=0;
            for(int t=0; t<15;++t)
            type[t]='\0';
            quan=0;
            prior=0;
            next=0;
      };
};
```

The class generic interface is used by each of the three cell interface classes in the middle layer. Basically what the three cell interface classes do is the message I/O function, but each class has its own private data. The declaration of the generic interface class simplifies the development of the three cell interface classes. With the generic interface class the program developer can concentrate on message processing rather than on both the message processing and I/O function.

The class generic interface has the following data:

1. Two file pointers which point to the input and output mailbox.
2. A string of characters which serves as a buffer for input messages.

The class generic interface has two public methods, one of them reading messages from the input mail box, and the other writing messages to the output mail box.

The formal definition and implementation is shown below:

```
class geninter
{
      private:
            FILE* fpin;
            FILE* fpout;
```

```
            char buffer[30];
        public:
            char* iread(char* a);
            void iwrite(char* a, char* m);
};
char* geninter::iread(char* a)
{
        if((fpin=fopen(a, "r"))==0)
        {
            buffer[0]='\0';
            return buffer;
        }
        else
        {
            fread(buffer,1,30,fpin);
            fclose(fpin);
            unlink(a);
            return buffer;
        }
};
```

It is important to notice that class parts program management and class generic interface are designed specifically for the STARLAN network. These two classes may not work with other networks. But all the other classes are portable.

4.5 Network Communications

In this CIM system, the shop controller communicates with the cell controllers through the AT&T STARLAN network. The STARLAN network is compatible with the IEEE 802.3 draft standard for local area networks. The data transmission speed of STARLAN is 1 megabit per second. Since STARLAN uses a star protocol, all computers can only communicate directly with the central computer (the server). Thus if a computer needs to communicate with another computer other than the server, two steps must be taken. First, one computer sends the message to the server, then the other retrieves that message from the server. The message format selected for this research is shown in Figure 4.4.

The working cycle of this shop controller is 5 seconds. Some communication restrictions for the cell controllers are:

1. Read shop commands every 12 seconds; after reading delete the message file.
2. Write feedback messages immediately after finishing a command; if the shop controller has not retrieved the last feedback message, wait for 15 seconds and try again.
3. If there is a problem, report the problem first before sending the message "done."

The commands for the cell controllers, the feedback messages, and the method to transmit feedback messages are given in Tables 4.1, 4.2 and 4.3 respectively.

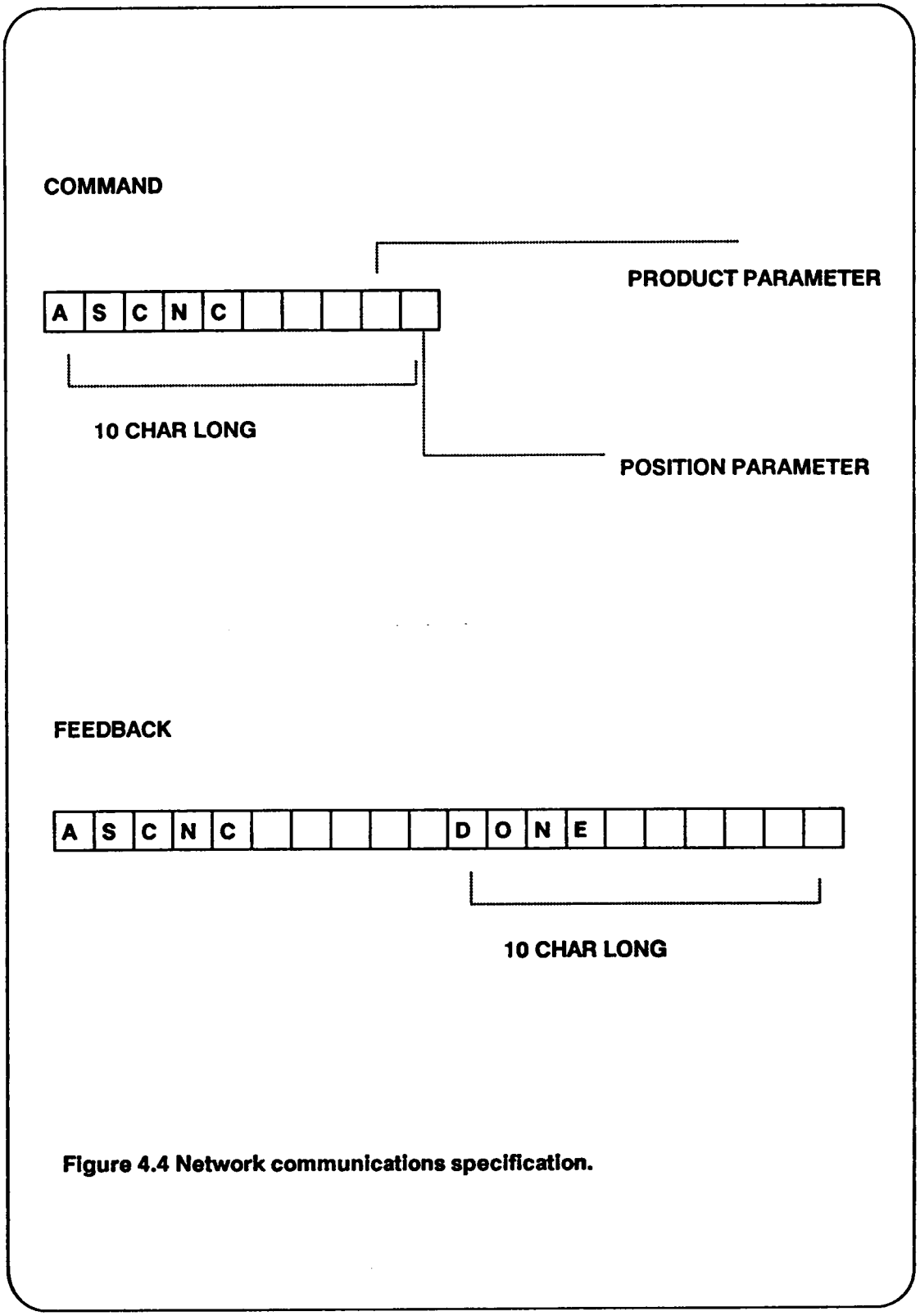

Figure 4.4 Network communications specification.

Table 4.1 Shop commands for the cell controllers.

1. **Assembly Cell:**

   initia , asrobot , ascnc , burobot , bucnc ,

   lorobot , locnc , lolin , okgoon ;

   The assembly cell gets the shop commands from file "awsout.dat".

2. **Material Handling:**

   ratoas , ratomi , astora , mitora , astomi ,

   mitoas , optora , ratoop , okgoon , initia ;

   The material handling system gets the shop commands from "mhsout.dat".

3. **Machining Cell:**

   mirobot , micnc , initia , okgoon;

   The machining cell gets the shop commands from file "mwsout.dat".

Table 4.2 Status message list for the cell controllers.

1. **Machining Cell.**

   Response to the commands: 1. ready, 2. busy, 3. done.

   Problem list: 1. pallet problem, 2. machine No.1 problem,
   3. machine No.2 problem.

2. **Assembly Cell.**

   Response to the commands: 1. ready, 2. busy, 3. done.

   Request: 1. robot feed request, 2. CNC feed request,
   3. link feed request.

   Problem list: 1. pallet problem, 2. assembly difficulty,
   3. kit making difficulty.

3. **Material Handling.**

   Response to the commands: 1. ready, 2. busy, 3. done.

   Problem list: 1. clear conveyor, 2. conveyor problem,
   3 AS/RS problem, 4. machine vision problem.

Table 4.3 Method of sending the status message.

1. Response to commands:

   attach "busy, done" to the end of the commands;

   "ready" should be sent alone.

2. Problem report:

   "problem 1" is example of the feedback message. The tenth character is the parameter.

3. Request message:

   "rfeed ", "cfeed ", "lfeed " are the messages for the feeder request.

5. Analysis and Discussion

In this chapter, the capabilities and limitations of the shop controller will be discussed. Some problems which occurred in the design and implementation of the shop controller will also be addressed.

5.1 Implementation Problem

The design, development and implementation of the shop controller has preceeded the development and implementation of the three workcell controllers. This has made the testing of the shop controller in a real environment impossible. To solve this problem, three cell controller simulators were developed to run both interactively and automatically.

The operation of the interactive cell controller simulator is very simple. First, the simulator receives a command from the shop controller. This is then displayed on the cell controller screen, along with several possible responses. The operator chooses an answer, then the simulator sends the message back to shop controller. The operator may want to wait a period of time before responding to the command. This waiting time is similar to the processing time of the command. Each of the three workcell controllers would be represented by a different computer running a simulator program.

In order to test the operational features of the shop controller, interactive simulation programs were written for the three workcell

controllers which received messages from the shop control conputer and displayed the information on the monitor. Keyboard input could then be used to create a desired response for the workcell controller to send to shop control. This provided a mechanism to force shop control through different courses of action. The interactive simulator menus are shown in Figure 5.1, 5.2 and 5.3. The implementation of these simulators are listed in appendices. Since these simulators can be manually driven to simulate the various anticipated states of the system, they were very useful in debugging and improving the production rules.

In order to simulate longer periods of operation, fully automatic simulators were written for each of the cell controllers. These programs received shop control commands and then randomly selected appropriate responses. The system was then permitted to run and the resulting operations observed. Operator inputs to the shop control were utilized to drive the system through desired operation conditions and the resulting actions of the workcell controllers were observed. Listing of the automatic cell controller simulators are given in the appendices.

5.2 Capabilities and Limitations of the Shop Controller

Usually an expert system is developed for a particular situation, and in this case, there is no exception. This expert system has several limitations. First, it can only handle two products. If more products are to be included, more information about the product structures and the relationship to the capabilities of each cell should be added to the

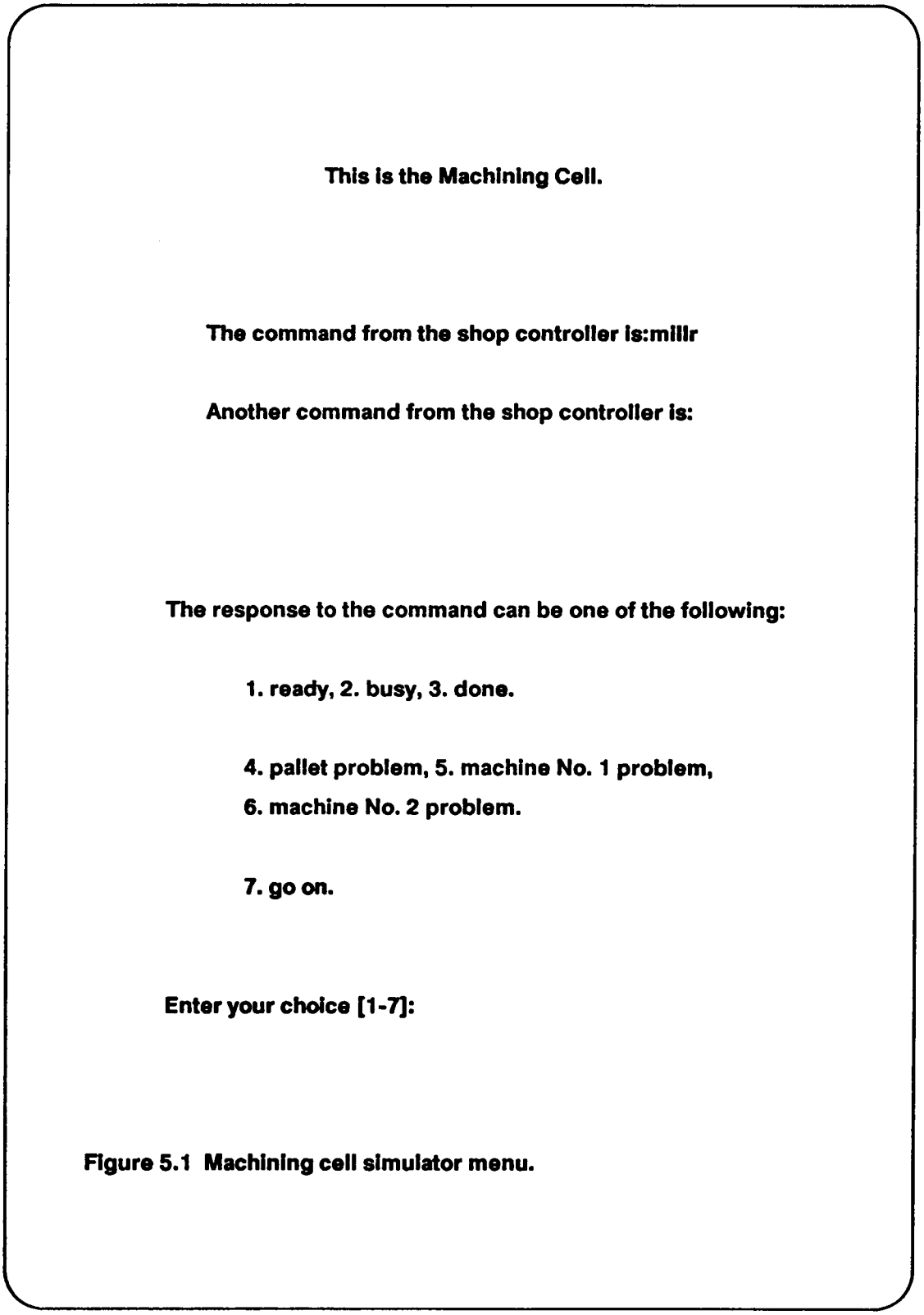

Figure 5.1 Machining cell simulator menu.

This is the Assembly Cell.

The command from the shop controller is:asrobot

Another command from the shop controller is:

The response to the command can be one of the following:

1. ready, 2. busy, 3. done.

4. robot feed request, 5. CNC feed request.
6. link feed request.

7. pallet problem, 8. assembly difficulty.
9. kit making difficulty.

10. go on.

Enter your choice [1-10]:

Figure 5.2 Assembly cell simulator menu.

This is the Material Handling System.

The command from the shop controller is:astomi 6

Another command from the shop controller is:

The response to the command can be one of the following:

1. ready, 2. busy, 3. done.

4. clear conveyor, 5. conveyor problem.

6. AS/RS problem, 7. machine vision problem.

8. go on.

Enter your choice [1-8]:

Figure 5.3 Material handling system simulator menu.

system. Secondly, the production rules have not been tested in the real environment of the physical system. In order to make the control system function properly with the real system, some production rules may need to be changed, and others may need to be added. Thirdly, the expert system needs to be improved to handle additional system control functions, such as parts programs transfer, inventory management and other operations. Inventory Manager and Parts Program Manager have been fully developed, but were not incorporated into the decision part (expert system) to keep the tasks of the workcell controllers as simple as possible.

5.3 STARLAN Network

A Local Area Network (LAN) is an essential part of any distributed control system. It functions as a set of communication linkages which provide data exchange for data communication devices in a small area. The key elements which determine the performance of a LAN are its transmission speed, transmission medium, and medium access control protocol [Stallings, 1985].

The LAN used in this system is the AT&T STARLAN network running at 1 megabit per second. Topologically it is a star network and it utilizes the CSMA/CD access protocol. CSMA/CD stands for Carrier Sense Multiple Access with Collision Detection and is referred to as "listen while talk." A station wishing to transmit listens to the medium to see whether another transmission is in process. If the medium is available, the station begins transmitting; if it is not, it waits for a period of

time and makes another attempt. While a station is in the transmitting process, it still listens to the medium to determine whether other stations want to transmit. If other stations want to transmit, it sends a busy signal to indicate a transmission is in process, and waits for a period of time. After the waiting, it tries to transmit again [Stallings, 1985].

One problem with the STARLAN network that was observed during the trial run was that the network was so busy that all cell controllers could not send any message to the shop controller. This was caused by the fact that each controller was executing a very tight loop which accessed the network continually when there were no other activities. To solve this problem, delays were added to the control loop such that the shop controller's cycle time was extended to about five seconds. Some restrictions on the cell controllers were also added. These restrictions are discussed in the section "4.5 Network Communications."

Another problem with the STARLAN network is that if two cell controllers submit a request to use the network at the same time, only one can do the transmission, and the other one will stop running and enter into an interrupt state. Theoretically, this kind of problem should not occur. It is believed that the STARLAN network is not robust enough to function in this kind of environment. Actually an local area network which uses CSMA/CD with a star topology is the ideal network for this CIM control system because of the imbalanced communication load.

5.4 Operator Interface Problem

One problem with the shop controller software system is its inability to respond to the state changes of the system while it is in the operator interface process. In the present shop control system, the operator interface is designed as an interrupt driven process. During this operation, the shop controller stops checking messages from the cell controllers. Since it usually takes less than a minute for an operator to input some desired information, this interrupt does not have a serious impact on the system performance. In the future, the problem can be solved by installing the operator interface functions on a separate computer. That computer would serve as a preprocessor for the operator's inputs and communicate with the shop controller just like a cell controller. Another way to solve the problem would be to install the shop controller program and the operator interface program onto a computer which runs the Unix operating system. Since Unix is a multitasking operating system, the two programs can run concurrently and communicate with each other through shared memory.

6. Conclusions

6.1 Conclusions

In recent years, American industry has been striving to achieve computer integrated manufacturing in an attempt to reduce production costs and improve its position in the world market place. The integration of different manufacturing processes and functions requires a common communication mechanism. This common communication mechanism is the system control software which coordinates the various production systems through a local area network. The development of this system control software has proven to be a great challenge because of the complexity of the production systems.

In this research, a mechanism for the integration of a variety of manufacturing processes and functions has been developed. The system being considered is located in the Computer Integrated Manufacturing Laboratory at Virginia Tech and has three subsystems: the machining cell, the assembly cell, and the material handling sub-system. Each subsystem forms a cellular production unit which has its own PC as a cell controller. In order to control this production system, an intelligent shop controller was developed, using the C++ language. In the development of the shop controller, a new method called the three-layer integrated approach was used.

In the three-layer integrated approach, two basic techniques are used. One of them is expert systems, the other one is object-oriented programming. The central layer of the control system is the decision-making part and is implemented as a rule-based expert system. The middle layer is the action layer and is implemented as an object-oriented program with several action objects. This layer serves as a preprocessor which manipulates the information flow between the decision layer and the outside world. The algorithmic knowledge, part of the static database, and other information sub-systems reside in this layer. The outer layer is called the service layer which contains some generic objects that interact with the action layer and permit communications with the environment. This layer might also include mathematical computing objects. Usually the objects in the outer layer are used by the objects in the middle layer. This approach can be used to design the control system for any level in the production system.

This approach has several advantages:

1. Software intelligence. By its very nature, a CIM system can provide a vast amount of data regarding the past and present states of the system. The problem of a system controller is how to use the data in controlling the system. The three-layer integrated approach attempts to include human expertise in the system via the central layer. Adaptation and modification of this layer can be used to improve system performance.
2. Modularity. The production functions of the middle and third layers of the control system are modeled as a set of objects. Each object is independent and the expert systems is used to

integrate these objects into a working unit. Hence, it is easy to modify the control system structure designed with this approach. For example, when a new cell is added, the programmer will need to design a new interface object for the new cell using the generic I/O class, and change some rules in the expert system to accommodate the changes. Changes to other parts of the code would be minimal, if any.

3. Reusability. Program reusability is enhanced by object-oriented programming (OOP) because the use of the encapsulation concept. With OOP, the programmer needs to understand the behavior of a class as specified by the methods; the programmer does not have to worry about its implementation. Thus, if a class exists which can perform the desired functions, the programmer can incorporate it into his/her software in a new situation.
4. Maintainability. Program maintainability is enhanced by OOP. Changes in data and procedures can be localized to the classes or objects which implement the data and procedures. If the class can keep the existing interface format, there will be no fall-out effects on other data and procedures in the software system.

The drawback of the three-layer integrated approach is the size of the software system. Using OOP makes the software code larger than using a more traditional method. This problem can be remedied by using efficient languages such as C, or by keeping the production unit the controller manages small. Additionally, as the speed and size of the control computers increases, the effects of this disadvantage are reduced.

6.2 Future Research

The ideas introduced in this work can be extended to enhance control of the laboratory system. Some areas in this work need further investigation.

First, the production rules implemented in the shop controller need to be improved. Only a base set of rules were implemented and experience needs to be gained with the actual system. Additionally, more use could be made of algorithmic processing to permit the expert system to make better decisions. This might include the implementation of more advanced scheduling algorithms for improving the performance of the system under heavy loads or the dynamic control of the material handling system to permit the control of multiple pallets simultaneously. Learning could also be added to the system. The system could begin with a basic knowledge of the production system, and add more knowledge as production goes on. This is a self-improving process. Some research can be done using an "AI" language, such as Prolog or Lisp, to implement the expert system while using C++ to write the objects. The combination of the two languages would enhance the performance of the system and greatly extend its potential applicability to more general systems. Finally, an effort should be made at putting the operator interface on a separate computer that would serve as the management interactive subsystem. This would increase the efficiency of the shop controller and eliminate potential conflicts in task management.

Bibliography

Albus, J.S., C.R. McLean, A.J. Barbera and M.L. Fitzgerakd, "An Architecture for Real-Time Sensory-Interactive Control of Robots in a Manufacturing Facility." Proceedings of the 4th IFAC Symposium on Information Control Problems in Manufacturing Technology. Gaithersburg, MD, Oct. 26-28, 1982.

Arai, Y., S. Hat, T. Imakubo and K. Kikuchi, "Production Control System of Microcomputers Hierarchical Structure for FMS." Proceedings of the 1st International Conference on Flexible Manufacturing Systems. Brighton, UK, Oct. 20-22, 1982, pp. 259-268.

Bedworth, D.D. and J. Bailey, *Integrated Production Control System*. John Wiley & Sons, New York, 1987.

Ben-Arieh, D., "Knowledge Based Control System for Automated Production and Assembly," PhD. Thesis, Department of Industrial Engineering, Purdue University, 1985.

Bertrand, J.W.M., "A Hierarchical Approach to Structuring the Production Control in Multi-Production Management Systems." Modeling Production Management Systems. P. Falster and R.B. Mazumder, ed., North-Holland, 1984.

Charniak, E. and D. McDermont, *Introduction to Artificial Intelligence*. Addison Wesley, Reading, Massachusetts, 1985.

Cox, B.J., *Objective-Oriented Programming*. Addison Wesley, Reading, Massachusetts, 1986.

DEC/Philips, *Reference Model of Production Systems*. Digital Equipment Corporation and Netherland Philips Bedrijven BV, 1987.

Dutton, D.J., "The Design and Implementation of A Cell Controller," M.S. Thesis, Department of Industrial and System Engineering, Georgia Institute of Technology, 1987.

Frarnum, G.T., "Automation for Survival," Manufacturing Engineering. Vol. 96, No. 4, April, 1986.

Feibel, W., *Using QuickC*. Osborne, Berkeley, 1988.

Fischler, M.A. and O. Firschein, *Intelligence: the Eye, the Brain, and the Computer*. Addison Wesley, Readings, Massachusetts, 1987

Fox, M.S., B. Allen and G. Strohm, "A Computer Aided Decision System." Management Science. Vol 15, No. 10, June 1969, pp. 550-561,

Gershwin, S.B., R.R. Hildebrant, R. Suri and S.K. Mitter, "A Control Perspective on Recent Trends in Manufacturing Systems." IEEE Control System Magazine. Vol. 6, No.2, April, 1986, pp. 194-199.

Ham, I, K. Hitomi and T. Yoshida, *Group Technology*. Kluwen-Nijhoff Pub., Boston, 1985.

Jones, A.T. and C.R. McLean, "A Cell Control System for the AMRF." Computers in Engineering. Vol. 2, The Society of Mechanical Engineers, 1984.

McLean,C., M. Mitchell and E. Barkmeyer, "A Computer Architecture for Small-Batch Manufacturing," IEEE Spectrum. Vol. 20, No. 5, May 1983, pp. 59-64.

Mesarovic, M.D., D. Macko and Y. Takahara, *The Theory of Hierarchical Multilevel Systems*. Academic Press, New York, 1970.

Nau, D.S., "Expert Computer Systems." Computer. Vol. 16, No. 2, Feb. 1983, pp. 63-85.

Rolston, D.W., *Principles of Artificial Intelligence and Expert Systems*. McGraw-Hill, New York, 1988.

Schildt, H., *Artificial Intelligence Using C*. McGraw-Hill, Berkeley, 1987.

Stallings, W., "Local Networks," *Local Network Technology*, W. Stallings, ed., IEEE Computer Society Press, Washington D.C., 1985.

Stroustrup, B., *The C++ Programming Language*. Addison Wesley, Reading, Massachusetts, 1986.

Teicholz, E, "Computer Integrated Manufacturing," Datamation. Vol. 30, No. 3, March, 1984, pp. 169-174.

U.S. Department of Commerce, *Statistical Abstracts of United States 1987*. U.S. Government Printing Office, Washington D.C., 1986.

Wiener, R.S. and L.J. Pinson, *An Introduction to Objective-Oriented Programming and C++*. Addison Wesley, Reading, Massachusetts, 1986.

Appendices.

Appendix A. Class Inventory and Its Test Program.

```
#include <stdio.h>
#include <string.h>
#include "inventor.h"
main()
{
      inventory inven;
      inven.insert("robot",30,5);
      inven.replenish("robot",20);
      inven.insert("vpisu", 40, 6);
      inven.replenish("vpisu", 50);
      inven.withdraw("vpisu", 60);
      inven.withdraw("robot",20);
      printf("%d",inven.check("robot"));
      printf("\n%d", inven.check("vpisu");
}
class component
{
      friend class inventory;
      private:
            char type[15];
            int quantity;
            int order_lvl;
      public:
            component()
            {
                  type[0]='\0';
                  quantity=0;
                  order_lvl=0;
            };
};
const max_types=50;
class inventory
{
      private:
            component storage[max_types];
            int last;
      public:
            inventory()
            {last=max_types-1;};
            void insert(char* name, int q, int o);
            void replenish(char* name, int num);
            int check(char* name);
            int withdraw(char* name, int num);//this function return
            //-1, if the inventory level falls below the order level.
};
void inventory::insert(char* name, int q, int o)
```

```
{
      //the type name is key for managing the inventory.
      last=++last%max_types;
      strcpy(storage[last].type,name);
      storage[last].quantity=q;
      storage[last].order_lvl=o;
};
void inventory::replenish(char* name, int num)
{
      int found=0;
      int current=0;
      while(current<=last && !found)
            {if (strcmp(storage[current].type, name)==0)
                  found=1;
            else
                  ++current;
            }
      if(found)
      storage[current].quantity=storage[current].quantity+num;
};
int inventory::withdraw(char* name, int num)
{
      int found=0;
      int current=0;
      while(current<=last && !found)
            {if (strcmp(storage[current].type, name)==0)
                  found=1;
            else
                  ++current;
            }
      if(!found) return 0;
      else
      {
            storage[current].quantity=storage[current].quantity-num;
            if (storage[current].quantity<storage[current].order_lvl)
                  return -1;
            else
                  return 1;
      }
};
int inventory::check(char* name)
{
      int found=0;
      int current=0;
      while(current<=last && !found)
            {if (strcmp(storage[current].type, name)==0)
                  found=1;
            else
                  ++current;
            }
      if(!found) return 0;
      else
            return storage[current].quantity;
};
```

Appendix B. Class Order Manager and Its Test Program.

```
#include <stdio.h>
#include <string.h>
#include "entity.h"
#include "omanage.h"
main()
{
      entity fee;
      fee.ord_num=1024;
      strcpy(fee.type,"freeman");
      fee.quan=400;
      fee.prior=12;
      entity fee1;
      fee1.ord_num=1536;
      fee1.quan=4000;
      strcpy(fee1.type, "zhang");
      printf("\n\n\n");
      entity fee2;
      fee2.ord_num=1624;
      strcpy(fee2.type,"free");
      fee2.quan=400;
      entity fee3;
      fee3.ord_num=1596;
      fee3.quan=4000;
      strcpy(fee3.type, "zhang");
      entity fee4;
      fee4.ord_num=1044;
      strcpy(fee4.type,"free");
      fee4.quan=400;
      omanage orders;
      orders.accept(&fee);
      orders.accept(&fee1);
      orders.accept(&fee2);
      orders.accept(&fee3);
      orders.accept(&fee4);
      orders.show();
      printf("determine order present. %d",orders.present(1044));
      orders.remove(1024);
      printf("determine order present. %d",orders.present(1024));
      orders.show();
      orders.save();
      orders.clear();
      orders.load();
      orders.show();
}
//Interface of order management.
//This module can accept orders with priority.  If you don't
//specify the priority of an order, this order will be set with
//default priority
class omanage
```

```
{
      private:
            entity* root;
      public:
            omanage() {root=0;}
            void accept(entity* a);// add to the back of the queue.
            entity* release();//release the head of the queue.
            int present(int a);// find if an order is still in the queue
            int change_prior(int ord, int pri);//chang order priority.
            int remove(int a);//remove an order from the queue.
            void show();//show all orders in the queue.
            void clear();//clear all orders in the queue.
            void save();//save the order status.
            void load();//load the order status.
            ~omanage(){clear();}
};
//implementation of order management.
//file omanage.h
void omanage::accept(entity* a)
{
      entity * previous, * current;
      if(!a->prior)
      {
            if(root)
            {
                  previous=root;
                  current=root->next;
                  while(current!=0)
                  {
                        previous=current;
                        current=current->next;
                  }
                  previous->next=a;
            }
            else
            root=a;
      }
      else
      {
            if(root)
            {
                  if (a->prior>root->prior)
                  {
                        a->next=root;
                        root=a;
                  }
                  else
                  {
                        int found=0;
                        previous=root;
                        current=root->next;
                        while((current!=0) && (!found))
                        {
                              if (a->prior>current->prior)
```

```
                        found=1;
                        else
                        {
                              previous=current;
                              current=current->next;
                        }
                  }
                  if(found)
                  {
                        previous->next=a;
                        a->next=current;
                  }
                  else
                  previous->next=a;
            }
      }
      else
      root=a;
}
};
entity* omanage::release()
{
      entity* temp;
      temp=root;
      if(temp)
      {
            root=root->next;
            temp->next=0;
            return temp;
      }
      else
      return temp;
};
int omanage::present(int a)
{
      int found=0;
      entity* current=root;
      while((current!=0)&& (!found))
      {
            if (a==current->ord_num)
            found=1;
            else
            current=current->next;
      }
      if (found)
      return 1;
      else
      return 0;
};
int omanage::remove(int a)
{
      int found=0;
      entity* current=root;
      entity* previous=0;
```

```
        while((current!=0)&& (!found))
        {
                if (a==current->ord_num)
                found=1;
                else
                {
                        previous=current;
                        current=current->next;
                }
        }
        if (found)
        {
                if(current==root)
                {
                        root=root->next;
                        delete current;
                        return 1;
                }
                else
                {
                        previous->next=current->next;
                        delete current;
                        return 1;
                }
        }
        else
        return 0;
};
void omanage::clear()
{
        entity* temp=root;
        root=0;
        if(temp)
        {
        do
        {
                entity* temp1=temp;
                temp=temp->next;
                delete temp1;
        }while(temp);
        };
};
int omanage::change_prior(int ord, int pri)
{
        int found=0;
        entity* current=root;
        entity* temp;
        temp=new entity;
        while(current && !found)
        {
                if(ord==current->ord_num)
                found=1;
                else
                current=current->next;
```

```
        }
        if(found)
        {
        temp->ord_num=ord;
        temp->quan=current->quan;
        temp->prior=pri;
        strcpy(temp->type,current->type);
        remove(ord);
        accept(temp);
        return;
        }
        else
        return;
};
void omanage::show()
{
        entity* current=root;
        fprintf(stdprn,"\n\t\t\t    Order List");
        fprintf(stdprn,"\n\t\tThe Virginia Tech Automated Production
        Facility");
        fprintf(stdprn,"\n\nOrder No.\tType Name\tQuantity\tPriority");
        fprintf(stdprn,"\n\n-----------------------------------------
        -------");
        while(current)
        {
              fprintf(stdprn,"\n\n%d\t\t%s\t\t%d\t\t%d",current->ord_num,
               current->type, current->quan,current->prior);
              current=current->next;
        }
        fprintf(stdprn,"\n-----------------------------------------
        -----");
        fprintf(stdprn,"\n\n\n\n");
};
void omanage::save()
{
        FILE* fp;
        entity* current;
        char ch='h';
        current=root;
        if((fp=fopen("omanage.dat","w"))==0)
        {
              printf("\n\nCan not open file to save\n");
              return;
        }
        printf("\n\n Saving Order Status\n\n");
        while(current)
        {
              fprintf(fp,"%d",current->ord_num);
              for(int t=0; t<15; ++t)
              putc(current->type[t],fp);
              fprintf(fp,"%d",current->quan);
              putc(ch,fp);
              fprintf(fp,"%d",current->prior);
              putc(ch,fp);
```

```
            current=current->next;
        }
        fprintf(fp,"%d",0);
        for(int t=0;t<15;++t)
        putc(ch,fp);
        fprintf(fp,"%d",0);
        putc(ch,fp);
        fprintf(fp,"%d",0);
        putc(ch,fp);
        fclose(fp);
};
void omanage::load()
{
        FILE* fp;
        entity* current;
        char ch;
        int flag=1;
        if((fp=fopen("omanage.dat","r"))==0)
        {
        printf("\n\nNo Order Status Saved in Memory");
        return;
        }
        printf("\n\nLoading Order in Memory\n\n");
        do
        {
        current=new entity;
        fscanf(fp,"%d",&current->ord_num);
        for(int t=0;t<15;++t)
        current->type[t]=fgetc(fp);
        fscanf(fp,"%d",&current->quan);
        ch=fgetc(fp);
        fscanf(fp,"%d",&current->prior);
        ch=fgetc(fp);
        if(current->ord_num>0)
        accept(current);
        else
        flag=0;
        }while(flag>0);
        delete current;
        fclose(fp);
}
```

Appendix C. Class Report and Its Test Program.

```
#include <stdio.h>
#include <string.h>
#include <time.h>
#include "entity.h"
#include "report.h"
main()
{
      entity fee;
      fee.ord_num=1024;
      strcpy(fee.type,"freeman");
      fee.quan=400;
      entity fee1;
      fee1.ord_num=1536;
      fee1.quan=4000;
      strcpy(fee1.type, "zhang");
      report rep;
      rep.append(&fee);
      rep.append(&fee1);
      rep.show();
      rep.save();
      printf("\n\n\n");
      rep.clear();
      rep.load();
      rep.show();

      entity fee2;
      fee2.ord_num=1624;
      strcpy(fee2.type,"free");
      fee2.quan=400;
      entity fee3;
      fee3.ord_num=1596;
      fee3.quan=4000;
      strcpy(fee3.type, "zhang");
      entity fee4;
      fee4.ord_num=1044;
      strcpy(fee4.type,"free");
      fee4.quan=400;
      rep.append(&fee3);
      rep.append(&fee2);
      rep.append(&fee4);
      rep.show();
}
class entot
{
      friend class report;
      private:
            char name[15];
            int total;
            entot* next;
```

```
        public:
        entot()
        {
                name[0]='\0';
                total=0;
                next=0;
        }
};
//To use this file, you must delare <stdio.h>, <time.h> and (string.h>.
class  report
{
        private:
                entity* root; // Head of the list.
                entot* top; // Head of the total number list.
        public:
                report() {root=0; top=0;}
                void append(entity* a);
                int present(int orderno);
                int show(entity* y=0, int first=2);
                void save();// save the current status.
                void load();// load the previous status.
                void clear(); // Remove all the entities.
                ~report() {clear();}
};
void report::append(entity* a)
{
        entity * previous, * current;
        entity* temp=root;
        if (temp)
        {
                previous=temp;
                current=temp->next;
                while(current!=0)
                {
                        previous=current;
                        current=current->next;
                }
                previous->next=a;
        }
        else
        root=a;
};
int report::show(entity* y=0, int first=2)
{
        entity* ent;
        int freq=1331;
        if(first==2)
        {
                ent=root;
        }
        else
        {
                ent=y;
        }
```

```
if(ent==0)
{
      if(top==0)
      {
            sound_beep(freq);
            printf("=====> Nothing has been produced so far.");
            printf("\n\n\n\n\n\n\n");
            return 0;
      }
      else
      {
            entity* current=root;
            entot* temp=top;
            time_t tclock;
            time(&tclock);
            fprintf(stdprn,"\n\t\t\t%s\n",asctime(localtime(&tcloc
            k)));
            fprintf(stdprn,"\n\t\t\t    Production Report");
            fprintf(stdprn,"\n\t\tThe Virginia Tech Automated
            Production Facility");
            fprintf(stdprn,"\n\nOrder No.\tType Name\tQuantity");
            fprintf(stdprn,"\n\n-----------------------------------
            ------------------");
            while(current)
            {
                  fprintf(stdprn,"\n\n%d\t\t%s\t\t%d",current-
                  >ord_num, current->type,
                    current->quan);
                  current=current->next;
            }
            fprintf(stdprn,"\n-----------------------------------
            ----------------");
            fprintf(stdprn,"\n\n\n\tType Name\tTotal Number");
            fprintf(stdprn,"\n\t-------------------------------
            -");
            while(temp)
            {
                  entot* temp1;
                  fprintf(stdprn,"\n\n\t%s\t\t%d",temp->name,
                  temp->total);
                  temp1=temp;
                  temp=temp->next;
                  delete temp1;
            }
            fprintf(stdprn,"\n\n\n\n\n\n\n\n");
            top=0;
            return 1;
      }
}
else
{
      if (first>0)
      {
            if(top==0)
```

```
                {
                        entot* temp;
                        temp=new entot;
                        strcpy(temp->name,ent->type);
                        temp->total=ent->quan;
                        top=temp;
                        show(ent->next,0);
                }
                else
                {
                        entot* current=top;
                        entot* previous=0;
                        while(current!=0)
                        {
                                previous=current;
                                current=current->next;
                        }
                        entot* temp;
                        temp=new entot;
                        strcpy(temp->name, ent->type);
                        temp->total=ent->quan;
                        previous->next=temp;
                        show(ent->next,0);
                }
        }
        else
        {
                int found=0;
                entot* current=top;
                entot* previous=0;
                while((current!=0)&&!found)
                {
                        if(strcmp(current->name, ent->type)==0)
                        found=1;
                        else
                        {
                                previous=current;
                                current=current->next;
                        }
                }
                if (found)
                {
                        current->total=current->total+ent->quan;
                        show(ent->next, 0);
                }
                else
                show(ent,1);
        }
    }
};
void report::clear()
{
    entot* t=top;
    entity* y=root;
```

```
        if(t!=0)
        do
        {
              entot* tt=t;
              t=t->next;
              delete tt;
        }while(t!=0);
        if(y!=0)
        do
        {
              entity* yy=y;
              y=y->next;
              delete yy;
        }while(y!=0);
        top=0;
        root=0;
};
int report::present(int orderno)
{
        entity* current;
        current=root;
        int found=0;
        while(current && !found)
        {
              if(current->ord_num==orderno)
              found=1;
              else
              current=current->next;
        }
        if (found)
        return 1;
        else
        return 0;
};
void report::save()
{
        FILE* fp;
        entity* current;
        char ch='h';
        current=root;
        if((fp=fopen("report.dat","w"))==0)
        {
              printf("\n\nCan not open file to save\n");
              return;
        }
        printf("\n\n Saving Report Status\n\n");
        while(current)
        {
              fprintf(fp,"%d",current->ord_num);
              for(int t=0; t<15; ++t)
              putc(current->type[t],fp);
              fprintf(fp,"%d",current->quan);
              putc(ch,fp);
              fprintf(fp,"%d",current->prior);
```

```
            putc(ch,fp);
            current=current->next;
      }
      fprintf(fp,"%d",0);
      for(int t=0;t<15;++t)
      putc(ch,fp);
      fprintf(fp,"%d",0);
      putc(ch,fp);
      fprintf(fp,"%d",0);
      putc(ch,fp);
      fclose(fp);
};
void report::load()
{
      FILE* fp;
      entity* current;
      char ch;
      int flg=1;
      if((fp=fopen("report.dat","r"))==0)
      {
      printf("\n\nNo Report Status Saved in Memory");
      return;
      }
      printf("\n\nLoading Report Status\n\n");
      do
      {
      current=new entity;
      fscanf(fp,"%d",&current->ord_num);
      for(int t=0;t<15;++t)
      current->type[t]=fgetc(fp);
      fscanf(fp,"%d",&current->quan);
      ch=fgetc(fp);
      fscanf(fp,"%d",&current->prior);
      ch=fgetc(fp);
      if(current->ord_num>0)
      {
            append(current);
            flg=1;
      }
      else
      flg=0;
      }while(flg==1);
      delete current;
      fclose(fp);
}
```

Appendix D. Class Parts Program Manager and its test program.

```
#include <io.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include "pprog.h"
main()
{
      pprog p;
      p.newpart("entot.bak");
      p.newpart("manage.bak");
      p.newpart("omanage.bak");
      p.remove("omanage.bak");
      p.replace("manage.bak");
      int result;
      result=p.inquiry("manage.bak");
      if(result)
      printf("\n Manage.bak is the directory.\n");
      else
      printf("\n Manage.bak is no in the directory.\n");
      p.show();
      printf("\n\n fisrt show\n\n");
      p.save();
      p.load();
      p.show();
}
//To use this class module, you must include <stdio.h>,
//<stdlib.h>,<io.h>,<string.h>.
//File class program node.
class pnode
{
      friend class pprog;
      private:
            char contents[15];
            pnode* next;
      public:
            pnode()
            {
                  for(int i=0; i<15; i++)
                  contents[i]='\0';
                  next=0;
            }
};
class pprog
{
      private:
            pnode* head;
      public:
            pprog() {head=0;};
            int inquiry(char* a);
```

```
            int newpart(char* a);
            int remove(char* a);
            int replace(char* a);
            void show();
            void save();
            void load();
};
int pprog::inquiry(char* a)
{
      pnode* current;
      current=head;
      int found=0;
      while (current && !found)
      {
            if (strcmp(current->contents, a)==0)
            found=1;
            else
            current=current->next;
      }
      if(found)
      return 1;
      else
      return 0;
};
int pprog::newpart(char* a)
{
      pnode* temp;
      temp=new pnode;
      strcpy(temp->contents, a);
      if (head)
      {
            temp->next=head;
            head=temp;
      }
      else
      head=temp;
      // File manipulation.
      char comname[50];
      char* ccp="copy a:";
      strcpy(comname, ccp);
      strncat(comname,a,15);
      printf("\n%s", comname);
      int result;
      result=system(comname);
      if (result==-1)
      {
            printf("\nFile copy failed, please do it when the system is
            down.\n");
            return 0;
      }
      else
      {
            printf("\nSuccessful file tranfer! \n");
            return 1;
```

```
        }
}
int pprog::remove(char* a)
{
        pnode * previous, * current, * temp;
        previous=0;
        current=head;
        int found=0;
        while (current && !found)
        {
                if (strcmp(current->contents, a)==0)
                found=1;
                else
                {
                        previous=current;
                        current=current->next;
                }
        }
        if(found)
        {
                if(current==head)
                {
                        head=current->next;
                        delete current;
                }
                else
                {
                        previous->next=current->next;
                        delete current;
                }
                unlink(a);
                return 1;
        }
        else
        {
                printf("\nProgram %s not exist", a);
                return 0;
        }
};
void pprog::show()
{
        pnode* current=head;
        fprintf(stdprn,"\n\n\t\tPart Programs\n\n");
        fprintf(stdprn,"\n\n-----------------------------------------
        ----");
        while( current)
        {
                fprintf(stdprn, "\n%s \n", current->contents);
                current=current->next;
        }
        fprintf(stdprn, "\n\nEnd of file list.");
        fprintf(stdprn,"\n\n\n\n\n\n\n");
};
int pprog::replace(char* a)
```

```
{
      int result;
      remove(a);
      result=newpart(a);
      if(result)
      printf("\nProgram is successfully replaced\n");
      else
      printf("\nProgram replacement failed.\n");
};
void pprog::save()
{
      FILE* fp;
      pnode* temp;
      temp=head;
      char ch='q';
      if((fp=fopen("pprog.dat","w"))==0)
      {
            printf("\n\nCan not open file to save\n");
            return;
      }
      printf("\n\n Saving Part Programs\n\n");
      while(temp)
      {
            for(int t=0; t<15; ++t)
            putc(temp->contents[t],fp);
            temp=temp->next;
      }
      for(int t=0;t<15;++t)
      putc(ch,fp);
      fclose(fp);
}
void pprog::load()
{
      FILE* fp;
      pnode* temp;
      temp=new pnode;
      if((fp=fopen("pprog.dat","r"))==0)
      {
      printf("\n\nNo Part Program in Memory");
      return;
      }
      printf("\n\nLoading Part Program\n\n");
      for(int t=0;t<15;++t)
      temp->contents[t]=fgetc(fp);
      if(temp->contents[0]=='q' && temp->contents[1]=='q')
      {
      head=0;
      return;
      }
      else
      {
      head=temp;
      do
      {
```

```
        temp=new pnode;
        for( int x=0;x<15;++x)
        temp->contents[x]=fgetc(fp);
        if(temp->contents[0]!='q' && temp->contents[1]!='q')
        {
        temp->next=head;
        head=temp;
        }
        else
        temp->next=0;
        }while(temp->contents[0]!='q' && temp->contents[1]!='q');
        delete temp;
        };
        fclose(fp);
}
```

Appendix E. Class AS/RS and Its Test Program.

```
//File of class ASRS.
//This class has two basic functions, insert a pallet and withdraw a
pallet.
#include <string.h>
#include <stdio.h>
#include "asrs.h"
main()
{
asrs yaoen;
printf("\n%d",yaoen.comein("yao"));
printf("\n%d",yaoen.comein("zhang"));
printf("\n%d",yaoen.comein("free"));
yaoen.save();
yaoen.load();
printf("\n%d",yaoen.goout("yao"));
printf("\n%d",yaoen.goout("zhang"));
};
//File asrs.h
//To use this file, <stdio.h> and <string.h> need to be included.
class asnode
{
      friend class asrs;
      private:
            int bnum;
            char type[15];
            int later;
      public:
            asnode()
            {
                  bnum=0;
                  for(int t=0;t<15;++t)
                  type[t]='\0';
                  later=0;
            };
};
const max_rack=124;
class asrs
{
      private:
            asnode box[max_rack];
      public:
            asrs()
            {
            box[0].bnum=53;
            box[1].bnum=52;
            box[2].bnum=44;
            box[3].bnum=54;
            box[4].bnum=51;
            box[5].bnum=43;
```

```
box[6].bnum=35;
box[7].bnum=45;
box[8].bnum=71;
box[9].bnum=50;
box[10].bnum=61;
box[11].bnum=60;
box[12].bnum=42;
box[13].bnum=49;
box[14].bnum=41;
box[15].bnum=34;
box[16].bnum=26;
box[17].bnum=36;
box[18].bnum=72;
box[19].bnum=80;
box[20].bnum=70;
box[21].bnum=59;
box[22].bnum=48;
box[23].bnum=47;
box[24].bnum=40;
box[25].bnum=33;
box[26].bnum=69;
box[27].bnum=58;
box[28].bnum=46;
box[29].bnum=39;
box[30].bnum=32;
box[31].bnum=25;
box[32].bnum=17;
box[33].bnum=27;
box[34].bnum=81;
box[35].bnum=89;
box[36].bnum=79;
box[37].bnum=68;
box[38].bnum=57;
box[39].bnum=38;
box[40].bnum=31;
box[41].bnum=24;
box[42].bnum=56;
box[43].bnum=67;
box[44].bnum=78;
box[45].bnum=37;
box[46].bnum=30;
box[47].bnum=23;
box[48].bnum=16;
box[49].bnum=8;
box[50].bnum=18;
box[51].bnum=90;
box[52].bnum=98;
box[53].bnum=88;
box[54].bnum=77;
box[55].bnum=66;
box[56].bnum=55;
box[57].bnum=29;
box[58].bnum=22;
box[59].bnum=15;
```

```
box[60].bnum=65;
box[61].bnum=76;
box[62].bnum=87;
box[63].bnum=28;
box[64].bnum=21;
box[65].bnum=14;
box[66].bnum=7;
box[67].bnum=9;
box[68].bnum=64;
box[69].bnum=75;
box[70].bnum=86;
box[71].bnum=97;
box[72].bnum=107;
box[73].bnum=99;
box[74].bnum=20;
box[75].bnum=13;
box[76].bnum=6;
box[77].bnum=74;
box[78].bnum=85;
box[79].bnum=96;
box[80].bnum=19;
box[81].bnum=12;
box[82].bnum=5;
box[83].bnum=73;
box[84].bnum=84;
box[85].bnum=95;
box[86].bnum=106;
box[87].bnum=116;
box[88].bnum=108;
box[89].bnum=11;
box[90].bnum=4;
box[91].bnum=83;
box[92].bnum=94;
box[93].bnum=68;
box[94].bnum=105;
box[95].bnum=10;
box[96].bnum=3;
box[97].bnum=82;
box[98].bnum=92;
box[99].bnum=104;
box[100].bnum=115;
box[101].bnum=125;
box[102].bnum=2;
box[103].bnum=92;
box[104].bnum=103;
box[105].bnum=114;
box[106].bnum=1;
box[107].bnum=91;
box[108].bnum=102;
box[109].bnum=113;
box[110].bnum=124;
box[111].bnum=126;
box[112].bnum=101;
box[113].bnum=112;
```

```
            box[114].bnum=123;
            box[115].bnum=100;
            box[116].bnum=111;
            box[117].bnum=122;
            box[118].bnum=110;
            box[119].bnum=121;
            box[120].bnum=109;
            box[121].bnum=120;
            box[122].bnum=119;
            box[123].bnum=118;
            }
            int check(char* name);
            int comein(char* name);//Return the position, otherwise
            return 0.
            int goout(char* name);//Return the position, otherwise
            return 0.
            void save();//save the as/rs status.
            void load();//load the as/rs staus.
};
int asrs::comein(char* name)
{
      //the type name is key for managing the inventory.
      int found=0;
      int current=0;
      while(current<=max_rack && !found)
            {if (box[current].type[0]=='\0')
                  found=1;
            else
                  ++current;
            }
      if(found)
      {
            strcpy(box[current].type, name);
            return box[current].bnum;
      }
      else
            return 0;
};
int asrs::goout(char* name)
{
      int found=0;
      int current=0;
      while(current<=max_rack && !found)
            {if (strcmp(box[current].type, name)==0)
                  found=1;
            else
                  ++current;
            }
      if(found)
      {
            box[current].type[0]='\0';
            return  box[current].bnum;
      }
      else
```

```
        return 0;
};
void asrs::save()
{
        FILE* fp;
        if((fp=fopen("asrs.dat","w"))==0)
        {
                printf("\n\nCan not open file to save\n");
                return;
        }
        printf("\n\n Saving AS/RS Status\n\n");
        for(int i=0;i<124;++i)
        {
                for(int t=0; t<15; ++t)
                putc(box[i].type[t],fp);
                fprintf(fp,"%d",box[i].later);
        }
        fclose(fp);
};
void asrs::load()
{
        FILE* fp;
        if((fp=fopen("asrs.dat","r"))==0)
        {
                printf("\n\nNo AS/RS status has been saved\n\n");
                return;
        }
        printf("\n\n Loading AS/RS Status\n\n");
        for(int i=0;i<124;++i)
        {
                for(int t=0; t<15; ++t)
                box[i].type[t]=fgetc(fp);
                fscanf(fp,"%d",box[i].later);
        }
        fclose(fp);
};
int asrs::check(char* name)
{
        int found=0;
        for(int i=0;i<124;++i)
        {
                if(strcmp(box[i].type,name)==0)
                ++found;
        }
        return found;
};
```

Appendix F. Class Interfaces for AWS and MHS.

```
//file awsint.h
class aws : public geninter
{
      private:
            char outf[15];
            char inf[15];
            char asrob[15];
            char ascnc[15];
            char burob[15];
            char bucnc[15];
            char lorob[15];
            char locnc[15];
            char lolin[15];
            char initia[15];
            char goona[15];
      public:
            aws()
            {
            strcpy(outf,"awsout.dat");
            strcpy(inf,"awsin.dat");
            strcpy(asrob,"asrobot   ");
            strcpy(ascnc,"ascnc     ");
            strcpy(burob,"burobot   ");
            strcpy(bucnc,"bucnc     ");
            strcpy(lorob,"lorobot   ");
            strcpy(locnc,"locnc     ");
            strcpy(lolin,"lolin     ");
            strcpy(goona,"okgoon    ");
            strcpy(initia,"initia    ");
            };
            void agoon()
            {
            geninter::iwrite(outf,goona);
            };
            void assemr()
            {
            geninter::iwrite(outf,asrob);
            };
            void assemc()
            {
            geninter::iwrite(outf,ascnc);
            };
            void buildr()
            {
            geninter::iwrite(outf,burob);
            };
            void buildc()
            {
            geninter::iwrite(outf,bucnc);
```

```
            );
            void loadr()
            {
            geninter::iwrite(outf,lorob);
            };
            void loadc()
            {
            geninter::iwrite(outf,locnc);
            };
            void loadl()
            {
            geninter::iwrite(outf,lolin);
            };
            void inita()
            {
            geninter::iwrite(outf,initia);
            };
            char* readaws()
            {
            return  geninter::iread(inf);
            }
};
// class MHS interface for the material handling system.
class mhs : public geninter
{
      private:
            char outf[15];
            char inf[15];
            char ratoas[15];
            char ratomi[15];
            char astora[15];
            char mitora[15];
            char astomi[15];
            char mitoas[15];
            char optora[15];
            char ratoop[15];
            char goonmh[15];
            char initia[15];
      public:
            mhs()
            {
            strcpy(outf,"mhsout.dat");
            strcpy(inf,"mhsin.dat");
            strcpy(ratoas,"ratoas    ");
            strcpy(ratomi,"ratomi    ");
            strcpy(astora,"astora    ");
            strcpy(mitora,"mitora    ");
            strcpy(astomi,"astomi    ");
            strcpy(mitoas,"mitoas    ");
            strcpy(optora,"optora    ");
            strcpy(ratoop,"ratoop    ");
            strcpy(goonmh,"okgoon    ");
            strcpy(initia,"initia    ");
            );
```

```
void hgoon()
{
geninter::iwrite(outf,goonmh);
};
void rta(int a, int b)
{
ratoas[9]=a;
ratoas[8]=b;
geninter::iwrite(outf,ratoas);
};
void rtm(int a, int b)
{
ratomi[9]=a;
ratomi[8]=b;
geninter::iwrite(outf,ratomi);
};
void atr(int a)
{
astora[9]=a;
geninter::iwrite(outf,astora);
};
void mtr(int a)
{
mitora[9]=a;
geninter::iwrite(outf,mitora);
};
void atm()
{
geninter::iwrite(outf,astomi);
};
void mta()
{
geninter::iwrite(outf,mitoas);
};
void otr(int a)
{
optora[9]=a;
geninter::iwrite(outf,optora);
};
void rto(int a, int b)
{
ratoop[9]=a;
ratoop[8]=b;
geninter::iwrite(outf,ratoop);
};
void ini()
{
geninter::iwrite(outf,initia);
};
char* readmhs()
{
return  geninter::iread(inf);
}
};
```

Appendix G. Operator Interface Module.

```
//human initialization.
typedef void (* menu_fcn)();
menu_fcn command[11];
char dumy[10];
void human()
{
      extern void assign_functions();
      extern void function_1();
      extern void function_2();
      extern void function_3();
      extern void function_4();
      extern void function_5();
      extern void function_6();
      extern void function_7();
      extern void function_8();
      extern void function_9();
      extern void function_10();

      int choice;
      assign_functions();
      do
      {
            printf("\n1. Input an order");
            printf("\n2. Add raw material");
            printf("\n3. View an order's progress");
            printf("\n4. Change an order's priority");
            printf("\n5. Delete an order from the queue");
            printf("\n6. Do some parts program manipulation");
            printf("\n7. Request a production report");
            printf("\n8. Stop the whole system operation");
            printf("\n9. Show orders still in the queue");
            printf("\n10. Show part programs in storage.");
            printf("\n11. Exit the operator interface, back to normal
            run");
            printf("\n\nEnter your choice please:");
            scanf("%d", &choice);
            gets(dumy);
            if(choice>=1 && choice<=10)
            command[choice-1]();
      }
      while (choice !=11);
      printf("\n\n\n\n\n\n");
      printf("\n\nIf you want to have access to the system,");
      printf("\nplease type 'monitor' or 'loading' or 'problem'.");
      printf("\nThe system will respond to you in a second.\n");
      manreq=0;
      return;
}
void function_1()
```

```
{
      int yao;
      char choice[10];
      char temp[10];
      do
      {
            entity* newo;
            newo=new entity;
            printf("\n\n\n");
            printf("\nEnter the order number, choose one from [1000-
            9999]");
            printf("\nBe sure each order has an unique number:");
            scanf("%d",&newo->ord_num);
            gets(dumy);
            printf("\nEnter product (or part) name please:\n");
            gets(newo->type);
            printf("\nEnter the quantity of the order:");
            scanf("%d",&newo->quan);
            gets(dumy);
            printf("\nDo you want to this order a priority [Yes , No]");
            gets(temp);
            if(temp[0]=='y')
            {
                  printf("\nEnter the priority for this order [1-50]:");
                  scanf("%d",&newo->prior);
                  gets(dumy);
            }
            printf("\n\n\n\n");
            printf("\tHere is a list of what you have just entered,\n");
            printf("\tdo you want change anything ?");
            printf("\n\t1. Order Number --> %d", newo->ord_num);
            printf("\n\t2. Order Type -->  %s", newo->type);
            printf("\n\t3. Order Quanatity --> %d", newo->quan);
            printf("\n\t4. Order Priority --> %d", newo->prior);
            printf("\n\n\t5. If everything is correct.");
            printf("\n\n\n\tIf you want to change, enter the line No.[1-
            5]:");
            scanf("%d",&yao);
            gets(dumy);
            if(yao==1)
            {
            printf("\nEnter the order number, choose one from [1000-
            9999]");
            printf("\nBe sure each order has an unique number:");
            scanf("%d",&newo->ord_num);
            gets(dumy);
            }
            else if(yao==2)
            {
            printf("\nEnter product (or part) name please:\n");
            gets(newo->type);
            }
            else if(yao==3)
            {
```

```
            printf("\nEnter the quantity of the order:");
            scanf("%d",&newo->quan);
            gets(dumy);
            }
            else if(yao==4)
            {
            printf("\nEnter the priority of the order:");
            scanf("%d",&newo->quan);
            gets(dumy);
            }
            else
            {
            printf("\n\n No chanage.  Everything is correct.\n");
            }
            orders.accept(newo);
            printf("\nWant to input more orders ? [Yes,No]:");
            gets(choice);
      }
      while (choice[0]=='y');
      return;
}
void function_2()
{
      char choice[10];
      char temp[10];
      char newvar[15];
      do
      {
            int x, y;
            printf("\nEnter component name please:");
            gets(newvar);
            printf("\nEnter the quantity please:");
            scanf("%d",&x);
            gets(dumy);
            printf("\nIf this is a new kind of component, you must");
            printf("\nenter minimum order level. Is this a new kind
            of");
            printf("\ncomponents ? [Yes or No]:");
            gets(temp);
            if(temp[0]=='y')
            {
                  printf("\nEnter the order level for this component:");
                  scanf("%d",&y);
                  gets(dumy);
                  inven.insert(newvar,x,y);
            }
            else
            inven.replenish(newvar,x);
            printf("\nWant to do more component replenishment?
            [Yes,No]:");
            gets(choice);
      }
      while (choice[0]=='y');
      return;
```

```
}
void function_3()
{
      char choice[10];
      int ifpresent;
      int ord_number;
      do
      {
            printf("\nEnter the order number you want to check:");
            scanf("%d",&ord_number);
            gets(dumy);
            ifpresent=orders.present(ord_number);
            if(ifpresent)
            printf("\n This order is still in the queue.");
            else
            {
                  ifpresent=rep.present(ord_number);
                  if(ifpresent)
                  printf("\nThis order has been finished.");
                  else if(ord_number==procent->ord_num)
                  printf("\n This order is in the process.");
                  else
                  printf("\nThe order has never been entered.\n");
            }
            printf("\nWant to check more orders ? [Yes,No]:");
            gets(choice);
      }
      while (choice[0]=='y');
      return;
}
void function_4()
{
      char choice[10];
      int ifpresent, orderpr;
      int ord_number;
      do
      {
            printf("\nEnter the order number please:");
            scanf("%d",&ord_number);
            gets(dumy);
            ifpresent=orders.present(ord_number);
            if(!ifpresent)
            printf("\n This order is no longer in the queue.");
            else
            {
                  printf("\nEnter the priority for this order [1-50].");
                  scanf("%d",&orderpr);
                  gets(dumy);
                  orders.change_prior(ord_number, orderpr);
            }
            printf("\nWant to do more order changes ? [Yes,No]:");
            gets(choice);
      }
      while (choice[0]=='y');
```

```
        return;
}
void function_5()
{
        char choice[10];
        int delord;
        int ifpresent;
        do
        {
                printf("\nEnter the order number you want to remove :");
                scanf("%d",&delord);
                gets(dumy);
                ifpresent=orders.present(delord);
                if(!ifpresent)
                printf("\n This order is no longer in the queue.");
                else
                orders.remove(delord);
                printf("\nWant to do more order deletion ? [Yes,No]:");
                gets(choice);
        }
        while (choice[0]=='y');
        return;
}
void function_6()
{
        char choice[10];
        char part_name[15];
        int select;
        do
        {
                printf("\nYou have two choice to select.");
                printf("\nLoad a new one[1], or replace an old one[2]:");
                scanf("%d",&select);
                gets(dumy);
                printf("\nFirst put the disk which contains the program");
                printf("\nin drive a:, then type the filename :");
                gets(part_name);
                if(select==1)
                p.newpart(part_name);
                if(select==2)
                p.replace(part_name);
                printf("\nWant to do more file manipulation ? [Yes,No]:");
                gets(choice);
        }
        while (choice[0]=='y');
        return;
}
void function_7()
{
        printf("\nYou will get the report from printer.\n\n\n\n");
        rep.show();
        printf("\n\n\n\n");
        return;
}
```

```
void function_8()
{
      printf("\nThis function ends the operation.\n\n\n\n\n");
      keystop=1;
      status();
      return;
}
void function_9()
{
      printf("\nYou will get the order list from printer.\n\n\n\n");
      orders.show();
      printf("\n\n\n\n");
      return;
}
void function_10()
{
      printf("\nYou will get the program list from printer.\n\n\n\n");
      p.show();
      printf("\n\n\n\n");
      return;
}
void assign_functions()
{
      command[0]=function_1;
      command[1]=function_2;
      command[2]=function_3;
      command[3]=function_4;
      command[4]=function_5;
      command[5]=function_6;
      command[6]=function_7;
      command[7]=function_8;
      command[8]=function_9;
      command[9]=function_10;
      return;
}
```

Appendix H. Three Interactive Cell Simulators.

```
//This is the machining cell simulator.
#include <stdio.h>
#include <string.h>
#include <io.h>
#include "facinter.h"
main()
{
      char dumy[10];
      char problem[11];
      char inmess[30];
      char semess[30];
      char buffer[30];
      char command[30];
      char* outf="mwsin.dat";
      char* inf="mwsout.dat";
      char* done="done";
      char* busy="busy";
      char* ready="ready";
      facinter ff;
      int temp,choice;
      strcpy(problem,"problem   ");
      strcpy(inmess,"nodata");
      semess[0]='\0';
      buffer[0]='\0';
      command[0]='\0';
      system("d:");
do{
      system("cls");
      printf("\n\n\n\n\t\tThis is Machining Workstation.");
      strcpy(buffer,ff.iread(inf));
      if((strncmp(inmess,"nodata",6))==0)
      {
            inmess[0]='\0';
            strcpy(inmess,buffer);
      }
      else
      strcpy(semess,buffer);
      printf("\n\n\n\t\tThe command from CELL controller          is:");
      if((strncmp(inmess,"nodata",6))==0)
      ;
      else
      printf("%s",inmess);
      printf("\n\n\t\tAnother command from CELL controller is:");
      if((strncmp(semess,"nodata",6))==0)
      ;
      else
      printf("%s",semess);
      printf("\n\n\n\tThe respond to the command can one of the
      following:");
```

```
printf("\n\n\t1. Ready, 2. Busy, 3. Done, 4. Pallet Problem.");
printf("\n\t5. Machine No.1 Problem, 6. Machine No.2 Problem, 7.
Go on.");
printf("\n\n\t Enter your choice[1-7]:");
scanf("%d",&choice);
gets(dumy);
if(choice>=1 && choice<=6)
{
if(choice==1)
{
      strcpy(command,ready);
      if(strncmp(inmess,"initia",6)==0)
      {
      inmess[0]='\0';
      strcpy(inmess,"nodata");
      }
}
else if(choice==2)
{
      strcpy(command,inmess);
      strncat(command,busy,4);
}
else if(choice==3)
{
      strcpy(command,inmess);
      strncat(command,done,4);
      inmess[0]='\0';
      strcpy(inmess,"nodata");
}
else if(choice==4)
{
      problem[9]=1;
      strcpy(command,problem);
}
else if(choice==5)
{
      problem[9]=2;
      strcpy(command,problem);
}
else if(choice==6)
{
      problem[9]=3;
      strcpy(command,problem);
}

else
;
do{
      temp=ff.iwrite(outf,command);
      if(temp==0)
      {
            double x;
            for(int t=0;t<5000;++t)
            {
```

```
                    x=log10(t);
                    }
              }
        }while(temp==0);
        }
        else
        {
              if(choice==7)
              {
                    semess[0]='\0';
                    strcpy(semess,"nodata");
              }
        }
}while(1);
}
//This is the assembly cell simulator.
#include <stdio.h>
#include <string.h>
#include <io.h>
#include "facinter.h"
main()
{
      char dumy[10];
      char problem[11];
      char inmess[30];
      char semess[30];
      char buffer[30];
      char command[30];
      char* outf="awsin.dat";
      char* inf="awsout.dat";
      char* done="done";
      char* busy="busy";
      char* ready="ready";
      char* rfeed="rfeed";
      char* cfeed="cfeed";
      char* lfeed="lfeed";
      facinter ff;
      int temp,choice;
      strcpy(problem,"problem   ");
      strcpy(inmess,"nodata");
      semess[0]='\0';
      buffer[0]='\0';
      command[0]='\0';
      system("d:");
do{
      system("cls");
      printf("\n\n\n\n\t\tThis is Assembly Workstation.");
      strcpy(buffer,ff.iread(inf));
      if((strncmp(inmess,"nodata",6))==0)
      {
            inmess[0]='\0';
            strcpy(inmess,buffer);
      }
      else
```

```
strcpy(semess,buffer);
printf("\n\n\n\t\tThe command from CELL controller is:");
if((strncmp(inmess,"nodata",6))==0)
;
else
printf("%s",inmess);
printf("\n\n\t\tAnother command from CELL controller is:");
if((strncmp(semess,"nodata",6))==0)
;
else
printf("%s",semess);
printf("\n\n\n\tThe respond to the command can one of the
following:");
printf("\n\n\t1. Ready, 2. Busy, 3. Done.");
printf("\n\n\t4. Rob Feed Request, 5. CNC Feed Request.");
printf("\n\t6. Link Feed Request.");
printf("\n\n\t7. Pallet Problem, 8. Assembly Difficulty.");
printf("\n\t9. Kit Making Difficulty.");
printf("\n\n\t10. Go on.");
printf("\n\n\t Enter your choice[1-10]:");
scanf("%d",&choice);
gets(dumy);
if(choice>=1 && choice<=9)
{
if(choice==1)
{
      strcpy(command,ready);
      if(strncmp(inmess,"initia",6)==0)
      {
      inmess[0]='\0';
      strcpy(inmess,"nodata");
      }
}
else if(choice==2)
{
      strcpy(command,inmess);
      strncat(command,busy,4);
}
else if(choice==3)
{
      strcpy(command,inmess);
      strncat(command,done,4);
      inmess[0]='\0';
      strcpy(inmess,"nodata");
}
else if(choice==4)
{
      strcpy(command,rfeed);
}
else if(choice==5)
{
      strcpy(command,cfeed);
}
else if(choice==6)
```

```
        {
              strcpy(command,lfeed);
        }
        else if(choice==7)
        {
              problem[9]-1;
              strcpy(command,problem);
        }
        else if(choice==8)
        {
              problem[9]-2;
              strcpy(command,problem);
        }
        else if(choice==9)
        {
              problem[9]-3;
              strcpy(command,problem);
        }

        else
        ;
        do{
              temp=ff.iwrite(outf,command);
              if(temp==0)
              {
                    double x;
                    for(int t=0;t<5000;++t)
                    {
                    x=sqrt(t);
                    }
              }
        }while(temp==0);
        }
        else
        {
              if(choice==7)
              {
                    semess[0]='\0';
                    strcpy(semess,"nodata");
              }
        }
}while(1);
}
//This is the material handling system simulator.
#include <stdio.h>
#include <string.h>
#include <io.h>
#include "facinter.h"
main()
{
      char dumy[10];
      char problem[11];
      char inmess[30];
      char semess[30];
```

```
        char buffer[30];
        char command[30];
        char* outf="mwsin.dat";
        char* inf="mwsout.dat";
        char* done="done";
        char* busy="busy";
        char* ready="ready";
        facinter ff;
        int temp,choice;
        strcpy(problem,"problem    ");
        strcpy(inmess,"nodata");
        semess[0]='\0';
        buffer[0]='\0';
        command[0]='\0';
        system("d:");
do{
        system("cls");
        printf("\n\n\n\n\t\tThis is Machining Workstation.");
        strcpy(buffer,ff.iread(inf));
        if((strncmp(inmess,"nodata",6))==0)
        {
              inmess[0]='\0';
              strcpy(inmess,buffer);
        }
        else
        strcpy(semess,buffer);
        printf("\n\n\n\t\tThe command from CELL controller is:");
        if((strncmp(inmess,"nodata",6))==0)
        ;
        else
        printf("%s",inmess);
        printf("\n\n\t\tAnother command from CELL controller is:");
        if((strncmp(semess,"nodata",6))==0)
        ;
        else
        printf("%s",semess);
        printf("\n\n\n\tThe respond to the command can one of the
        following:");
        printf("\n\n\t1. Ready, 2. Busy, 3. Done, 4. Pallet Problem.");
        printf("\n\t5. Machine No.1 Problem, 6. Machine No.2 Problem, 7.
        Go on.");
        printf("\n\n\t Enter your choice[1-7]:");
        scanf("%d",&choice);
        gets(dumy);
        if(choice>=1 && choice<=6)
        {
        if(choice==1)
        {
              strcpy(command,ready);
              if(strncmp(inmess,"initia",6)==0)
              {
              inmess[0]='\0';
              strcpy(inmess,"nodata");
              }
```

```
        }
        else if(choice==2)
        {
              strcpy(command,inmess);
              strncat(command,busy,4);
        }
        else if(choice==3)
        {
              strcpy(command,inmess);
              strncat(command,done,4);
              inmess[0]='\0';
              strcpy(inmess,"nodata");
        }
        else if(choice==4)
        {
              problem[9]=1;
              strcpy(command,problem);
        }
        else if(choice==5)
        {
              problem[9]=2;
              strcpy(command,problem);
        }
        else if(choice==6)
        {
              problem[9]=3;
              strcpy(command,problem);
        }

        else
        ;
        do{
              temp=ff.iwrite(outf,command);
              if(temp==0)
              {
                    double x;
                    for(int t=0;t<5000;++t)
                    {
                    x=log10(t);
                    }
              }
        }while(temp==0);
        }
        else
        {
              if(choice==7)
              {
                    semess[0]='\0';
                    strcpy(semess,"nodata");
              }
        }
}while(1);
}
```

Appendix I. Main Driver of the Shop Controller.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <sound.h>
#include <bios.h>
#include <string.h>
#include <io.h>
#include <math.h>
#include "entity.h"
#include "report.h"
#include "omanage.h"
#include "pprog.h"
#include "inven.h"
#include "asrs.h"
#include "geninter.h"
#include "mwsint.h"
#include "awsint.h"
#include "mhsint.h"
#include "human.h"
main()
{
      void rawmat();
      void human();
      void updates();
      void initia();
      void status();
      void problem();
      initia();
      printf("\n\n\n\n\n\n\n\n");
      printf("\n\nIf you want to have access to the system,");
      printf("\nplease type 'monitor' or 'loading' or 'problem'.");
      printf("\nThe system will respond to you in a second.\n");
      printf("\n\n\n\n\n\n\n\n");
      system("d:");
      aface.inita();
      mface.initm();
      hface.ini();
do
{
      int univ;     //universal integer.
      int cycles=5000, uptime=250, dntime=250;
      int freq=3331;
      updates();
      //production rule checking.
      if(mblem)
      {
            sound_tone(cycles,uptime,dntime);
            printf("\n\n\n\n\n\n\n");
            if(mblem==1)
```

```
        printf("\nProblem:   Machining Station-->local storage.");
        else if(mblem==2)
        printf("\nProblem:   Machining Station-->machine 1.");
        else if(mblem==3)
        printf("\nProblem:   Machining Station-->machine 2.");
        else
        printf("\nProblem:   Machining Station.");
}
if(material)
{
        sound_tone(cycles,uptime,dntime);
        printf("\n\n\n\n\n");
        if(material==1)
        printf("\nProblem:   Material Request-->no raw robot
        base.");
        else if(material==2)
        printf("\nProblem:   Material Request-->no raw CNC base.");
        else if(material==3)
        printf("\nProblem:   Material Request-->no raw link.");
        else if(material==4)
        printf("\nProblem:   Material Request-->no empty pallet.");
        else
        ;
}
if(ablem)
{
        sound_tone(cycles,uptime,dntime);
        printf("\n\n\n\n\n");
        if(ablem==1)
        printf("\nProblem:   Assembly Station-->local storage.");
        else if(ablem==2)
        printf("\nProblem:   Assembly Station-->assembly
        difficulty.");
        else if(ablem==3)
        printf("\nProblem:   Assembly Station-->kit-making
        difficulty.");
        else
        printf("\nProblem:   Assembly Station.");
}
if(hblem)
{
        sound_tone(cycles,uptime,dntime);
        printf("\n\n\n\n\n");
        if(hblem==1)
        printf("\nProblem:   Material Handling-->clean conveyor.");
        else if(hblem==2)
        printf("\nProblem:   Material Handling-->conveyor not
        working");
        else if(hblem==3)
        printf("\nProblem:   Material Handling-->AS/RS not
        working.");
        else if(hblem==4)
        printf("\nProblem:   Material Handling-->machine vision.");
        else
```

```
        printf("\nProblem:   Material Handling.");
}
//orders manage part.
if(!procent)
{
        procent=orders.release();
        if(procent)
        {
                if(strncmp(procent->type,"robot",5)==0)
                {
                        orkrob=procent->quan;
                        orderr=orkrob;
                        distin=1;
                }
                else if(strncmp(procent->type,"cnc",3)==0)
                {
                        orkcnc=procent->quan;
                        orderc=orkcnc;
                        distin=2;
                }
                else
                {
                        entity* ttp;
                        ttp=procent;
                        procent=0;
                        delete ttp;
                        sound_beep(freq);
                        sound_beep(freq);
                        printf("\n\n\n\n\n");
                        printf("\n\nWe can not accept the order which
                        has a");
                        printf("\ntype name '%s'.",procent->type);
                        printf("This order is disposed.\n");
                }
        }
}
if(orderr>0)
{
        if(arobp>=orderr && mhswm==1)
        {
                if(orcoun==0)
                {
                        sound_beep(freq);
                        sound_beep(freq);
                        sound_beep(freq);
                        sound_beep(freq);
                        sound_beep(freq);
                        sound_beep(freq);
                        sound_beep(freq);
                        sound_beep(freq);
                        printf("\n\nOrder No. %d",procent->ord_num);
                        printf("    will be shifted to operator\n\n\n");
                        printf("\n\n\n\n\n");
                        fprintf(stdprn,"\n\nOrder No. %d",procent-
```

```
                        >ord_num);
                        fprintf(stdprn,"    will be shifted to
                        operator\n\n\n");
                        fprintf(stdprn,"\n\n\n\n\n\n");
                  }
                  univ-stor.goout(arob);
                  hface.rto(univ,1);
                  mhswm=2;
                  hcomm=8;
                  ++orcoun;
                  if(orcoun>=orderr)
                  {
                        entity* temp;
                        orderr=0;
                        orcoun=0;
                        arobp=0;
                        temp=procent;
                        procent=0;
                        rep.append(temp);
                        distin=0;
                  }
            }
      }
      if(orderc>0)
      {
            if(acncp>=orderc && mhswm==1)
            {
                  if(orcoun==0)
                  {
                        sound_beep(freq);
                        sound_beep(freq);
                        sound_beep(freq);
                        sound_beep(freq);
                        sound_beep(freq);
                        sound_beep(freq);
                        sound_beep(freq);
                        sound_beep(freq);
                        printf("\n\nOrder No. %d",procent->ord_num);
                        printf("    will be shifted to operator\n\n\n");
                        printf("\n\n\n\n\n");
                        fprintf(stdprn,"\n\nOrder No. %d",procent-
                        >ord_num);
                        fprintf(stdprn,"    will be shifted to
                        operator\n\n\n");
                        fprintf(stdprn,"\n\n\n\n\n\n\n");
                  }
                  univ-stor.goout(acnc);
                  hface.rto(univ,5);
                  mhswm=2;
                  hcomm=8;
                  ++orcoun;
                  if(orcoun>=orderc)
                  {
                        entity* temp;
```

```
                        orderc=0;
                        orcoun=0;
                        acncp=0;
                        temp=procent;
                        procent=0;
                        rep.append(temp);
                        distin=0;
                    }
                }
        }
//time part goes here.
        time(&finish);
        time(&stop);
        t=localtime(&finish);
        if(difftime(start,finish)>1800.0)
        {
                start=finish;
                status();
        }
        if(difftime(begin,stop)>3600.0)
        {
                begin=stop;
                printf("\n\n A production report will be printed.\n\n");
                rep.show();
        }
        if(t->tm_hour>=17)
        {
                printf("\nEnd of the day.\n");
                keystop=1;
                rep.show();
                rep.clear();
                status();
        }
        //operator's part.
        if(manreq==1)
        {
                human();
        }
        if(manreq==2 && mhswm==1)
        {
                rawmat();
        }
        if(manreq==3)
        {
                problem();
        }
if(distin==1)
{
        //maching work station part.
        if(mwswm==1)
        {
                if(mpkr==1)
                {
                        mface.millr();
```

```
                mwswm=2;
                mcomm=1;
        }
        else if(mpkr==0 && krobp>0 && mhswm==1)
        {
                univ=stor.goout(krob);
                hface.rtm(univ,3);
                mhswm=2;
                --krobp;
                hcomm=2;
                mpkr=2;
        }
        else
        ;
}
if(mwswm==3)
{
        if(mhswm==1 && awswm==1)
        {
                univ=apmr+apbr+apbl+apem;
                if(univ==0)
                {
                        if(krobp>0 || orkrob==0)
                        {
                                hface.mta();
                                mhswm=2;
                                hcomm=6;
                                mpkr=0;
                                apmr=2;
                                mwswm=1;
                        }
                }
        }
        else if(mhswm==1)
        {
                univ=stor.comein(mrob);
                hface.mtr(univ);
                mhswm=2;
                hcomm=4;
                mpkr=0;
                ++mrobp;
                mwswm=1;
        }
        else
        ;
}
if(awswm==1)
{
        if(apmr==1)
        {
                aface.assemr();
                awswm=2;
                acomm=1;
        }
```

```
else if(apem==1)
{
      aface.buildr();
      awswm=2;
      acomm=3;
}
else if(apbr==1)
{
      aface.loadr();
      awswm=2;
      acomm=5;
      asfeed=0;
}
else if(apbl==1)
{
      aface.loadl();
      awswm=2;
      acomm=7;
      asfeed=0;
}
else if(orkrob==0 && mrobp>0 && mhswm==1)
{
      univ=stor.goout(mrob);
      hface.rta(univ,2);
      mhswm=2;
      --mrobp;
      hcomm=1;
      apmr=2;
}
else if(krobp>0 && mhswm==1 && mrobp>0)
{
      univ=stor.goout(mrob);
      hface.rta(univ,2);
      --mrobp;
      mhswm=2;
      hcomm=1;
      apmr=2;
}
else if(asfeed==1 && mhswm==1)
{
      if(rrobp>0)
      {
            univ=stor.goout(rrob);
            hface.rta(univ,4);
            --rrobp;
            mhswm=2;
            hcomm=1;
            apbr=2;
      }
      else
      material=1;
}
else if(asfeed==3 && mhswm==1)
{
```

```
                if(rlinp>0)
                {
                        univ-stor.goout(rlin);
                        hface.rta(univ,9);
                        --rlinp;
                        mhswm-2;
                        hcomm-1;
                        apbl-2;
                }
                else
                material-3;
        }
        else if(orkrob>0 && mhswm--1)
        {
                if(emptp>0)
                {
                        univ-stor.goout(empt);
                        hface.rta(univ,10);
                        --emptp;
                        mhswm-2;
                        hcomm-1;
                        apem-2;
                }
                else
                material-4;
        }
        else
        ;
}
if(awswm--3 && mhswm--1)
{
        if(apmr--1)
        {
                apmr-0;
                awswm-1;
                univ-stor.comein(arob);
                hface.atr(univ);
                mhswm-2;
                ++arobp;
                hcomm-3;
        }
        else if(apem--1)
        {
                if(mwswm--1 && mpkr--0)
                {
                        apem-0;
                        awswm-1;
                        hface.atm();
                        mhswm-2;
                        hcomm-5;
                        mpkr-2;
                        --orkrob;
                }
                else
```

```
                {
                        apem=0;
                        awswm=1;
                        univ=stor.comein(krob);
                        hface.atr(univ);
                        mhswm=2;
                        hcomm=3;
                        ++krobp;
                        --orkrob;
                }
        }
        else if(apbr==1 || apbl==1)
        {
                if(apbr==1)
                apbr=0;
                if(apbl==1)
                apbl=0;
                awswm=1;
                univ=stor.comein(empt);
                hface.atr(univ);
                mhswm=2;
                hcomm=3;
                ++emptp;
        }
    }
    else
    ;
}
if(distin==2)
{
    //maching work station part.
    if(mwswm==1)
    {
        if(mpkc==1)
        {
                mface.millc();
                mwswm=2;
                mcomm=2;
        }
        else if(mpkc==0 && kcncp>0 && mhswm==1)
        {
                univ=stor.goout(kcnc);
                hface.rtm(univ,7);
                mhswm=2;
                --kcncp;
                hcomm=2;
                mpkc=2;
        }
        else
        ;
    }
    if(mwswm==3)
    {
        if(mhswm==1 && awswm==1)
```

```
        {
              univ=apmc+apbc+apbl+apem;
              if(univ==0)
              {
                    if(kcncp>0 || orkcnc==0)
                    {
                          hface.mta();
                          mhswm=2;
                          hcomm=6;
                          mpkc=0;
                          mwswm=1;
                          apmc=2;
                    }
              }
        }
        else if(mhswm==1)
        {
              univ=stor.comein(mcnc);
              hface.mtr(univ);
              mhswm=2;
              hcomm=4;
              mpkc=0;
              ++mcncp;
              mwswm=1;
        }
        else
        ;
}
if(awswm==1)
{
        if(apmc==1)
        {
              aface.assemc();
              awswm=2;
              acomm=2;
        }
        else if(apem==1)
        {
              aface.buildc();
              awswm=2;
              acomm=4;
        }
        else if(apbc==1)
        {
              aface.loadc();
              awswm=2;
              acomm=6;
              asfeed=0;
        }
        else if(apbl==1)
        {
              aface.loadl();
              awswm=2;
              acomm=7;
```

```
        asfeed=0;
    }
    else if(orkcnc==0 && mcncp>0 && mhswm==1)
    {
        univ=stor.goout(mcnc);
        hface.rta(univ,6);
        mhswm=2;
        --mcncp;
        hcomm=1;
        apmc=2;
    }
    else if(kcncp>0 && mhswm==1 && mcncp>0)
    {
        univ=stor.goout(mcnc);
        hface.rta(univ,6);
        --mcncp;
        mhswm=2;
        hcomm=1;
        apmc=2;
    }
    else if(asfeed==2 && mhswm==1)
    {
        if(rcncp>0)
        {
            univ=stor.goout(rcnc);
            hface.rta(univ,8);
            --rcncp;
            mhswm=2;
            hcomm=1;
            apbc=2;
        }
        else
        material=2;
    }
    else if(asfeed==3 && mhswm==1)
    {
        if(rlinp>0)
        {
            univ=stor.goout(rlin);
            hface.rta(univ,9);
            --rlinp;
            mhswm=2;
            hcomm=1;
            apbl=2;
        }
        else
        material=3;
    }
    else if(orkcnc>0 && mhswm==1)
    {
        if(emptp>0)
        {
            univ=stor.goout(empt);
            hface.rta(univ,10);
```

```
                --emptp;
                mhswm=2;
                hcomm=1;
                apem=2;
            }
            else
            material=4;
        }
        else
        ;
}
if(awswm==3 && mhswm==1)
{
        if(apmc==1)
        {
            apmc=0;
            awswm=1;
            univ=stor.comein(acnc);
            hface.atr(univ);
            mhswm=2;
            ++acncp;
            hcomm=3;
        }
        else if(apem==1)
        {
            if(mwswm==1 && mpkc==0)
            {
                apem=0;
                awswm=1;
                hface.atm();
                mhswm=2;
                hcomm=5;
                mpkc=2;
                --orkcnc;
            }
            else
            {
                apem=0;
                awswm=1;
                univ=stor.comein(kcnc);
                hface.atr(univ);
                mhswm=2;
                hcomm=3;
                ++kcncp;
                --orkcnc;
            }
        }
        else if(apbc==1 || apbl==1)
        {
            if(apbc==1)
            apbr=0;
            if(apbl==1)
            apbl=0;
            awswm=1;
```

```
                    univ=stor.comein(empt);
                    hface.atr(univ);
                    mhswm=2;
                    hcomm=3;
                    ++emptp;
              }
              else
              ;
       }
}
if(mhswm==3)
{
       if(hcomm==1)
       {
              mhswm=1;
              if(apmr==2)
              apmr=1;
              else if(apmc==2)
              apmc=1;
              else if(apbr==2)
              apbr=1;
              else if(apbl==2)
              apbl=1;
              else if(apbc==2)
              apbc=1;
              else if(apem==2)
              apem=1;
              else
              ;
       }
       else if(hcomm==2)
       {
              mhswm=1;
              if(mpkr==2)
              mpkr=1;
              else if(mpkc==2)
              mpkc=1;
              else
              ;
       }
       else if(hcomm==3 || hcomm==4)
       {
              mhswm=1;
       }
       else if(hcomm==5)
       {
              mhswm=1;
              if(mpkr==2)
              mpkr=1;
              if(mpkc==2)
              mpkc=1;
       }
       else if(hcomm==6)
       {
```

```
            mhswm=1;
            if(apmr==2)
            apmr=1;
            if(apmc==2)
            apmc=1;
      }
      else if(hcomm==7 || hcomm==8)
      {
            mhswm=1;
      }
      else
      ;
}
}while(keystop==0);
}
void problem()
{
      char mess[10];
      if(mblem)
      {
            printf("\n\n\n\n\n\n\n\n");
            printf("\n\tHave you fixed the following problem:");
            if(mblem==1)
            {
                  printf("\n\tProblem:   Machining Station-->local
                  storage.");
                  printf("\n\n\t   Anwser: yes or no. ==>");
                  gets(mess);
                  if(mess[0]=='y')
                  {
                        mface.mgoon();
                        mblem=0;
                  }
            }
            else if(mblem==2)
            {
                  printf("\nProblem:   Machining Station-->machine 1.");
                  printf("\n\n\t   Anwser: yes or no. ==>");
                  gets(mess);
                  if(mess[0]=='y')
                  {
                        mface.mgoon();
                        mblem=0;
                  }
            }
            else if(mblem==3)
            {
                  printf("\nProblem:   Machining Station-->machine 2.");
                  printf("\n\n\t   Anwser: yes or no. ==>");
                  gets(mess);
                  if(mess[0]=='y')
                  {
                        mface.mgoon();
                        mblem=0;
```

```
            }
        }
        else
        {
            printf("\nProblem:   Machining Station.");
            printf("\n\n\t   Anwser: yes or no. -->");
            gets(mess);
            if(mess[0]=='y')
            {
                mface.mgoon();
                mblem=0;
            }
        }
    }
    if(ablem)
    {
        printf("\n\n\n\n\n\n\n\n");
        printf("\n\tHave you fixed the following problem:");
        if(ablem==1)
        {
            printf("\nProblem:   Assembly Station-->local
            storage.");
            printf("\n\n\t   Anwser: yes or no. -->");
            gets(mess);
            if(mess[0]=='y')
            {
                aface.agoon();
                ablem=0;
            }
        }
        else if(ablem==2)
        {
            printf("\nProblem:   Assembly Station-->assembly
            difficulty.");
            printf("\n\n\t   Anwser: yes or no. -->");
            gets(mess);
            if(mess[0]=='y')
            {
                aface.agoon();
                ablem=0;
            }
        }
        else if(ablem==3)
        {
            printf("\nProblem:   Assembly Station-->kit-making
            difficulty.");
            printf("\n\n\t   Anwser: yes or no. -->");
            gets(mess);
            if(mess[0]=='y')
            {
                aface.agoon();
                ablem=0;
            }
        }
```

```
        else
        {
                printf("\nProblem:   Assembly Station.");
                printf("\n\n\t   Anwser: yes or no. ==>");
                gets(mess);
                if(mess[0]=='y')
                {
                        aface.agoon();
                        ablem=0;
                }
        }
}
if(hblem)
{
        printf("\n\n\n\n\n\n\n\n");
        printf("\n\tHave you fixed the following problem:");
        if(hblem==1)
        {
                printf("\nProblem:   Material Handling-->clean
                conveyor.");
                printf("\n\n\t   Anwser: yes or no. ==>");
                gets(mess);
                if(mess[0]=='y')
                {
                        hface.hgoon();
                        hblem=0;
                }
        }
        else if(hblem==2)
        {
                printf("\nProblem:   Material Handling-->conveyor not
                working");
                printf("\n\n\t   Anwser: yes or no. ==>");
                gets(mess);
                if(mess[0]=='y')
                {
                        hface.hgoon();
                        hblem=0;
                }
        }
        else if(hblem==3)
        {
                printf("\nProblem:   Material Handling-->AS/RS not
                working.");
                printf("\n\n\t   Anwser: yes or no. ==>");
                gets(mess);
                if(mess[0]=='y')
                {
                        hface.hgoon();
                        hblem=0;
                }
        }
        else if(hblem==4)
        {
```

```
                printf("\nProblem:   Material Handling-->machine
                vision.");
                printf("\n\n\t   Anwser: yes or no. -->");
                gets(mess);
                if(mess[0]=='y')
                {
                      hface.hgoon();
                      hblem=0;
                }
          }
          else
          {
                printf("\nProblem:   Material Handling.");
                printf("\n\n\t   Anwser: yes or no. -->");
                gets(mess);
                if(mess[0]=='y')
                {
                      hface.hgoon();
                      hblem=0;
                }
          }
     }
     manreq=0;
}
//this function updates the working memory.
void updates()
{
//calculate the processing time
double x;
for(int t=0;t<4000;++t)
{
     x=sqrt(t);
}
//get operator's request.
if(bioskey(1))
{
     gets(comtmp1);
     if(strncmp(comtmp1,"monitor",7)==0)
     manreq=1;
     else if(strncmp(comtmp1,"loading",7)==0)
     manreq=2;
     else if(strncmp(comtmp1,"problem",7)==0)
     manreq=3;
     else
     {
     ;
     }
}
     strcpy(comtmp1,mface.readmws());
     if(!comtmp1[0])
     ;
     else if(strncmp(comtmp1,"ready",5)==0)
     mwswm=1;
     else if(strncmp(comtmp1,"problem",7)==0)
```

```
        {
              mblem=comtmp1[9];
        }
        else if (mcomm==1)
        {
              if(strncmp(comtmp1,mmillr,9)==0)
              {
                    for(int i=0;i<9;++i)
                    comtmp2[i]=comtmp1[i+10];
                    if(strncmp(comtmp2,"busy",4)==0)
                    mwswm=2;
                    else if(strncmp(comtmp2,"done",4)==0)
                    mwswm=3;
              }
        }
        else if (mcomm==2)
        {
              if(strncmp(comtmp1,mmillc,9)==0)
              {
                    for(int i=0;i<9;++i)
                    comtmp2[i]=comtmp1[i+10];
                    if(strncmp(comtmp2,"busy",4)==0)
                    mwswm=2;
                    else if(strncmp(comtmp2,"done",4)==0)
                    mwswm=3;
              }
        }
        else
        ;
//assembly station.
        strcpy(comtmp1,aface.readaws());
        if(!comtmp1[0])
        ;
        else if(strncmp(comtmp1,"ready",5)==0)
              awswm=1;
        else if(strncmp(comtmp1,"problem",7)==0)
              {
              ablem=comtmp1[9];
              }
        else if(strncmp(comtmp1,"rfeed",5)==0)
              asfeed=1;
        else if(strncmp(comtmp1,"cfeed",5)==0)
              asfeed=2;
        else if(strncmp(comtmp1,"lfeed",5)==0)
              asfeed=3;
        else if (acomm==1)
        {
              if(strncmp(comtmp1,aasser,9)==0)
              {
                    for(int i=0;i<9;++i)
                    comtmp2[i]=comtmp1[i+10];
                    if(strncmp(comtmp2,"busy",4)==0)
                    awswm=2;
                    else if(strncmp(comtmp2,"done",4)==0)
```

```
                awswm=3;
        }
}
else if (acomm==2)
{
        if(strncmp(comtmp1,aassec,9)==0)
        {
                for(int i=0;i<9;++i)
                comtmp2[i]=comtmp1[i+10];
                if(strncmp(comtmp2,"busy",4)==0)
                awswm=2;
                else if(strncmp(comtmp2,"done",4)==0)
                awswm=3;
        }
}
else if (acomm==3)
{
        if(strncmp(comtmp1,abuilr,9)==0)
        {
                for(int i=0;i<9;++i)
                comtmp2[i]=comtmp1[i+10];
                if(strncmp(comtmp2,"busy",4)==0)
                awswm=2;
                else if(strncmp(comtmp2,"done",4)==0)
                awswm=3;
        }
}
else if (acomm==4)
{
        if(strncmp(comtmp1,abuilc,9)==0)
        {
                for(int i=0;i<9;++i)
                comtmp2[i]=comtmp1[i+10];
                if(strncmp(comtmp2,"busy",4)==0)
                awswm=2;
                else if(strncmp(comtmp2,"done",4)==0)
                awswm=3;
        }
}
else if (acomm==5)
{
        if(strncmp(comtmp1,aloadr,9)==0)
        {
                for(int i=0;i<9;++i)
                comtmp2[i]=comtmp1[i+10];
                if(strncmp(comtmp2,"busy",4)==0)
                awswm=2;
                else if(strncmp(comtmp2,"done",4)==0)
                awswm=3;
        }
}
else if (acomm==6)
{
        if(strncmp(comtmp1,aloadc,9)==0)
```

```
            {
                  for(int i=0;i<9;++i)
                  comtmp2[i]=comtmp1[i+10];
                  if(strncmp(comtmp2,"busy",4)==0)
                  awswm=2;
                  else if(strncmp(comtmp2,"done",4)==0)
                  awswm=3;
            }
      }
      else if (acomm==7)
      {
            if(strncmp(comtmp1,aloadl,9)==0)
            {
                  for(int i=0;i<9;++i)
                  comtmp2[i]=comtmp1[i+10];
                  if(strncmp(comtmp2,"busy",4)==0)
                  awswm=2;
                  else if(strncmp(comtmp2,"done",4)==0)
                  awswm=3;
            }
      }
      else
      ;
//material handling station.
      strcpy(comtmp1,hface.readmhs());
      if(!comtmp1[0])
      ;
      else if(strncmp(comtmp1,"ready",5)==0)
            mhswm=1;
            else if(strncmp(comtmp1,"problem",7)==0)
            {
            hblem=comtmp1[9];
            }
            else if (hcomm==1)
      {
            if(strncmp(comtmp1,mhsrta,6)==0)
            {
                  for(int i=0;i<9;++i)
                  comtmp2[i]=comtmp1[i+10];
                  if(strncmp(comtmp2,"busy",4)==0)
                  mhswm=2;
                  else if(strncmp(comtmp2,"done",4)==0)
                  mhswm=3;
            }
      }
      else if (hcomm==2)
      {
            if(strncmp(comtmp1,mhsrtm,6)==0)
            {
                  for(int i=0;i<9;++i)
                  comtmp2[i]=comtmp1[i+10];
                  if(strncmp(comtmp2,"busy",4)==0)
                  mhswm=2;
                  else if(strncmp(comtmp2,"done",4)==0)
```

```
                mhswm=3;
        }
}
else if (hcomm==3)
{
        if(strncmp(comtmp1,mhsatr,6)==0)
        {
                for(int i=0;i<9;++i)
                comtmp2[i]=comtmp1[i+10];
                if(strncmp(comtmp2,"busy",4)==0)
                mhswm=2;
                else if(strncmp(comtmp2,"done",4)==0)
                mhswm=3;
        }
}
else if (hcomm==4)
{
        if(strncmp(comtmp1,mhsmtr,6)==0)
        {
                for(int i=0;i<9;++i)
                comtmp2[i]=comtmp1[i+10];
                if(strncmp(comtmp2,"busy",4)==0)
                mhswm=2;
                else if(strncmp(comtmp2,"done",4)==0)
                mhswm=3;
        }
}
else if (hcomm==5)
{
        if(strncmp(comtmp1,mhsatm,6)==0)
        {
                for(int i=0;i<9;++i)
                comtmp2[i]=comtmp1[i+10];
                if(strncmp(comtmp2,"busy",4)==0)
                mhswm=2;
                else if(strncmp(comtmp2,"done",4)==0)
                mhswm=3;
        }
}
else if (hcomm==6)
{
        if(strncmp(comtmp1,mhsmta,6)==0)
        {
                for(int i=0;i<9;++i)
                comtmp2[i]=comtmp1[i+10];
                if(strncmp(comtmp2,"busy",4)==0)
                mhswm=2;
                else if(strncmp(comtmp2,"done",4)==0)
                mhswm=3;
        }
}
else if (hcomm==7)
{
        if(strncmp(comtmp1,mhsotr,6)==0)
```

```
            {
                for(int i=0;i<9;++i)
                comtmp2[i]=comtmp1[i+10];
                if(strncmp(comtmp2,"busy",4)==0)
                mhswm=2;
                else if(strncmp(comtmp2,"done",4)==0)
                mhswm=3;
            }
      }
      else if (hcomm==8)
      {
            if(strncmp(comtmp1,mhsrto,6)==0)
            {
                  for(int i=0;i<9;++i)
                  comtmp2[i]=comtmp1[i+10];
                  if(strncmp(comtmp2,"busy",4)==0)
                  mhswm=2;
                  else if(strncmp(comtmp2,"done",4)==0)
                  mhswm=3;
            }
      }
      else
      ;
}
//this file can be used to load raw material and empty pallet.
void rawmat()
{
      int temp;
      int pasrs;
      char choice[10];
      char mess1[10];
      char mess2[10];

      printf("\n\n\n\n\n\n\n");
      printf("\n\t1. Load a robot base pallet.");
      printf("\n\t2. Load a CNC base pallet.");
      printf("\n\t3. Load a link pallet.");
      printf("\n\t4. Load a empty pallet.");
      printf("\n\n\t First you should put your pallet on the
      conveyor,");
      printf("\n\t then select your choice [1-4]:");
      scanf("%d",&temp);
      gets(dumy);
      if(temp==1)
      {
            pasrs=stor.comein(rrob);
            hface.otr(pasrs);
            ++rrobp;
            if(material==1)
            material=0;
            mhswm=2;
            hcomm=7;
      }
      else if(temp==2)
```

```
      {
            pasrs=stor.comein(rcnc);
            hface.otr(pasrs);
            ++rcncp;
            if(material==2)
            material=0;
            mhswm=2;
            hcomm=7;
      }
      else if(temp==3)
      {
            pasrs=stor.comein(rlin);
            hface.otr(pasrs);
            ++rlinp;
            if(material==3)
            material=0;
            mhswm=2;
            hcomm=7;
      }
      else if(temp==4)
      {
            pasrs=stor.comein(empt);
            hface.otr(pasrs);
            ++emptp;
            if(material==4)
            material=0;
            mhswm=2;
            hcomm=7;
      }
      else
      ;
      printf("\nYou did give the right choice.");
      printf("\n\n\t Do you want to do more load raw material?
      [yes,no]:");
      gets(choice);
      if(choice[0]=='y')
      manreq=2;
      else
      manreq=0;
      return;initia.h".
//file initia.h
);
//Important notice: next section is the file "initia.h".
//file initia.h
// time initialization.
time_t start, finish,begin,stop;
struct tm *t;h".
//************************************
//object initialization.
// initialization.
inventory inven;
omanage orders;
pprog p;
report rep;
```

```
asrs stor;
mws mface;
mhs hface;
aws aface;

//*****************************************************************
//Working memory initialization.
//This order is the one that is in process.
entity* procent;
//These var defines the orders of robot or CNC.
int orderr, orderc;
int orcoun;                                    //dumy used to count sth.
//work station status.
int mhswm, mwswm, awswm;
//work station problem variable.
int ablem, mblem,hblem;

//pallet showing up at machine station and assembly station.
int mpkr,mpkc;
int apmr,apmc,apbr,apbc,apbl,apem;             //apem means empty pallet.

//robot and cnc pallets on the rack.
int arobp,mrobp,krobp,rrobp;
int acncp,mcncp,kcncp,rcncp,rlinp;
int emptp;                                     // empty pallet.

//orders of robots and cncs.
int orkrob,orkcnc;

//human function-8() should set this var to stop signal.
int keystop=0;

//operator request,either supervise or operator function.
int manreq;

//feeder request.
int asfeed;
//Material request.
int material=0;
//var to ditinguish robot and cnc.
int distin;
//*****************************************************************
//file initia.h
//Pallets on the rack.
char* mrob="macrob";
char* krob="kitrob";
char* rrob="rawrob";
char* mcnc="maccnc";
char* kcnc="kitcnc";
char* rcnc="rawcnc";
char* rlin="rawlin";
char* empt="emptyp";
char* arob="assrob";
char* acnc="asscnc";
```

```
//*****************************************************************
//file   comman.h
//This part defines some variables which will be used
//in updating working memory.
//first, commands for the maching workstation.
int mcomm;
char* mmillr="mirobot   ";
char* mmillc="micnc     ";
//second, commands for the assembly workstation.
int acomm;
char* aasser="asrobot   ";
char* aassec="ascnc     ";
char* abuilr="burobot   ";
char* abuilc="bucnc     ";
char* aloadr="lorobot   ";
char* aloadc="locnc     ";
char* aloadl="lolin     ";
//third, commands for the assembly workstation.
int hcomm;
char* mhsrta="ratoas    ";
char* mhsrtm="ratomi    ";
char* mhsatr="astora    ";
char* mhsmtr="mitora    ";
char* mhsatm="astomi    ";
char* mhsmta="mitoas    ";
char* mhsotr="optora    ";
char* mhsrto="ratoop    ";
char* mhsini="initia    ";
//to hold feedback temparorily.
char comtmp1[20];
char comtmp2[10];
//*****************************************************************
//Real initialization.
void missave()
{
      FILE* fp;
      entity* current;
      char ch='h';
      current=procent;
      if((fp=fopen("mislan.dat","w"))==0)
      {
            printf("\n\nCan not open file to save\n");
            return;
      }
      printf("\n\n Saving Mislaneous Data\n\n");
      if(current)
      {
            fprintf(fp,"%d",current->ord_num);
            for(int t=0; t<15; ++t)
            putc(current->type[t],fp);
            fprintf(fp,"%d",current->quan);
            putc(ch,fp);
            fprintf(fp,"%d",current->prior);
            putc(ch,fp);
```

```
}
else
{
        fprintf(fp,"%d",0);
        for(int t=0; t<15; ++t)
        putc(ch,fp);
        fprintf(fp,"%d",0);
        putc(ch,fp);
        fprintf(fp,"%d",0);
        putc(ch,fp);
}
fprintf(fp,"%d",hcomm);
putc(ch,fp);
fprintf(fp,"%d",mcomm);
putc(ch,fp);
fprintf(fp,"%d",acomm);
putc(ch,fp);
fprintf(fp,"%d",mhswm);
putc(ch,fp);
fprintf(fp,"%d",awswm);
putc(ch,fp);
fprintf(fp,"%d",mwswm);
putc(ch,fp);
fprintf(fp,"%d",mpkr);
putc(ch,fp);
fprintf(fp,"%d",mpkc);
putc(ch,fp);
fprintf(fp,"%d",apmr);
putc(ch,fp);
fprintf(fp,"%d",apmc);
putc(ch,fp);
fprintf(fp,"%d",apbr);
putc(ch,fp);
fprintf(fp,"%d",apbc);
putc(ch,fp);
fprintf(fp,"%d",apbl);
putc(ch,fp);
fprintf(fp,"%d",orkrob);
putc(ch,fp);
fprintf(fp,"%d",orkcnc);
putc(ch,fp);
fprintf(fp,"%d",manreq);
putc(ch,fp);
fprintf(fp,"%d",asfeed);
putc(ch,fp);
fprintf(fp,"%d",distin);
putc(ch,fp);
fprintf(fp,"%d",orderr);
putc(ch,fp);
fprintf(fp,"%d",orderc);
putc(ch,fp);
fprintf(fp,"%d",orcoun);
putc(ch,fp);
fclose(fp);
```

```
};
void misload()
{
      FILE* fp;
      entity* current;
      char ch;
      if((fp=fopen("mislan.dat","r"))==0)
      {
      printf("\n\nNo Mislaneous Data Saved in Memory");
      //real initialization.
      hcomm=0;
      acomm=0;
      mcomm=0;
      mhswm=2;
      mwswm=2;
      awswm=2;
      mpkr=0;
      mpkc=0;
      apmr=0;
      apmc=0;
      apbr=0;
      apbc=0;
      apbl=0;
      orkrob=0;
      orkcnc=0;
      manreq=0;
      asfeed=0;
      distin=0;
      orderr=0;
      orderc=0;
      orcoun=0;
      return;
      }
      else
      {
      printf("\n\nLoading Part Program\n\n");
      current=new entity;
      fscanf(fp,"%d",&current->ord_num);
      for(int t=0;t<15;++t)
      current->type[t]=fgetc(fp);
      fscanf(fp,"%d",&current->quan);
      ch=fgetc(fp);
      fscanf(fp,"%d",&current->prior);
      ch=fgetc(fp);
      if(current->ord_num)
      procent=current;
      else
      {
            procent=0;
            delete current;
      }
      fscanf(fp,"%d",&hcomm);
      ch=fgetc(fp);
      fscanf(fp,"%d",&mcomm);
```

```
        ch=fgetc(fp);
        fscanf(fp,"%d",&acomm);
        ch=fgetc(fp);
        fscanf(fp,"%d",&mhswm);
        ch=fgetc(fp);
        fscanf(fp,"%d",&awswm);
        ch=fgetc(fp);
        fscanf(fp,"%d",&mwswm);
        ch=fgetc(fp);
        fscanf(fp,"%d",&mpkr);
        ch=fgetc(fp);
        fscanf(fp,"%d",&mpkc);
        ch=fgetc(fp);
        fscanf(fp,"%d",&apmr);
        ch=fgetc(fp);
        fscanf(fp,"%d",&apmc);
        ch=fgetc(fp);
        fscanf(fp,"%d",&apbr);
        ch=fgetc(fp);
        fscanf(fp,"%d",&apbc);
        ch=fgetc(fp);
        fscanf(fp,"%d",&apbl);
        ch=fgetc(fp);
        fscanf(fp,"%d",&orkrob);
        ch=fgetc(fp);
        fscanf(fp,"%d",&orkcnc);
        ch=fgetc(fp);
        fscanf(fp,"%d",&manreq);
        ch=fgetc(fp);
        fscanf(fp,"%d",&asfeed);
        ch=fgetc(fp);
        fscanf(fp,"%d",&distin);
        ch=fgetc(fp);
        fscanf(fp,"%d",&orderr);
        ch=fgetc(fp);
        fscanf(fp,"%d",&orderc);
        ch=fgetc(fp);
        fscanf(fp,"%d",&orcoun);
        ch=fgetc(fp);
        }
        fclose(fp);
};
void initia()
{
        stor.load();
        p.load();
        rep.load();
        orders.load();
        //pallets initialization.
        mrobp=stor.check(mrob);
        krobp=stor.check(krob);
        rrobp=stor.check(rrob);
        mcncp=stor.check(mcnc);
        kcncp=stor.check(kcnc);
```

```
        rcncp=stor.check(rcnc);
        rlinp=stor.check(rlin);
        emptp=stor.check(empt);
        arobp=stor.check(arob);
        acncp=stor.check(acnc);
        misload();
        time(&start);
        time(&begin);
}
void status()
{
        system("c:");
        stor.save();
        p.save();
        rep.save();
        orders.save();
        missave();
        system("d:");
}
```

Appendix J. Three Automatic Cell Simulators.

```
//This is the automatic simulator for the machining cell.
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <stdlib.h>
#include <io.h>
#include "facinter.h"
#include "subrou.h"
main()
{
      int base, rang, chan;
      int flag, recog, diffp;
      char dumy[10];
      char problem[11];
      char inmess[30];
      char semess[30];
      char buffer[30];
      char command[30];
      char* outf="mwsin.dat";
      char* inf="mwsout.dat";
      char* done="done";
      char* busy="busy";
      char* ready="ready";
      void timdel(int, int);
      int opport(int);
      facinter ff;
      int temp,choice;
      strcpy(problem,"problem   ");
      strcpy(inmess,"nodata");
      semess[0]='\0';
      buffer[0]='\0';
      command[0]='\0';
      base=0;
      chan=0;
      rang=0;
      flag=0;
      srand(2);
      system("d:");
do
{
      system("cls");
      printf("\n\n\n\n\t\tThis is Machining Workstation.");
      strcpy(buffer,ff.iread(inf));
      if((strncmp(inmess,"nodata",6))==0)
      {
            inmess[0]='\0';
            strcpy(inmess,buffer);
      }
      else
```

```
strcpy(semess,buffer);
printf("\n\n\n\t\tThe command from CELL controller is:");
if((strncmp(inmess,"nodata",6))==0)
;
else
printf("%s",inmess);
printf("\n\n\t\tAnother command from CELL controller is:");
if((strncmp(semess,"nodata",6))==0)
;
else
printf("%s",semess);
printf("\n\n\n\tThe respond to the command can one of the
following:");
printf("\n\n\t1. Ready, 2. Busy, 3. Done, 4. Pallet Problem.");
printf("\n\t5. Machine No.1 Problem, 6. Machine No.2 Problem, 7.
Go on.");
if((strncmp(inmess,"initia",6))==0)
{
      strcpy(command,ready);
      inmess[0]='\0';
      strcpy(inmess,"nodata");
      do
      {
      temp=ff.iwrite(outf,command);
      if(temp==0)
      {
            double x;
            for(int t=0;t<5000;++t)
            {
            x=sqrt(t);
            }
      }
      }while(temp==0);
}
if (flag==1)
{
      if((strncmp(semess,"nodata",6))==0)
      timdel(8,0);
      else
      {
            semess[0]='\0';
            strcpy(semess,"nodata");
            flag=2;
      }
}

if((strncmp(inmess,"nodata",6))==0)
{
timdel(11,0);
}
else
{
      if(flag==0)
      {
```

```
        recog=opport(10);
        if(recog==1)
        {
              flag=1;
              diffp=opport(30);
              if(diffp==1)
              {
                    problem[9]=1;
                    strcpy(command, problem);
              }
              else
              {
                    diffp=opport(30);
                    if(diffp==1)
                    {
                          problem[9]=2;
                          strcpy(command, problem);
                    }
                    else
                    {
                          problem[9]=3;
                          strcpy(command, problem);
                    }
              }
              do
              {
                    temp=ff.iwrite(outf,command);
                    if(temp==0)
                    {
                          double x;
                          for(int t=0;t<5000;++t)
                          {
                          x=sqrt(t);
                          }
                    }
              }while(temp==0);
        } //recog close.
        else
        flag=0;
  } //flag=0 close here.
  if(flag==0 || flag==2)
  {
        timdel(100,10);
        strcpy(command,inmess);
        strncat(command,done,4);
        inmess[0]='\0';
        strcpy(inmess,"nodata");
        do
        {
        temp=ff.iwrite(outf,command);
        if(temp==0)
        {
              double x;
              for(int t=0;t<5000;++t)
```

```
                        {
                        x=sqrt(t);
                        }
                    }
                    }while(temp==0);
                }
        );//else ends first level.
}while(1);
}

//This is the automatic simulator for the assembly cell.
//file afeed.cpp
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <stdlib.h>
#include <io.h>
#include "facinter.h"
#include "subrou.h"
main()
{
        int base,rang,chan;
        int flag, recog,diffp;
        char dumy[10];
        char problem[11];
        char inmess[30];
        char semess[30];
        char buffer[30];
        char command[30];
        char* outf="awsin.dat";
        char* inf="awsout.dat";
        char* done="done";
        char* busy="busy";
        char* ready="ready";
        char* rfeed="rfeed";
        char* cfeed="cfeed";
        char* lfeed="lfeed";
        void timdel(int,int);
        int opport(int);
        facinter ff;
        int temp,choice;
        strcpy(problem,"problem   ");
        strcpy(inmess,"nodata");
        semess[0]='\0';
        buffer[0]='\0';
        command[0]='\0';
        base=0;
        rang=0;
        flag=0;
        chan=0;
        srand(4);
        system("d:");
do
{
```

```
system("cls");
printf("\n\n\n\n\t\tThis is Assembly Workstation.");
strcpy(buffer,ff.iread(inf));
if((strncmp(inmess,"nodata",6))==0)
{
      inmess[0]='\0';
      strcpy(inmess,buffer);
}
else
strcpy(semess,buffer);
printf("\n\n\n\t\tThe command from CELL controller is:");
if((strncmp(inmess,"nodata",6))==0)
;
else
printf("%s",inmess);
printf("\n\n\t\tAnother command from CELL controller is:");
if((strncmp(semess,"nodata",6))==0)
;
else
printf("%s",semess);
printf("\n\n\n\tThe respond to the command can one of the
following:");
printf("\n\n\t1. Ready, 2. Busy, 3. Done.");
printf("\n\n\t4. Rob Feed Request, 5. CNC Feed Request.");
printf("\n\t6. Link Feed Request.");
printf("\n\n\t7. Pallet Problem, 8. Assembly Difficulty.");
printf("\n\t9. Kit Making Difficulty.");
printf("\n\n\t10. Go on.");
if((strncmp(inmess,"initia",6))==0)
{
      strcpy(command,ready);
      inmess[0]='\0';
      strcpy(inmess,"nodata");
      do
      {
      temp=ff.iwrite(outf,command);
      if(temp==0)
      {
            double x;
            for(int t=0;t<5000;++t)
            {
            x=sqrt(t);
            }
      }
      }while(temp==0);
}

if (flag==1)
{
      if((strncmp(semess,"nodata",6))==0)
      timdel(8,0);
      else
      {
            semess[0]='\0';
```

```
                strcpy(semess,"nodata");
                flag=2;
        }
}

if((strncmp(inmess,"nodata",6))==0)
{
timdel(11,0);
}
else
{
        if(flag==0)
        {
                recog=opport(20);
                if(recog==1)
                {
                        flag=1;
                        diffp=opport(16);
                        if(diffp==1)
                        {
                                problem[9]=1;
                                strcpy(command, problem);
                        }
                        else if((diffp=opport(16))==1)
                        {       problem[9]=2;
                                strcpy(command, problem);
                        }
                        else if((diffp=opport(16))==1)
                        {
                                problem[9]=3;
                                strcpy(command, problem);
                        }
                        else if((diffp=opport(16))==1)
                        {
                                strcpy(command, rfeed);
                        }
                        else if((diffp=opport(16))==1)
                        {
                                strcpy(command, cfeed);
                        }
                        else
                        {
                                strcpy(command, lfeed);
                        }
                        do
                        {
                                temp=ff.iwrite(outf,command);
                                if(temp==0)
                                {
                                        double x;
                                        for(int t=0;t<5000;++t)
                                        {
                                        x=sqrt(t);
                                        }
```

```
                        }
                    }while(temp==0);
                } //recog close.
                else
                flag=0;
            } //flag=0 close here.
            if(flag==0 || flag==2)
            {
                timdel(100,10);
                strcpy(command,inmess);
                strncat(command,done,4);
                inmess[0]='\0';
                strcpy(inmess,"nodata");
                do
                {
                temp=ff.iwrite(outf,command);
                if(temp==0)
                {
                    double x;
                    for(int t=0;t<5000;++t)
                    {
                    x=sqrt(t);
                    }
                }
                }while(temp==0);
            }
        };//else ends first level.
}while(1);
}
//This is the automatic simulator for the material handling system.
//file hfeed.cpp
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <stdlib.h>
#include <io.h>
#include "facinter.h"
#include "subrou.h"
main()
{
    int base, rang, chan;
    int flag, recog, diffp;
    char dumy[10];
    char problem[11];
    char inmess[30];
    char semess[30];
    char buffer[30];
    char command[30];
    char* outf="mhsin.dat";
    char* inf="mhsout.dat";
    char* ready="ready";
    char* done="done";
    char* busy="busy";
    void timdel(int, int);
```

```
        int opport(int);
        facinter ff;
        int temp,choice;
        strcpy(problem,"problem   ");
        strcpy(inmess,"nodata");
        semess[0]='\0';
        buffer[0]='\0';
        command[0]='\0';
        base=0;
        rang=0;
        flag=0;
        chan=0;
        srand(6);
        system("d:");
do
{
        system("cls");
        printf("\n\n\n\n\t\tThis is Material Handling. ");
        strcpy(buffer,ff.iread(inf));
        if((strncmp(inmess,"nodata",6))==0)
        {
              inmess[0]='\0';
              strcpy(inmess,buffer);
        }
        else
        strcpy(semess,buffer);
        printf("\n\n\n\t\tThe command from CELL controller is:");
        if((strncmp(inmess,"nodata",6))==0)
        ;
        else
        printf("%s",inmess);
        printf("\n\n\t\tAnother command from CELL controller is:");
        if((strncmp(semess,"nodata",6))==0)
        ;
        else
        printf("%s",semess);
        printf("\n\n\n\tThe respond to the command can one of the
        following:");
        printf("\n\n\t1. Ready, 2. Busy, 3. Done.");
        printf("\n\n\t4. Clear Conveyor, 5. Conveyor Problem.");
        printf("\n\t6. AS/RS Problem, 7. Machine Vision Problem.");
        printf("\n\n\t8. Go on.");
        if((strncmp(inmess,"initia",6))==0)
        {
              strcpy(command,ready);
              inmess[0]='\0';
              strcpy(inmess,"nodata");
              do
              {
              temp=ff.iwrite(outf,command);
              if(temp==0)
              {
                    double x;
                    for(int t=0;t<5000;++t)
```

```
                {
                x=sqrt(t);
                }
        }
        }while(temp==0);
}
if (flag==1)
{
        if((strncmp(semess,"nodata",6))==0)
        timdel(8,0);
        else
        {
                semess[0]='\0';
                strcpy(semess,"nodata");
                flag=2;
        }
}

if((strncmp(inmess,"nodata",6))==0)
{
timdel(11,0);
}
else
{
       if(flag==0)
       {
              recog=opport(10);
              if(recog==1)
              {
                     flag=1;
                     diffp=opport(25);
                     if(diffp==1)
                     {
                            problem[9]=1;
                            strcpy(command, problem);
                     }
                     else if((diffp=opport(25))==1)
                     {
                            problem[9]=2;
                            strcpy(command, problem);
                     }
                     else if((diffp=opport(25))==1)
                     {
                            problem[9]=3;
                            strcpy(command, problem);
                     }
                     else
                     {
                            problem[9]=4;
                            strcpy(command, problem);
                     }
                     do
                     {
                            temp=ff.iwrite(outf,command);
```

```
                        if(temp==0)
                        {
                              double x;
                              for(int t=0;t<5000;++t)
                              {
                              x=sqrt(t);
                              }
                        }
                  }while(temp==0);
            } //recog close.
            else
            flag=0;
      } //flag=0 close here.
      if(flag==0 || flag==2)
      {
            timdel(15,5);
            strcpy(command,inmess);
            strncat(command,done,4);
            inmess[0]='\0';
            strcpy(inmess,"nodata");
            do
            {
            temp=ff.iwrite(outf,command);
            if(temp==0)
            {
                  double x;
                  for(int t=0;t<5000;++t)
                  {
                  x=sqrt(t);
                  }
            }
            }while(temp==0);
      }
};//else ends first level.
}while(1);
}
// This is file "subrou.h"
// time function
void timdel (int base, int rang)
{
      int rantim;
      double x, rr;
      rantim=rand();
      rr=rantim/32767.0;
      rr=floor(rr*rang+0.5+base);
      rantim=rr;
      for (int i=0;i<rantim;++i)
      {
            for ( int t=0;t<400;++t)
            {
            x=sqrt(t);
            }
      }
}
```

```
//chan is the percentage
int opport(int chan)
{
      double dt;
      int timch;
      timch=rand();
      dt=timch/32767.0;
      dt=floor(dt*100);
      timch=dt;
      if(timch < chan)
      return 1;
      else
      return 0;
}
```

The vita has been removed from the scanned document