THE DEVELOPMENT OF LOW LEVEL COMMUNICATION INTERFACES FOR GENERIC WORK CELL CONTROL

by

Angela Nadine Ridgway

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 and Systems Engineering

APPROVED:

M. P. Deisenroth, Chairman

O. K. Eyada

R. J. Reasor

August 1991

Blacksburg, Virginia

ACKNOWLEDGEMENTS

Thanks to Dr. Michael Deisenroth for all the confidence in my work and me as a person. I couldn't ask for a better advisor or friend.

Thanks also to Dr. Eyada, Joni, Simon, Mike, Lori, and Benita.

TABLE OF CONTENTS

1.0 INTRODUCTION 1

1.1 Problem Statement 1

1.2 Objective 2

2.0 LITERATURE REVIEW 3

2.1 Origins and History 3

2.2 Hierarchy and Cell Structure 4

2.3 Cell Control Problems and Needs 4

2.4 Cell Control Functions 7

2.4.1 Communication 8

2.4.2 Control 10

2.4.3 Information Management 11

2.5 Cell Control Approaches 12

2.5.1 Customized Control 12

2.5.2 Generic Control 13

2.5.2.1 Communications Approaches 13

2.5.2.2 Software Aspects 17

2.6 Available Systems and Packages 19

2.6.1 Vendor Supplied Hardware Platforms 20

2.6.2 Standard Hardware Platforms 20

2.7 Communication Protocol Schemes 21

2.7.1 Manufacturing Automation Protocol 21

2.7.2 Distributed Automation Edition 22

2.8 Literature Summary 23

3.0 RESEARCH METHODOLOGY 24

4.0 RESEARCH ENVIRONMENTS AND DEVICES 27

4.1 DOS Environment 28

4.1.1 Programming Environment 28

4.1.1.1 Functions 28

4.1.2 Serial Communications 29

4.2 OS/2 Environment 29

4.2.1 Programming Environment 30

4.2.1.1 Processes 30

4.2.1.2 Threads 30

4.2.1.3 Time Slices 31

4.2.1.4 Semaphores 31

4.2.1.5 Serial Communications 32

4.3 Realtime Interface Co-Processor 32

4.3.1 Operating Environment 33

4.3.1.2 Tasks 33

4.3.1.2 Memory 33

4.3.1.3 System Unit - RIC Communications 34

4.3.1.4 Communications 36

4.3.2 DOS and OS/2 System Unit Environments 36

4.4 Programmable Logic Controller 37

4.4.1 Communications Interface 37

4.4.2 Memory Types 38

4.4.2.1 Ladder Memory 39

4.4.2.2 Scratch Pad Memory 39

4.4.2.3 V Memory 39

5.0 SOFTWARE DEVELOPMENT 40

5.1 Development Approach 40

5.2 TI435 PLC Functions 40

5.3 Operating System Code 44

5.3.1 COMM DOS Code 47

5.3.2 COMM OS/2 Code 47

5.3.3 RIC Code 49

5.4 Commonality 51

5.5 Code Overview 52

6.0 OTHER FACTORY DEVICES 54

6.1 Robots 54

6.1.2 IBM 7545 Robot 54

6.1.2.1 Read 55

6.1.2.2 Execute 55

6.1.2.3 New Program 56

6.1.2.4 Control 56

6.1.2.5 Teach 57

6.1.2.6 Robot Initiated Communications 57

6.1.2 Unimation PUMA 560 Robot 57

6.2 NC Machines 59

6.2.1 Dyna 2200/2400 Milling Machine 59

6.2.1.1 Upload 59

6.2.1.2 Download 60

6.2.2 HURCO BMC-20 Milling Machine 60

7.0 FRAMEWORK 61

7.1 Cell Control Function Categories 61

7.2 Device vs. Environment 63
7.2.1 Environment 63
7.2.2 Device 65
7.3 Data Framework 66
7.3.1 Function Parameters Structure 68
7.3.2 Environment Parameters Structure 70
7.3.3 Device Parameters Structure 70
7.3.4 Communication Parameters Structure 73
7.4 Code Framework 75
7.4.1 Cell Control Category Functions 75
7.4.2 Environment Code 75
7.4.2.1 Port Communications Code 76
7.4.2.2 Event Management Code 76
7.4.2.3 Resource Management Code 79
7.4.3 Device Code 79
7.4.3.1 Protocol Code 79
7.4.3.2 Data Manipulation Code 81
7.4.3.3 Control Function Support Code 82
7.5 Framework Overview 82
7.5.1 Control Functions 82
7.5.2 Data Overview 83
7.5.3 Code Overview 83
7.6 Application Example 86
7.6.1 Control Functions Categories 87
7.6.2 Data Framework 87
7.6.3 Code Framework 92
7.6.4 Example Summary 95

7.7 Benefits 97

8.0 CONCLUSION 99

8.1 Results 99

8.1 Recommendations 100

9.0 REFERENCES 102

VITA 105

LIST OF FIGURES

Figure 2.1 Plant Control Hierarchy 5

Figure 2.2 Decentralized vs. Centralized Control 6

Figure 2.3 Minimum Functionality Cell Controller 9

Figure 2.4 Communication Software Layers 15

Figure 2.5 Cell Controller Communication Model 16

Figure 7.1 Environment vs. Device Dependent Aspects 64

Figure 7.2 High Level Generic Structure 67

Figure 7.3 Function Parameters Structure 69

Figure 7.4 Environment Parameters Structure 71

Figure 7.5 Device Parameters Structure 72

Figure 7.6 Communications Parameters Structure 74

Figure 7.7 Data Framework 84

Figure 7.8 Code Framework 85

LIST OF TABLES

Table 4.1 System Unit - RIC Dialogue 35

Table 5.1 TI435 PLC Cell Control Routines 42

Table 5.2 TI435 PLC Cell Control Sub-functions 43

Table 5.3 TI435 PLC Cell Control Data Conversion Functions 45

Table 5.4 TI435 PLC Cell Control Protocol Functions 46

Table 5.5 TI435 PLC Cell Control Communications Functions 48

Table 7.1A Generic vs. Specific Communications Aspects 77

Table 7.1B Typical Communications Functions Names 77

Table 7.2A Generic vs. Specific Event Management Aspects 78

Table 7.2B Typical Event Management Function Names 78

Table 7.3A Generic vs. Specific Resource Management Aspects 80

Table 7.3B Typical Resource Management Function Names 80

Table 7.4 Example -- Control Function Categories 88

Table 7.5 Example -- High Level Generic Structure 89

Table 7.5A Example -- Function Parameters Structure 89

Table 7.5B Example -- Environment Parameters Structure 90

Table 7.5C Example -- Device Parameters Structure 91

Table 7.5D Example -- Environment Parameters Structure 91

Table 7.6A Example -- Function Category Code 93

Table 7.6B Example -- Environment Code 94

Table 7.6C Example -- Device Code 96

1.0 INTRODUCTION

As the desire for factory automation increases, so has the need to integrate machinery within the factory. More specifically, this integration is gaining importance in the area of manufacturing work cells [7]. A work cell can be defined as "a group of factory floor devices designed to work together to perform a related series of manufacturing tasks" [6]. Basically, four types of devices are present in a cell: numerical control (NC) machines which usually perform the major operations; inspection mechanisms; material handling devices; and a robot which loads and unloads parts to and from other devices in the cell [38]. Even though a cell may exist on the factory floor, it needs integration to make it a "cohesive unit" so that its performance adds to the plant performance [6].

By introducing work cells on the plant floor, the need for work cell control arises. Work cell control can be defined as "the act of coordinating multiple intelligent devices in a production environment in order to maximize the flexibility and productivity of the machines and people in that environment" [11].

1.1 Problem Statement

Many ideas exist about what functions a cell controller should perform and how it should interact with its environment. Because there are so many different ideas, development of cell controllers typically has not followed any standard format. The functions utilized by the cell controller may vary depending on the type of machinery but similar tasks are usually performed. The complexity of the cell

controller increases due to the differences in functional capability caused by machine intelligence or vendors specifications. The functions and structure of the cell controller need to be generalized to provide a common base but they must be flexible enough to facilitate the differences in factory machines.

1.2 Objective

The objective of this research was to create a framework which can be followed when developing low level machine specific cell control communications. The framework would assist the user in defining and structuring the information and functions associated with a particular device and operating environment. This framework will act as a guide in the creation of generic cell control communication functions. The term generic refers to the structure of the functions, the commonality between devices, and the interface to the user.

Even though different classes of factory devices perform different operations, similar cell control functions exist and should be accessed in a similar manner. The purpose of the framework is to act as a guide in the development of low level base routines which interface to various classes of factory devices. It is impossible to create a completely generic base which will interact with every device. However, it is possible to develop this base following a structured format which facilitates generic work cell control. The objective of the research was to create this framework. This will allow a higher level cell controller to be developed in a similar manner.

2.0 LITERATURE REVIEW

Extensive literature is available on all aspects of cell control. The scope of the literature review covers origin and history, control structure, problems, functionality, approaches, available control packages, and communication and protocol schemes.

2.1 Origins and History

Cell control is a relatively new technology, being less than ten years old. However, the concept of the manufacturing work cell itself is somewhat older [32]. The majority of the first cell controllers were programmable or numerical controllers used to coordinate several machines. Because these controllers usually utilized some type of discrete input/output, they worked well for coordination; but communications to other intelligent devices and data manipulation capabilities were poor. These controllers were generally used as local cell controllers, without integration to other systems [28].

General Motors Corporation and Digital Equipment Corporation were the forerunners in implementing cell control. GM used computers to supervise programmable logic controllers (PLCs) in their plants before 1982. Allen-Bradley Company was the first to use a specialized computer for these purposes. Personal computers have since been utilized extensively for monitoring PLCs. Currently, PLC monitoring is the most widely used application of cell control, but this domain is expanding quickly [32].

2.2 Hierarchy and Cell Structure

Work cell control is usually set up in a hierarchical fashion. This often stems from the plant wide control hierarchy. Usually, control is distributed from the higher plant floor level to the device level through the cell controller. This location in the hierarchy is shown in Figure 2.1 [14, 17]. The controller may communicate upward in the hierarchy with an area or factory controller or may act locally on its own. Downward communication exists with a vast group and variety of devices. The intelligence of these devices also varies. Communication with the intelligent devices is more complex than with simpler devices such as actuators, sensors, etc. [33].

2.3 Cell Control Problems and Needs

The problems and needs of cell control are almost as limitless as the number and kinds of devices which can be present in a cell. This limitless number of devices is the major source of problems and the major reason for utilizing a cell. Without the cell, control of the devices can become very complex. By creating the cell and the controller, the difficulty in creating control programs is reduced drastically from that of decentralized systems. Figure 2.2 illustrates the decentralized structure compared to the much simpler centralized, cell approach. The amount of communications and program interfacing needed to control the cell drastically decreases [38].

Another problem that arises due to the number of devices is the communication interfaces. Because no standard communication protocol has been

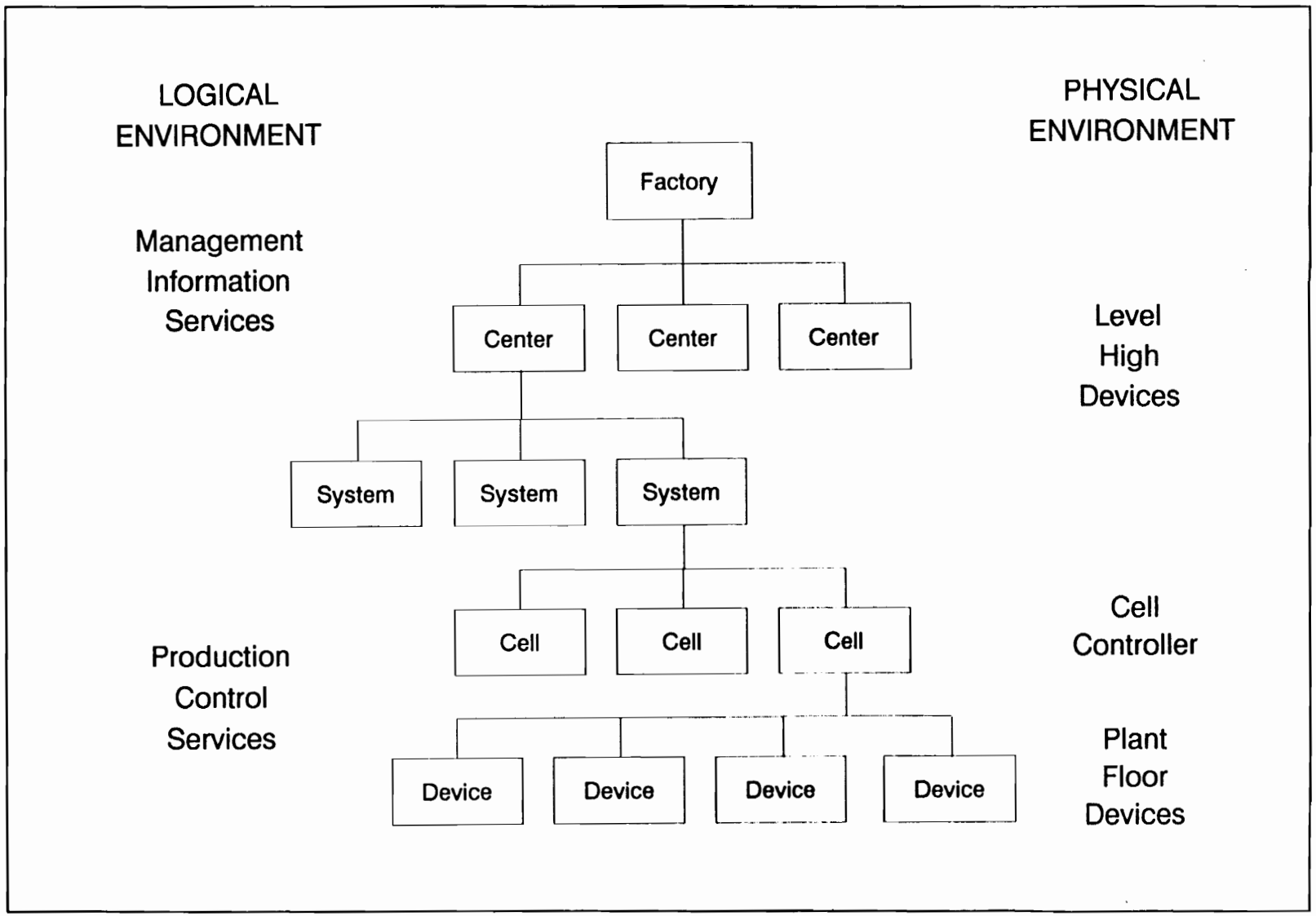


Figure 2.1 Plant Control Hierarchy (Adapted from [14, 17])

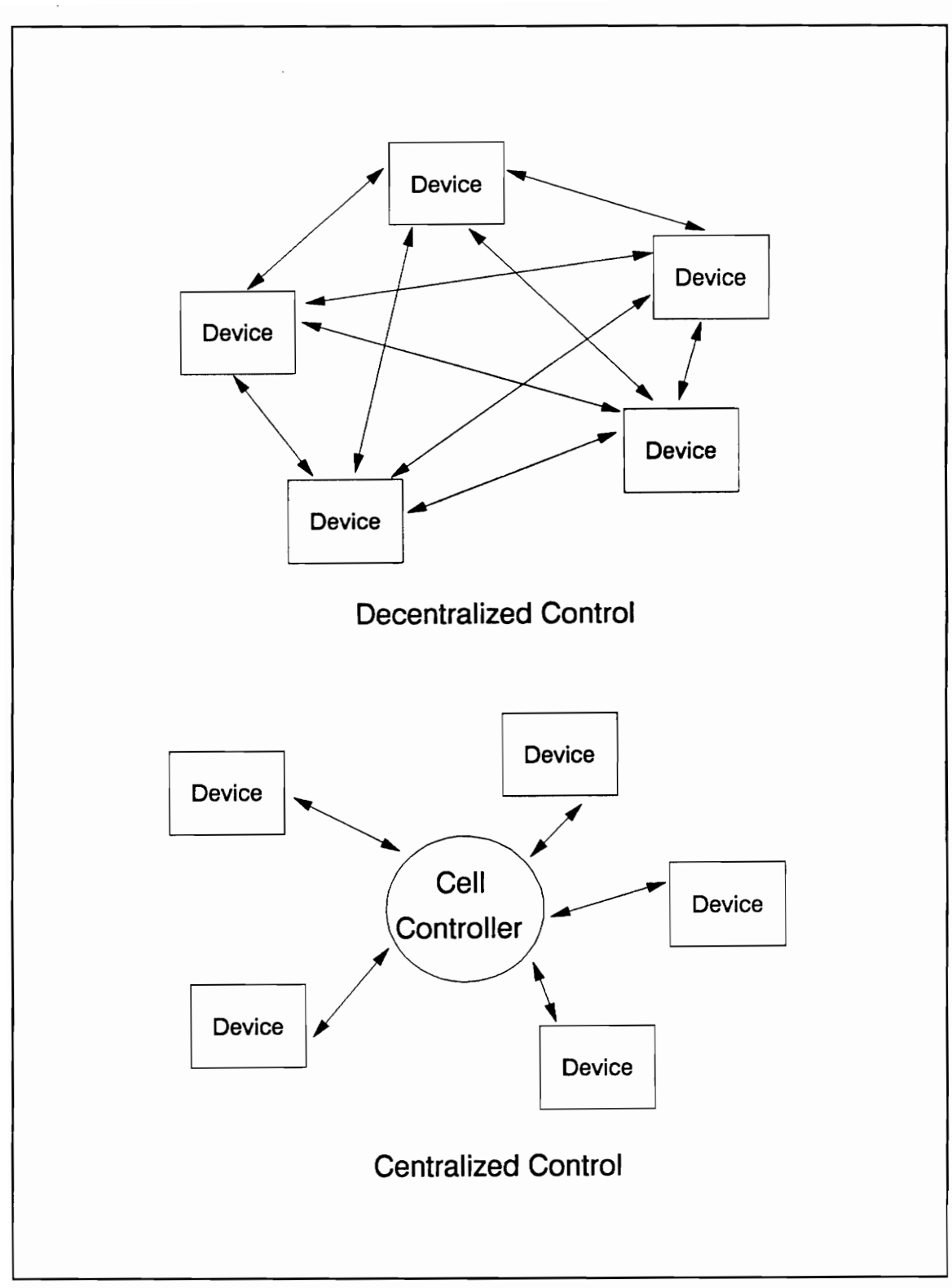


Figure 2.2 Decentralized vs. Centralized Control [38]

accepted, each device usually has its own. Therefore unique interfaces must be created which supports each one [32]. Effective control software must accommodate the vast assortment of environments and support communication to each [15, 28].

Cell control software has typically been custom coded, application specific, and designed to run on costly mainframe systems. The result has been the control of islands of automation which are difficult to integrate with other factory systems. Control software needs to be created on platforms which support application specific configurations, but allow expansion without recoding, and reduced initial and maintenance expenses [5]. A well planned cell controller utilizes standard and flexible software. This allows functionality in many environments and incremental expansion as needed [28].

2.4 Cell Control Functions

The desired functions of a cell controller vary depending on its application. The National Electrical Manufacturers Association (NEMA) has identified a "minimum functionality cell controller." Figure 2.3 depicts the core functions as (1) communication of "production information with either an operator, a higher level device, or both," (2) coordination and supervision of "at least one other independent production device controller which handles machine or process inputs/outputs," and (3) local storage and retrieval of production information. These attributes are guidelines for standardization [14]. The cell controller's primary function is to provide a "migration path from low to high levels of functionality." The functions of the cell controller should support this [15].

For the purposes of this literature review, cell control functions have been divided into three categories based on NEMA's minimum cell requirements: (1) communication, (2) control, and (3) information management.

2.4.1 COMMUNICATION

Communication must occur for the cell controller to perform any tasks. Referring back to Figure 2.1 which illustrates the control hierarchy, the cell controller must communicate to the levels above and below itself [14]. In addition, communication to equivalent layers of the hierarchy may be necessary. The cell controller is the means by which communication between the factory floor devices and the other levels of the hierarchy takes place. This communication link must support all functions of the cell controller [6].

The level above the cell may be a host computer or an operator [7, 14]. In either case, the cell controller is the link to the outside world [4]. At the equivalent level of the hierarchy, cell to cell communication may be necessary [7]. Obviously, communication must exist to each device below the cell in the hierarchy and the cell controller must have the ability to communicate with each one [7, 15, 28, 33].

Depending on the control structure, communication may or may not exist between some of the objects specified. In a completely hierarchical structure, communication exists only between a specific level and the levels above and below it. This follows for the devices within the cell. "If the full benefits of centralized cell control are to be realized, then no communications between devices are acceptable, not even those of discrete realtime signals" [38]. The cell controller should handle

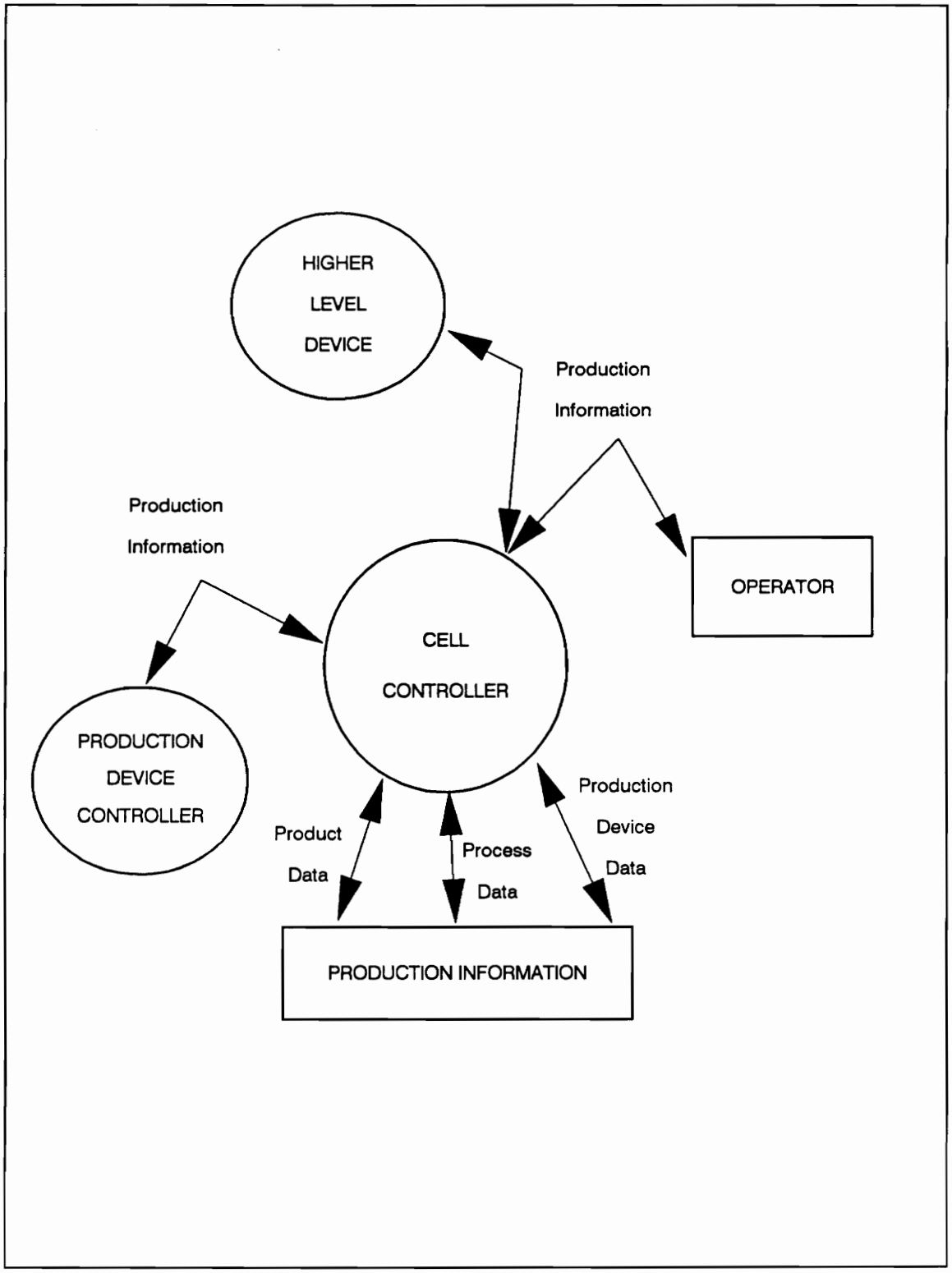


Figure 2.3 Minimum Functionality Cell Controller [14]

the communication between these devices [7, 38]. In practice, the specific application usually determines the communication structure.

Communication may be accomplished by several methods: a local area network or other network interface, serial point-to-point; or a standard communication protocol and hardware scheme. Because so much communication takes place, the method should be standardized by the user [7, 15, 33].

2.4.2 CONTROL

Control functions cover a broad range of capabilities. In addition to active control, most systems also provide passive control, or monitoring capabilities. Systems that provide only passive control are usually not considered true cell *controllers*; but a cell controller should provide this capability [14].

Active control usually involves supervision and coordination of the production devices to perform the tasks of the cell [4, 6, 38]. This also involves scheduling the tasks, [4, 28] and the start up and shut down of the devices [3]. One approach supports realtime control as a must for the cell controller [7, 38]. Another view states that because a cell controller's main function is *coordinating* machines, it does not need to perform as quickly as if it were *directly controlling* a machine. Therefore, precise realtime control is not needed; near realtime would be acceptable [11].

As previously stated, passive control is also a necessary function of the controller. The amount of monitoring is dependant on the application. Process monitoring for machines and products is typically utilized to keep track of such

behavior as quality, trends, performance, production output, and tool usage [3, 4, 6, 28, 38, 39].

2.4.3 INFORMATION MANAGEMENT

Information is exchanged between the cell controller and its surrounding environment. Information management takes place in almost all functions of cell control. Even control and coordination functions usually involve some type of information transfer. In a cell control environment, a variety of information types are handled such as programs, data, recipes, and status information [7].

The cell controller should locally store and provide management of any information necessary to perform the cell operations. This includes online program storage for the cell devices, and upload, download, and verify capabilities for these programs [3, 6, 7, 38]. The cell controller should always house the master programs for the devices. This allows comparison against the programs currently residing on the devices to verify that the correct version is loaded. If any change has occurred, the cell controller should possess the master copy to be loaded to the device [6]. Interaction between the cell and its host should be similar. The cell controller should accept any new or revised programs generating from its host [7].

Any other type of data necessary for cell operation should also be stored locally. This includes recipes, parts parameters, and production information. The same types of services to this data should be available as for programs [3, 7, 38].

Data acquisition and analysis can also be functions of the cell controller. This data type includes product and quality parameters, production totals, and process information. The information should be available for display when

necessary to the higher level interface. This also includes any status or data analysis reports and displays [4, 6, 7, 14, 28, 33, 38].

2.5 Cell Control Approaches

Two main approaches exist for developing work cell control. The first approach is a customized cell controller built to fit one particular application. The second is a controller created on a generic base that can be used for many situations. Both approaches will be discussed with more emphasis on generic control since this is the main emphasis of this research topic.

2.5.1 CUSTOMIZED CONTROL

Customized cell control has been the predominant approach in the past. Both hardware and software are customized to meet a specific manufacturing application. Usually, the software follows a general architecture that can be applied to any situation. This provides flexibility to the types of situations to which it can be applied. However, the coding is created specifically for the application. This makes development for each particular system complex [15]. Since hardware and software are designed, created, and installed for the specific situation, integration to the desired devices is done well. This reduces the chance of failure and also maximizes performance for the application. On the negative side, expansion and flexibility capabilities are poor and maintenance support is difficult since the system is unique [28].

2.5.2 GENERIC CONTROL

The second approach which is gaining popularity is generic cell control. In this approach, software is usually developed in such a way that it can be configured to fit most applications. A generic package is developed to handle a predefined set of key functions. This makes development less costly and successful results more attainable [3, 15].

Usually these generic systems are built around some type of industrial computer or work station as the cell controller. Software routines are added functionally to create the control. Since the hardware is accepted as standard, the system design, implementation, and maintenance is less of a problem. Maintenance concerns are still present, but the software is built on an accepted platform, so hopefully standard practices are used in designing the software which reduces problems [28].

Certain design aspects should be followed when developing a work cell controller to provide the flexibility required of a generic cell controller. The focus of these design aspects are now discussed in terms of communication and software development since these were the focus of the research.

2.5.2.1 Communications Approaches

Many obstacles are present which create problems for generic communications. Currently, no standard procedure is accepted and utilized for factory floor communications. Standards for communication protocols have begun to be developed but due to their limitations, they are not extensively used in factory

machines. Two of these methods will be discussed later. Proprietary protocols also pose a major problem since interfacing capabilities are limited by the vendor. Currently, most machines in the electronics industry use proprietary protocols [10].

Even PLC interfaces, which have been used extensively in industry, lack standardization. Different models of PLCs manufactured by the same vendor may utilize different serial communications protocols. Even though the protocols are usually non-proprietary, the variety makes creating their interfaces difficult. Other industrial devices have similar problems. A cell controller must support the protocol differences of the devices in its cell and do so simultaneously [2].

Although no completely generic communication system can be developed, it is possible to develop communications in such a way to facilitate generic cell control. One characteristic of generic communications is the way in which the user (operator or host computer) perceives it is happening. Communication to and control of different devices should appear to the user as occurring in an equivalent manner. The underlying protocol conversions may be different, but the user is unaware of this [11].

Messages sent between the controller and the device pass through many layers. Figure 2.4 shows eleven layers passed through during one way communication from controller to device. This does not include the control program or any response or acknowledgement that may take place [38]. A communication model shown in Figure 2.5 facilitates this message transfer. This model provides a user interface, communication system, protocol conversion, and hardware drivers [11].

A generic communications interface should not negatively affect the performance of the device or the system. The software developed for the

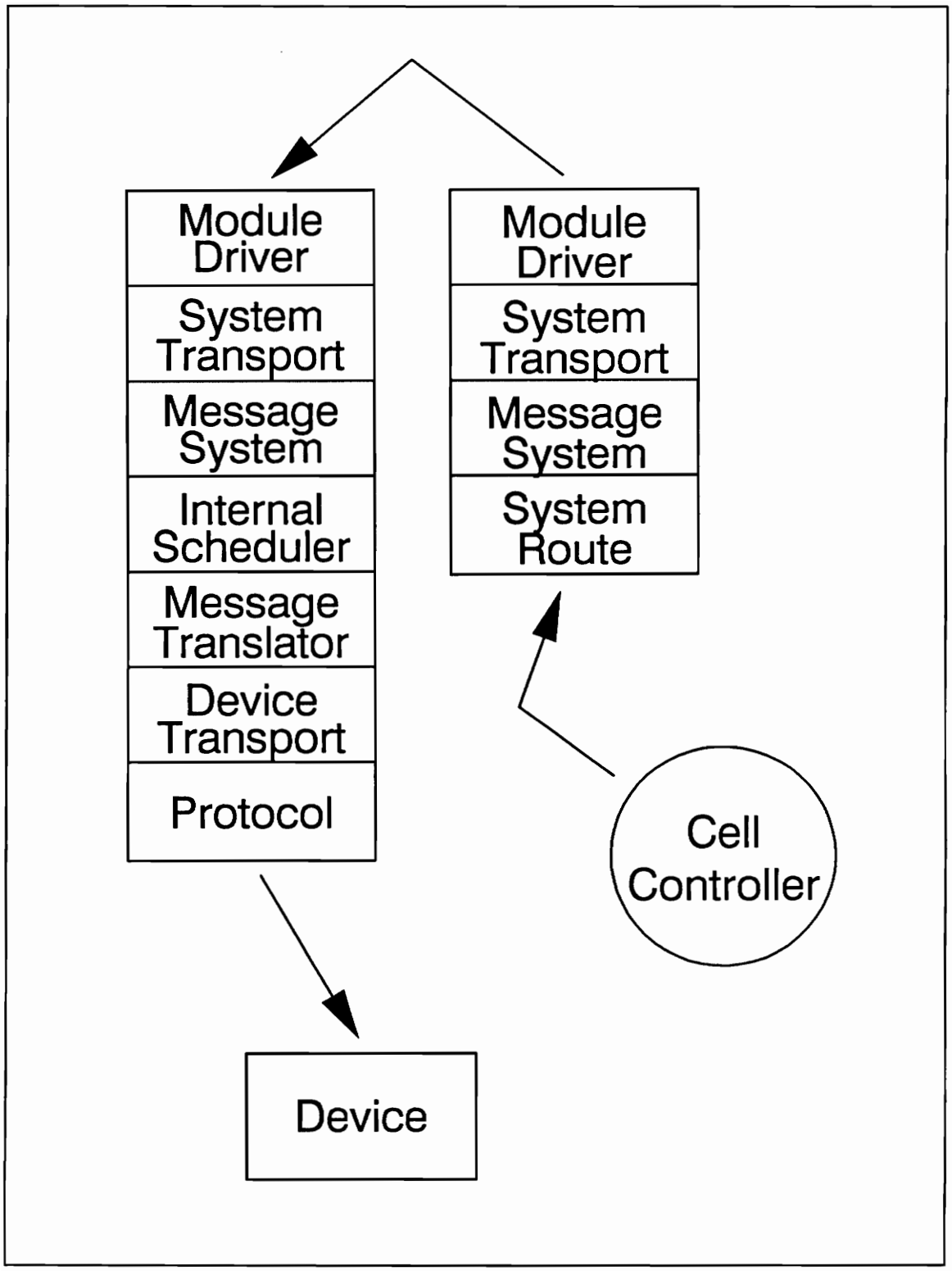


Figure 2.4 Communication Software Layers [38]

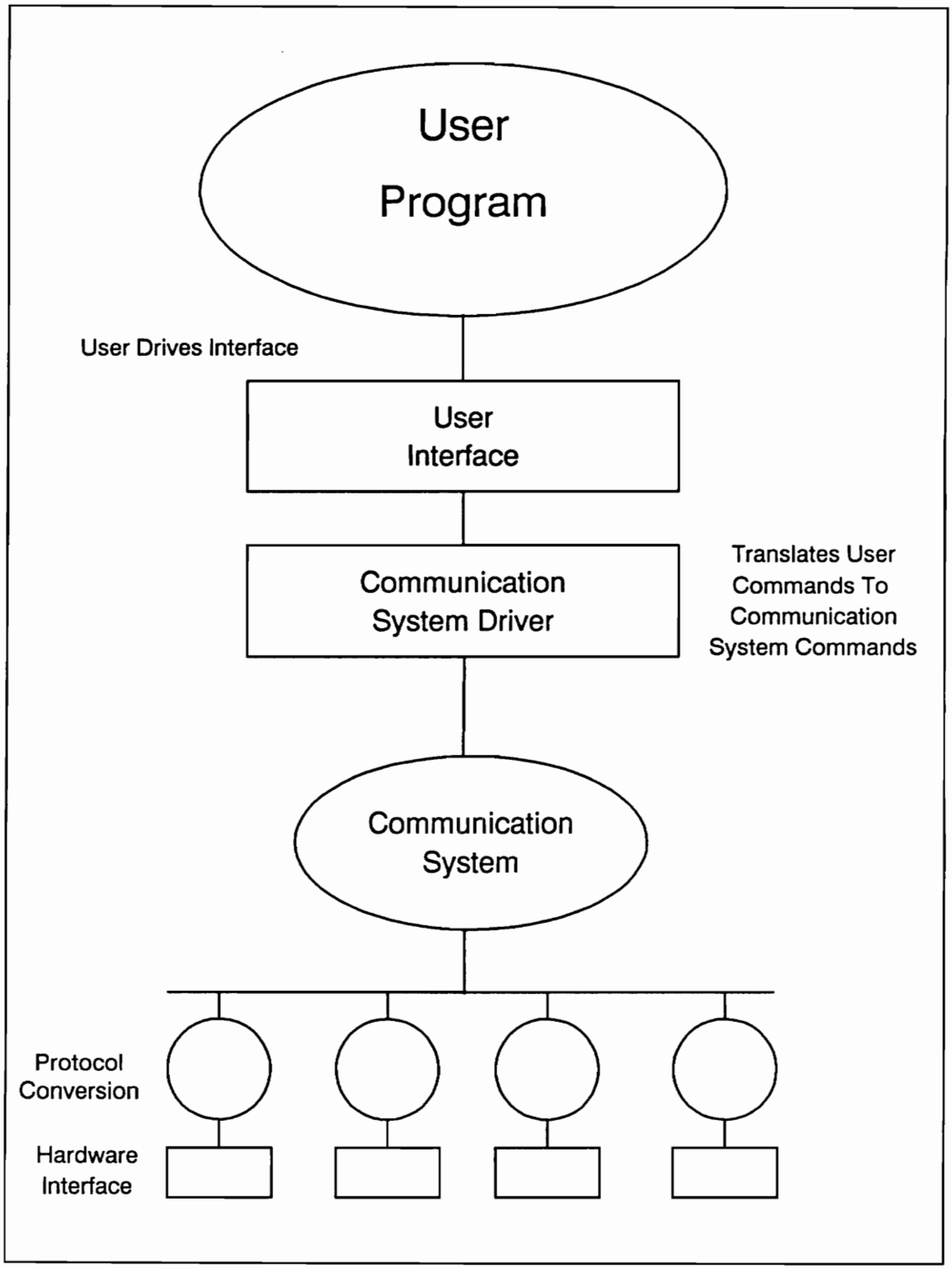


Figure 2.5 Cell Controller Communication Model[38]

communications should have realtime, multitasking, and event/semaphore handling abilities where applicable. It should also incorporate a modular architecture. Distinct low level interfaces which accommodate the required protocols and memory addresses must be present. Although this is necessary at the lower level, the user level commands should exist in a generic representation. This allows new devices to be supported by adding the required low level driver but without higher level program changes [2].

2.5.2.2 Software Aspects

The major component of the cell controller is the software driving it. Even though the specific hardware aspects are significant, a controller does not exist without an enormous amount control software [6].

Generic cell control programs require the software to be created to accommodate its functionality. In particular, the software should be flexible enough to support many different manufacturing environments and be expandable to accommodate changes and growth in the factory. This software flexibility is accomplished through planned development with various characteristics in mind [28].

Developing configurable software is one of the main ways to assure a flexible cell controller. Configurable modules and parameters should be used. Parameters include such items as device names, process routings, and alarm and production limits [10]. The modules can be assembled for specific applications and permit certain parameters to be utilized when needed. This configurations allows

parameters to be altered depending on the application without the need to alter code [3, 15, 33].

Standardization is another technique that promotes a generic cell controller. All possible elements should be standardized to make integration and implementation easier. This includes control code, functions, and information exchanges. The information and format of the items should be standardized [3]. The control structure of the software should be kept separate from the data. Therefore standardized control and data structures are necessary [29]. Other software aspects that should be standardized are documentation, interface, archival procedures, and testing [33].

The hierarchy of the control structure influences the software structure and control logic. Since cell control is usually set up in a hierarchical manner, the control modules in the hierarchy react in the same way. Inputs from a higher level module are decomposed and a response generated. Subsequently, the command is decomposed and outputs are distributed to the lower level modules [29]. The higher level modules needed are somewhat different in function and therefore different in code. But because the types of devices used are basically similar for cells, (CNCs, PLCs, robots, etc.), the lower level functionality can be similar [2].

Software can support information transfer through two methods: file-based and message-based communication. File-based transfers hold communications until a specified amount of data has accumulated. This approach is slow and used mostly for low volume transfers, low complexity, and low coordination systems [4].

Message-based communication sends only control commands or pertinent information. Messages are sent when an event occurs within the cell. These messages generate subsequent messages and events [4]. This approach supports

"communication between two independent programs running in different machines at different times with different data and command formats separated by a network or a data communication protocol" [10]. A message system is critical for communications in a multitasking environment operating in real or near realtime. The message system should make the programmer's job of moving data easier by providing the modularity necessary to distribute tasks between computers without changing code [10, 38].

Information transfer should occur regardless of where the application processes are physically located. It should not be necessary for the sender and receiver to be physically connected. This transfer can be accomplished by sending messages to and reading them from memory locations that are accessible by both the sender and receiver. This process is known as a "mailbox" technique for message transfer [29].

2.6 Available Systems and Packages

Vendors have recently begun marketing work cell control products. Some products utilize standard hardware platforms while others support only the vendor supplied platform. These products usually provide support for a few of the most popular factory devices and protocols. If communication is necessary for other devices or protocols, the vendor usually must supply them and the added cost could be significant [44].

Generic cell control applications are also present in the market. These usually contain functions for common applications such as coordination, scheduling, and resource and data management. Some tailoring is necessary with these

packages, but the cost and effort put forth are dramatically reduced. This method tries to standardize routine cell functions [44].

2.6.1 VENDOR SUPPLIED HARDWARE PLATFORMS

Honeywell, Incorporated markets the Manufacturing Automation System Controller (MAS/C) which is a complete package of hardware and software. This system includes a specialized computer, based on the 680X0 processor utilizing a VME bus. Standard control software includes an interface for the IPC 620 PLC. Optional software supports an Allen-Bradley PLC/2. The entire system supports area and cell supervision [18].

Motorola, Incorporated has the MDO-68K Cell Controller which is based on the MC680000 microprocessors and VME bus architecture. The control software utilizes a customized Motorola Computer Xplatform. It also provides common communications drivers. This package includes function chart programming and a control language for the user to configure the cell [35].

2.6.2 STANDARD HARDWARE PLATFORMS

ITP Systems, Incorporated offers MainStream. This product provides control to "any supported hardware platform." It also supports standards such as Manufacturing Message Specification (MMS) (ISO 9506), the X-Window System, and POSIX. It currently provides device drivers for three programmable controllers: Allen-Bradley PLC-2/5, Modicon 584/984, and GE Series 6. In

addition to this, it provides a tool kit to develop MMS interfaces to non-standard devices [27].

Northern Research and Engineering Corporation's Work Cell Manager runs on a VAX computer system with personal computer workstations. It offers communications with PLCs and other devices from vendors such as Allen-Bradley, Modicon, and GE. It also supports communications via DECnet [37].

FASTech Integration, Incorporated developed the CELLworks system which operates on the IBM PS/2 and 286/386 classes of computers. It utilizes the QNX (UNIX like) operating system. This package provides by far the most extensive range of device drivers, citing 19 device servers. In addition to these drivers, the package provides graphical and English-like programming tools [12].

2.7 Communication Protocol Schemes

Efforts have been made to standardize communications protocols. Two efforts which relate to the research are the Manufacturing and Automation Protocol (MAP) and IBM's Distributed Automation Edition (DAE). At this point in time, the success of each of these has been somewhat limited. A discussion of these two platforms follows.

2.7.1 MANUFACTURING AUTOMATION PROTOCOL

Manufacturing Automation Protocol was begun as an effort by General Motors in 1982. The purpose was to adhere all factory communications between

intelligent devices to this protocol standard. Since its beginnings, MAP has been through several revisions.

MAP is a combination of hardware and software communications specifications based on the Open Systems Interconnection (OSI) communications model. It is designed to support a wide range of manufacturing environments through the use of a broadband token-bus network. As mentioned, MAP has progressed through versions 2.1, 2.2, and 3.0. Many vendors have been reluctant to adhere to MAP standards since compatibility between versions is poor. For this reason, version 3.0 has been frozen for six years to try to increase support.

Interest in MAP has been somewhat low in the manufacturing environment for two main reasons: economics and vendor compliance. MAP is suitable for large expensive CPU systems. However, most potential users are reluctant to purchase MAP adapters which can cost more than the factory equipment. This view is especially prevalent in light of the revision compatibility.

The second issue affecting MAP's use is the reluctance of vendors of control equipment to provide connectivity to other devices or standard platforms. Most control equipment utilizes its own protocol format. In some cases, this protocol is proprietary. The user does not have access to the protocol specifications utilized in their control equipment [1, 40].

2.7.2 DISTRIBUTED AUTOMATION EDITION

Distributed Automation Edition (DAE) is a relatively new IBM product for factory communications. It provides a method of controlling resources (files, input/output devices, etc.) in a distributed environment. DAE would provide a

suitable environment for similar research. DAE's input output functionality only provides the following capabilities: initialize port, and send/receive messages from port. The content and verification of the messages are left for the user to develop. No device or task specific frameworks are provided for the user to model communication specific software development [20].

2.8 Literature Summary

As the literature review shows, the aspects involved in cell control are complex. Communication is one of the most important aspect of the cell controller and still creates the most problems.

Communication involves accessing all layers of the hierarchy. Downward communication to factory devices exists but it is still complicated because no standard has been accepted universally by equipment or software vendors. Some vendors choose to utilize proprietary protocols which restricts the user to interfacing with limited devices and control systems. Many standard non-proprietary protocols exist but even the interpretation of the data is unique depending on the device. The absence of an accepted standard makes creation of specific interfaces between a host and intelligent device necessary. Even with standards, the interpretation of the data is still device dependent. A method for creating functions to interpret this data is necessary.

3.0 RESEARCH METHODOLOGY

The research followed a methodology with the creation of the framework as the ultimate goal. This methodology was comprised of basically six steps:

(1) Investigate classes of factory devices.

(2) Investigate possible controller environments.

(3) Software development.

(4) Investigate previous research on devices.

(5) Retrace software development and expand concepts to other classes.

(6) Create framework for device classes and control environments.

The first step involved investigation into several classes of factory devices. The three types studied in detail were programmable logic controllers, robots, and numerical control milling machines. Specifically, the following devices were researched:

PLC:
Texas Instruments 435

Robots:
IBM 7545
Unimation PUMA 560

NC milling machines:
HURCO BMC-20
Dyna 2200/2400

Detailed documentation on the communication protocols and functionality of most of these devices was readily available. In some instances, previous research has been conducted with the device. The capabilities and restrictions of each of these devices is discussed in detail later in this document.

Several operating environments were available for the cell controller itself. The basic categories involved single tasking, multitasking, and coprocessor multitasking environments. Unique characteristics are associated with each environment. The specific environments researched include: DOS, OS/2, and IBM's Realtime Interface Coprocessor in the DOS and OS/2 environments. These are also discussed in detail later in this document.

Once the cell devices and controller environments were decided on and researched, cell control software functions were developed and coded for the TI435 PLC. The code was written prior to the development of the framework so that sufficient knowledge could be gained about the PLC itself as well as the controller operating environments. By actual coding, many intricate details were discovered which increased the level of knowledge about the device and the environments. This was necessary since previous research had not been done on this particular PLC or in several of the environments.

Background research has already been established on the other classes of devices. This includes communications and control code in some instances. Prior research projects created control code to perform specific tasks but was not designed to follow any framework. This code does provide information about communications to the devices as well as functional capabilities.

Once sufficient knowledge on the classes of devices and controller environments was obtained, the next phase was evaluating the cell control software which was developed for the PLC. The purpose of this step was not to recreate the code, but to use the knowledge obtained and determine how the code could have been developed in a more generic and structured manner. This step was the

beginning of the creation of the framework for generalized cell control communication. During this phase, the generalized concepts were brought together.

The final step was the actual creation of the framework. After writing the control software for one device in several control environments, investigating previous work on other devices in light of the objective, and looking back on the software, sufficient information was available to develop the framework. The framework is described in full detail in Chapter 7 of this document.

4.0 RESEARCH ENVIRONMENTS AND DEVICES

To gain knowledge of the classes of cell devices and controller operating devices, cell control software was designed for specific machines and operating systems. The software was developed for four controller operating system environments. Two operatin systems were considered: DOS and OS/2. Additionaly, two different communication access methods were invloved: the basic serial communication ports, and an IBM Realtime Interface Co-Processor (RIC) interface. These four environments provide varied capabilities which are inherent in a broad range of other operating systems. Cell control code was developed for the TI435 PLC in these operating system environments.

The computer on which the low level control functions were developed is an IBM 7552 industrial computer. This computer is designed specifically for factory use. The computer runs on an 80286 microprocessor and has a microchannel type bus architecture. Although the software was developed on this computer, it is portable to other IBM compatible personal or industrial computers with minimal modifications.

Details of the capabilities and restrictions of each operating environment will now be discussed. A similar discussion regarding the TI435 PLC follows next. Because the capabilities of the environments are complex and immense, only details pertinent to this research are discussed.

4.1 DOS Environment

The DOS environment is the least complex of the four operating environments. Information concerning this environment comes from knowledge gained over prolonged use as well as reference documents which were used while coding the cell control software [13, 31].

4.1.1 PROGRAMMING ENVIRONMENT

The biggest restriction of the DOS environment is that it allows only single event handling. This reduces the complexity of the coding since only one event occurs at any time. This restriction stipulates that the events be handled sequentially. One operation must complete before another can begin even if the events are unrelated.

4.1.1.1 Functions

A function is a unit of code which performs a specific operation. A function is called by the parent (or main) routine when execution of that operation is needed. Functions are created to make the code structured and easier to understand or when the operation is needed repeatedly. When a function is called, execution of the parent or calling program continues only after the function has completed its execution. Thus, the operations are performed sequentially. Variable data associated with a function is set up in the parent routine's memory space. When the function completes its execution, the memory space is released.

4.1.2 Serial Communications

Another restriction of the DOS environment deals with data transfer via the serial communications port. In any environment, information is transmitted and received through the serial port one bit at a time. The information transferred is grouped in the form of character bytes (8 bits). A transmit buffer holds the data before it is sent out of the serial port. A similar receive buffer also exists for incoming data. In the standard DOS environment, these buffers hold only one character. When receiving information, the application program must get the character from the buffer before the next character is received or the first character will be overwritten and lost. When sending information, the application program must check that the previous character has been sent before issuing the next character. Specific code must be written to perform the checks on the input and output buffers so that no information is lost. This slows the process of transferring data drastically. Because of this restriction, the baud rate for serial transmission is limited to 9600 bits per second.

4.2 OS/2 Environment

OS/2 provides a multitasking environment for the cell control software. Handling many events occurring simultaneously complicates the coding. In the cell control environment, although the events for any one machine or port must occur sequentially, several machines can be operating and accessed simultaneously. Information pertaining to this research regarding the OS/2 operating environment was obtained from several sources [8, 30, 34, 36].

4.2.1 PROGRAMMING ENVIRONMENT

In the OS/2 environment, three types of tasks are present: (1) functions, (2) processes and (3) threads. Functions in the OS/2 environment operate in the same manner as in the DOS single event environment. Processes and threads take advantage of the multitasking environment in two different ways.

4.2.1.1 Processes

Processes are events or programs running independently of each other. When referring to a high level language, a processes is a separate executable program operating in its own memory space. One process can spawn several processes but each is independent code. If one process dies, the other processes continue. Because each process operates in separate memory space, data cannot be exchanged via parameters. The processes can communicate by other means which are beyond the scope of the research.

4.2.1.2 Threads

Threads are similar to DOS functions in that they consist of a unit of code called from the parent process. The major difference being that execution of the parent process is not halted until thread execution completes. Once the thread has begun execution, the parent process continues execution at the next control statement. Both the thread and parent run simultaneously.

Threads resemble functions in that they utilize the parent's memory space. The parent must allocate stack space for the thread to access. This also allows data to be exchanged between the parent and thread via parameters. The parent has access to the parameter variables throughout the entire execution of the thread. It need not wait until the thread completes execution to determine the values of variables in the thread. Since the thread executes in the parent's memory space, if the parent dies, any threads created by the parent also die.

4.2.1.3 Time Slices

In a multitasking environment, CPU time is divided into time slices. Each program runs for a predetermined amount of time. The program has access to the CPU during this time slice. Control of the CPU is given to the events in a round robin fashion. Problems can arise if an operation is interrupted in the middle of execution. To remedy this, a critical section can be defined. The running program keeps control of the CPU until the critical section of code has completed execution.

4.2.1.4 Semaphores

A semaphore is an event flag acting as a synchronization mechanism. It is used to keep a task at certain point until a signal is given. Semaphores are usually used to keep several tasks from using one resource at the same time. Examples of resources are the screen interface, keyboard, or communications port. Semaphores are set or cleared by a process or thread using the resource. If a task wants control of a resource, the semaphore is checked. A clear semaphore signals the resource is

available for the task. A set semaphore signals the task the resource is currently in use.

Semaphores can reside in RAM or system memory. RAM semaphores use the same memory as the calling process. Therefore, the parent process is responsible for its maintenance. Only threads created by the parent process can utilize the RAM semaphore. System semaphores are allocated by the operating system. This allows separate processes to utilize the semaphore for synchronization.

4.2.1.5 Serial Communications

OS/2 provides operating system calls to accommodate serial communications. Large transmit and receive buffers are available in the OS/2 environment to prevent data from being overwritten, as it often is in the DOS environment. This removes the restriction of the maximum baud rate allowable.

4.3 Realtime Interface Co-Processor

The IBM Realtime Interface Co-Processor (RIC) card is an interface adapter used concurrently with DOS and OS/2 environments to provide multiport and multitasking abilities. The RIC card is a separate adapter card which houses up to 2 MB of its own memory and an 80186 microprocessor. By utilizing its own microprocessor and memory, the system unit (main computer) is relieved of the burden of performing some tasks. The following information on the IBM RIC card comes from a line of RIC user manuals and documentation [22, 23, 24, 25, 26].

4.3.1 OPERATING ENVIRONMENT

Several RIC operating system factors need discussion: code handling, memory, communications between the RIC card and host computer, and communications through a serial port. In this discussion, two processors and their associated memory spaces are considered. The "system unit" processor and memory refer to the CPU and RAM associated with the components normally thought of as the computer processor and memory. The "RIC side" has its own processor and memory which are separate from the components of the system unit.

4.3.1.2 Tasks

Tasks are separately compiled executable programs which run on the RIC card. They are loaded into RIC memory with a loader utility from the system unit side. Once loaded, tasks can run totally independent of the system unit. Up to 255 tasks can run simultaneously. The Realtime Control Microcode acts as the support code or operating system for the RIC card.

4.3.1.2 Memory

Each RIC card can house up to 2 MB of memory. RIC tasks execute in RIC memory, not system unit memory. This allows the system unit to perform other operations while tasks are running on the RIC side. Shared memory exists between the RIC card and the system unit for information transfer. Both the RIC task and

system unit can access the same block of memory which allows information transfer between the two.

4.3.1.3 System Unit - RIC Communications

The system unit and RIC card coordinate communication using status bytes, interrupts, and command bytes. By using these elements, a system unit and RIC task achieve the desired operations.

Each RIC task has a status byte which the system unit can check. The status byte identifies if the task is busy, idle, is currently using the shared storage, or if an error has occurred. The RIC task is responsible for setting and clearing the bits in its own status byte. This is merely a means for the system unit to know the state of the task and is not a means of passing data.

RIC tasks use interrupts to signal the system unit application when events on the RIC side occur. The RIC Realtime Control Microcode lets the system unit know which task initiated the interrupt. No information is passed with the interrupt. A common sequence is for the RIC task to perform an operation, set its status byte, write information into the shared storage for the system unit to access, and then set an interrupt flag to signal the system unit that the operation is complete.

The system unit communicates to the RIC tasks by issuing commands. The RIC task has a command byte which is set to the value of the command code from the system unit. The task would be set up to perform a specific operation based on the value of the command byte received. A typical dialogue between the system unit and the RIC task is shown in Table 4.1.

Table 4.1 System Unit - RIC Dialogue

SYSTEM UNIT	RIC TASK
(1) Set up interrupt handlers	
(2) Set up input/output buffers in shared memory.	
(3) Load task	
	(4) Initialize.
	(5) Signal SU initialization complete.
	(6) Set status byte: "loaded and initialized."
(7) Put data into output buffer.	
(8) Issue command byte to task.	
	(9) Read command byte. Branch accordingly.
	(10) Set status byte: "shared storage busy."
	(11) Read data from shared storage.
	(12) Perform operation.
	(13) Write data into shared storage.
	(14) Set status byte: "shared memory not busy."
	(15) Send interrupt to SU.
(16) Acknowledge interrupt. Read shared memory.	
(17) Continue dialogue until desired operations are completed.	
(18) Signal task to unload.	
	(19) Unload from RIC memory.

4.3.1.4 Communications

The RIC card supports serial communications via RS232 or RS422 ports. The multiport capabilities of the card provide use of up to eight ports. RIC specific calls set up the port and perform reading and writing. RIC routines are also utilized to create a large receive buffer to ensure information is not lost if the RIC task does not read the data fast enough.

4.3.2 DOS AND OS/2 SYSTEM UNIT ENVIRONMENTS

The RIC tasks are independent of the system unit operating system. The same RIC task can be used in DOS or OS/2 environments. This is true because the task is running on the RIC card under the Realtime Control Microcode and not in the system unit under DOS or OS/2. The system unit side performs the same basic functions whether operating in DOS or OS/2, although the OS/2 environment offer more features. These extra features were not necessary for the scope of this research. Therefore, the main difference focused on between the DOS and OS/2 environments, is the way interrupts from the RIC card are handled.

In the DOS environment, an interrupt flag is used to check incoming task interrupts. The flag for a specific task is checked when the system wants the status of the task. A set interrupt flag indicates the RIC task has issued an interrupt to the system unit.

The OS/2 environment utilizes semaphores to determine if a RIC task has issued an interrupt. Semaphores are created for each task and registered with the RIC operating system. The semaphore is set if the RIC task has issued an interrupt.

By using system semaphores, multiple system unit application programs can determine the task interrupt status.

4.4 Programmable Logic Controller

One of the most commonly used factory control devices is the programmable logic controller (PLC). A PLC is a highly intelligent digitally operated device which has programmable memory. The PLC controls discrete as well as analog inputs and outputs. The memory is programmed with ladder logic which is relay logic based. The ladder program controls the inputs, outputs, and accesses the memory of the PLC to perform operations necessary to control less intelligent devices [16].

A Texas Instruments 435 programmable logic controller was used in this research. This PLC operates similar to other brands of PLCs but also has its own unique features. The characteristics which affect the cell control research are discussed in detail. This information was obtained from the user's guides and internal Texas Instruments TI435 documentation [42, 43, 41].

4.4.1 COMMUNICATIONS INTERFACE

Two interface ports are available on the TI435 PLC: (1) communications control module (CCM) interface and (2) programmer port interface. The CCM interface adheres to a protocol of the same name while the programmer port utilizes the K sequence protocol. Each has a unique set of requirements for interfacing to a host computer. Both ports support standard RS232 serial communications. The programmer port is functionally dedicated to communication between the PLC and

a hand held programming unit referred to as a Machine Interface Unit (MIU). The code for this research utilizes the CCM port and follows that protocol. This is a standard communications protocol which is not limited to Texas Instruments PLCs. Other vendors such as General Electric also utilize the CCM protocol in their Series One and Three PLCs.

The CCM protocol supports a structured method of reading and writing data. The meaning of the information being transferred is specific to the PLC. For instance, the data transferred using this protocol on the GE Series PLCs would be interpreted in a different manner then the data transferred from the TI435 PLC.

The CCM protocol follows a standard block read/write format. Communications are always initiated by the host computer and directed to a specific PLC. The host dictates whether a read or write occurs. Data is always preceded and followed by specific control characters. Two types of checks are performed on the data to ensure a successful transfer has occurred: (1) parity check and (2) longitudinal redundancy check (LRC). Parity checks are inherent in the line set up. LRC is an exclusive OR performed on the data only. A calculated LRC is compared to an LRC transferred with the data.

4.4.2 MEMORY TYPES

The PLC performs many types of operations: mathematical calculations, memory storage, counting and timing operations, control of and information gathering from attached devices, and many others. Each operation utilizes a different type of memory. Three main memory types exist in the TI435 PLC: (1) ladder memory, (2) scratch pad memory, and (3) v memory.

4.4.2.1 Ladder Memory

Ladder memory stores the user ladder program for the PLC. Only one program may be resident. This memory retains its value through power on and off sequences. This memory has read/write access by the host computer.

4.4.2.2 Scratch Pad Memory

Scratch pad memory (SPM) stores information about the PLC set up. This memory supports read/write access to query or modify current settings. Each location or range of locations controls different aspects of the PLC. For example, the PLC operating mode can be determined by reading one SPM address and altered by writing to another address. Some other capabilities controlled through SPM include: (1) PLC configuration, (2) determining supported functions, and (3) reading and setting run time options and set ups.

4.4.2.3 V Memory

V memory refers to the PLC's volatile memory. The data stored in V memory is temporary, or lost when the PLC is powered off. Types of information residing in V memory include: timers, counters, accumulators, inputs, outputs, stages, and relays. The status of each of these can be read or altered.

5.0 SOFTWARE DEVELOPMENT

The developed software supports cell control for the device and environments detailed in Chapter 4: a Texas Instruments 435 programmable logic controller in the DOS, OS/2, RICDOS, and RICOS/2 operating environments. This chapter details the specifics of the code developed by discussing the approach to the software development, the code specifics for the PLC, operating system, and then discussing an overview of the code. Complete documentation of this code can be found elsewhere in the *TI435 PLC Communication Routines User's Guide*.

5.1 Development Approach

The development of the cell control software for the TI435 PLC followed several steps. A detailed knowledge of the PLC itself was first necessary before any code could be created. This was followed by learning the intricacies of each operating system and then creating the code for each. Cell control routines were written in the DOS environment first because this was the simplest environment as well as the most familiar. Coding in the OS/2 and RIC environments followed. Although the DOS environment was the simplest, it was by far the most restrictive in terms of capabilities.

5.2 TI435 PLC Functions

The functional capabilities of the TI435 PLC provided an outline of the specific cell control routines to be created. Twenty types of cell control routines

specific to the PLC were created. The code names and a description of their functionality are listed in Table 5.1. Basically, the following capabilities are provided by these routines:

(1) Upload/download ladder programs

(2) Clear ladder memory

(3) Read/write input output status

(4) Read/write timers, counters, and memory values

(5) Modify the PLC mode

(6) Check the PLC mode

(7) Read error status

(8) Provide user access/denial

The code which provides this functionality accesses several PLC memory types. The type of cell control routine stipulates the memory type accessed. Chapter 4 provides a description of the memory types.

Many functions supporting specific requirements of the PLC were developed. These include sub-functions necessary to perform the cell control routine, data conversion functions, and protocol specific functions.

The sub-functions perform operations necessary to perform the user callable cell control routine. An example is the *ladder_end* function used to determine the end of the ladder currently residing in memory. It performs a complete communications dialogue with the PLC but is only part of the clear ladder cell control routine. These sub-functions are listed in Table 5.2.

Data conversion functions were used to transform data to the necessary format to send to the PLC or to interpret the data received from the PLC. All data is sent to and received from the PLC in ASCII coded hex format. Therefore, it must

Table 5.1 TI435 PLC Cell Control Routines

download_TISOFT	download ladder in TI file format
upload_TISOFT	upload ladder in TI file format
download_user	download ladder in character file format
upload_user	upload ladder in character file format
clear_ladder	clear ladder memory
read_tmr_ctr	read timers or counters
write_tmr_ctr	write timers or counters
read_TMRA_UDC	read accum. timers or up down counters
write_TMRA_UDC	write accum. timers or up down counters
read_io_range	read input output status
write_io_range	write input output status
read_vmemory	read v memory range
write_vmemory	write v memory range
program_mode	switch to program mode
run_mode	switch to run mode
assign_password	assign password to cpu
unlock_cpu	unlock a locked cpu
lock_cpu	password lock the cpu
num_iopts	determine number of IO points attached
last_error	read last communications error

Table 5.2 TI435 PLC Cell Control Sub-functions

run	put PLC in run mode
program	put PLC in program mode
test_mode	put PLC in test mode
check_mode	determine current PLC mode
ladder_end	find end of user ladder
read_prog_name	read ladder program name
read_watchdog	read watchdog timeout value
read_ret_range	read retentive range values
write_prog_name	write ladder program name
write_watchdog	write watchdog timeout value
write_ret_range	write retentive range values
check_cpu	check if CPU is locked
test_iopt	test each bit of IO byte for status
calc_io_bytes	calculate total number of bytes to read
read_io	read IO bytes for IO range
set_iopt	set bit in IO byte to set IO status
write_io	write IO bytes to set IO status

be converted before it as sent to, as well as when it is received from the PLC. As mentioned, data is stored in different formats depending on the PLC memory type. Timers and counters are stored in memory as binary coded decimal. This is converted to ASCII coded hex format if read from the PLC. In a sense, the timer or counter values are read as ASCII coded binary coded decimal. A conversion function transforms the data from this format to a decimal (base 10) format for the user. All data requested from the PLC is passed to the user in base 10 format. All data sent to the PLC is in ASCII coded hex format. A listing of the conversion functions is found in Table 5.3.

As stated earlier, communications control module is the protocol utilized by the TI435 PLC. The software supports this protocol which utilizes an enquiry, header, data block. The enquiry begins communications to a particular PLC. The header block specifies the memory type, address, data length, and read/write option. The information is then transferred in ASCII coded hex format in a data block. The data received is checked for errors by calculating the LRC. Table 5.4 lists the functions created to perform the above sequence of protocol operations.

5.3 Operating System Code

Specific code was necessary to perform operating system tasks involved in cell control. Chapter 4 described the relevant capabilities of each operating system. While each environment allowed flexibility in the code, they also posed certain restrictions. These were used as each set of code was developed in an attempt to create somewhat generic code. The software developed for the TI435 PLC is not

Table 5.3 TI435 PLC Cell Control Data Conversion Functions

ascii_to_hex	1 ASCII coded hex byte to hex
dbl_ascii_to_hex	2 ASCII coded hex bytes to hex
hex_to_ascii	1 hex byte to ASCII coded hex
dbl_hex_to_ascii	2 hex byte to ASCII coded hex
aBCD_to_hex	4 digit ASCII coded BCD to hex
dbl_aBCD_hex	8 digit ASCII coded BCD to hex
long_hex_to_aBCD	hex to 8 digit ASCII coded BCD
dec_to_aBCD	hex to 4 digit ASCII coded BCD
aoct_to_hex	3 digit octal to hex

Table 5.4 TI435 PLC Cell Control Protocol Functions

enquiry	enquiry sequence
send_header	send header and get ACK
send_one_block	send one data block
send_whole_block	enquiry, header, and send one data block
read_one_block	read one data block
read_whole_block	enquiry, header, and read one data block
calc_lrc	calculate ASCII coded hex LRC
comp_lrc	compare calculated LRC with PLC LRC

the ideal generic code since its development acted as a learning device in this research. But it is a step towards this goal.

Cell control routines were created in what was determined as the most natural format for that operating system. In the multitasking environments, a method to check the status of the routine was also necessary. A set of operating system specific code handles communications through the serial port. This set was recreated for each environment. Table 5.5 lists the seven types of functions used to perform these operations. The code for each operating system is unique although the names remain the same.

5.3.1 COMM DOS CODE

Cell control routines were created as functions in the DOS environment. This interface is natural in the DOS environment and utilizes its specific capabilities. Parameters for each function are passed via data structures. This allows information pertaining to the function to be passed as a group. The specific DOS port communications routines follow those outlined in Table 5.5.

5.3.2 COMM OS/2 CODE

Threads were used for the cell control routines in the OS/2 environment for several reasons. Because threads are created in the parent's memory space, data exchange is simple; much easier than data exchange between processes. Again, structures are used to pass parameters for the routines for the same reason as

Table 5.5 TI435 PLC Cell Control Communications Functions

initialize	initialize and set up port
shutdown	close port
send_one	send one character out of port
send_array	send character array out of port
read_one	read one character from port
read_array	read character array from port
clear_rcv_buffer	clear characters in receive buffer
num_received	check number chars in receive buffer

mentioned above. All needed information is passed to the thread via the structure name.

Because the parent does not wait for the thread to complete execution before continuing with its own, a routine was developed to check the status of the thread. A semaphore signals the completion of the cell control thread. A clear semaphore signals the parent, or other process or thread, that the cell control thread has completed its execution and is no longer using the port resource. System semaphores allow other processes, which do not execute in the parent's memory space, to access the same semaphore as the cell control routines.

The parent has access to the values of the parameters at any time during the cell control thread execution. Currently, for read routines, the threads perform a one time read, and exit. They could be easily modified to perform continuous monitoring. Because the parent has access to the data continuously, the thread need not exit to pass the values to the parent.

Due to the nature of multitasking and time slice intervals in OS/2, critical sections were needed to prevent the PLC from sending a time out error. The critical section occurs during the enquiry and header sequences of the protocol. By noting this section as critical in the code, all operations (communications) in this section occur before control of the CPU is passed to another task.

5.3.3 RIC CODE

The RIC environment has three types of code which were created: (1) RIC side, (2) DOS system unit, and (3) OS/2 system unit. The RIC side performs the actual cell control routines, while the system unit side performs the supervisory

functions. The two sides interact to accomplish the complete cell control routine functions.

The cell control routines were created as tasks which are loaded by the system unit (DOS or OS/2). Twenty separately compiled programs make up the cell control routine tasks. These tasks can run simultaneously with the system unit control program. Data is again exchanged in the form of data structures which are located in shared memory for both the system unit and task to access. The system unit loads the task, and places the structure in shared memory. A start command is issued to the task which then performs the cell control functions, utilizing the data in shared memory. The task writes any necessary data in the same shared memory for the system unit to access. After its completion, the task interrupts the system unit.

The system unit can perform other operations while the task is executing on the RIC card. A function was created to check the task status. If the task has completed, the system unit reads any return parameters in shared storage. At this point, it issues an unload command to the task to free the RIC task memory space.

The system unit commands are similar in the both DOS and OS/2 environments. The main difference being in the way that interrupts are handled. Chapter four described the methods for both DOS and OS/2. The cell control software created provides functions in each operating system to start the cell control routines, and check their status. Inherent in these are functions to issue commands, read and write shared memory, and check the interrupts or semaphores.

5.4 Commonality

Commonality between environments is a desired trait of the cell control software from both the user's and the code developer's perspective. The software structure supports this in two main ways: function call and parameter structure. Not only will this make the user's task of utilizing the software easier, it made its development simpler. As previously explained, the cell control routines were set up in different ways depending on the operating environment: functions, threads, and tasks. To keep the user interface common, similar functions in each environment were created with the same name. For example, the read timer routine is the function *read_tmr_ctr* in every environment. The cell control operation it performs is the same: read a range of timers or counters; but the operating system interface differs. In the DOS environment, the function itself performs the cell control operation. In OS/2, the function starts a thread which performs the operation. Once the thread has started, the function returns control to the parent. A similar method is used in the RIC environment but with tasks. Once the task has loaded and begun, control is returned back to the system unit control code. The common function interface allows the cell control routine to be called in each environment in the same way with the user being unaware of any underlying differences.

Not only were the main cell control routines kept common, underlying support functions were created in the same manner. For example, port communications routines perform the same operations but have drastically different code specific to the operating system. By keeping the function call format the same, when the cell control code was ported to another environment, these functions were modified but not the functions calling them.

Parameters were also created in a format which allows them to be passed in a similar manner regardless of operating system. A predefined structure exists for each cell control routine which the user must initialize. A common set of variables exists for each routine which crosses environments. Some additional parameters are environment specific. Common variable types include those which are necessary to support the cell control routine regardless of environment: data type, range, PLC target, host ID, and error status. Environment specific parameters define the port or other additional information. In the more complex RIC environment the following information is also needed: RIC card ID, port ID, and user or device ID.

Because a common user interface was desired, the capabilities of each environment may not have been utilized to their full extent. In the OS/2 environment, the cell control routines could have been created as separate executable processes. This differs drastically from the DOS environment and the concept of a hierarchical control structure. Many of the RIC capabilities were not utilized to preserve commonality. The system unit simply loads the task and then checks for its completion.

5.5 Code Overview

The developed software provides a set of cell control routines for the TI435 PLC in the DOS, OS/2, RIC-DOS, and RIC-OS/2 environments. In each environment, twenty cell control routines are available. These were previously listed in Table 5.1.

The C language was chosen as the high level language for this research for several reasons. It provides the necessary features for developing modular and

configurable code. The most important being the use of data structures. Structures group data under one name making reference of several parameters for a control routine simple. The only information which needs to be passed to the control routines is the memory address of the specific structure. Microsoft C version 5.1 was used specifically because it is supported by all the researched operating environments and is supported by many other standard operating systems.

Specific communications functions were developed for each operating system. To preserve a common format, the functions perform the same basic operations and are named the same regardless of the environment.

Parameters changed depending on the environment, but by utilizing structures to pass them, the higher level code did not have to be modified to accommodate this. Only the structure format and the specific functions needed to be changed.

A separate document contains the complete source listing of the software developed. This listing contains all the total functions created to support cell control of the TI435 PLC in the four environments discussed. *The TI435 PLC Communication Routines User's Guide* gives detailed descriptions of each routine, function, thread, and task. Information regarding specific requirements such as operating system, language versions, and compiling instructions are also defined.

6.0 OTHER FACTORY DEVICES

The three major classes of work cell devices researched include: programmable logic controllers, robots, and numerical control machines. Each accommodates communication with its own specific protocol. Chapter 4 provided a detailed discussion of the PLC class. The other classes have been researched and the pertinent characteristics are discussed in this chapter.

6.1 Robots

Robots are highly intelligent devices used in the manufacturing environment. Usually, the robot has its own controller which must be interfaced to the host computer in a cell control environment. In the robot class of cell devices, two types of robots have been researched: the IBM 7545 robot and the Unimation Puma 560 robot.

6.1.2 IBM 7545 ROBOT

The IBM 7545 robot controller uses the AML/E language for robot programs. Communications to a host computer is supports through RS232 or RS422 serial links. A transaction based protocol is used in which the host computer initiates the actions. Transactions consist of an identifier, a record, and options. The identifier indicates what type of action will occur. The record contains the identifier, data, control characters, and error checks. The records from the host stipulate the type of action which will occur between the host (cell control)

computer and the robot controller. Six basic types of actions are supported by this protocol:

(1) read
(2) execute
(3) new program
(4) control
(5) teach
(6) present configuration

6.1.2.1 Read

Read actions are performed when the cell controller requests status or variable information. Within this class of identifier, nine types of reads can be performed:

(1) machine status
(2) reject (error) status
(3) robot type and microcode version
(4) robot parameter table
(5) current instruction address
(6) digital input and output
(7) all variable data
(8) current position
(9) particular variable value

6.1.2.2 Execute

Execute actions allow the cell controller to access functions supported by the operator robot control panel. The cell controller has access to nine types of operator control functions:

(1) send manipulator to home position
(2) recall memory
(3) reset error
(4) select auto mode
(5) start robot run cycle
(6) stop robot run cycle
(7) stop and remember stop point
(8) step through robot program
(9) select program to run

6.1.2.3 New Program

The new program action allows the cell controller to download a program into robot memory. The protocol does not support uploading a program from the robot controller memory to cell controller memory.

6.1.2.4 Control

Control actions provide the cell controller with the ability to control the execution of a robot program currently running. Six types of control operations are available to the cell controller:

(1) suspend execution
(2) restart execution
(3) execute until say stop
(4) set stop address for debugging
(5) reset robot controller
(9) modify variables

6.1.2.5 Teach

Teach actions allow the cell controller to change the physical location of the robot manipulator or alter other factors affecting the robot. Three types of teach actions are allowed:

(1) motion parameters (payload, velocity, and zone accuracy)
(2) manipulator movement to specific location
(3) altering digital outputs

6.1.2.6 Robot Initiated Communications

Another type of communications is robot controller initiated. The standard protocol only supports host initiated communications. Robot initiated communications support transmitting and receiving information from the host computer. This is done by including command statements in the robot AML/E program. The host computer must be in the XON state to let the robot controller know it can receive information. When the specified command statement is reached in the robot program, communications is initiated by the robot controller. This capability is useful in gathering information such as the number of completed parts, any faulty conditions, or cycle end [21].

6.1.2 UNIMATION PUMA 560 ROBOT

The Unimation PUMA 560 robot is controlled by a Digital PDP-11 controller. This controller has its own terminal interface to the controller. This supervisor utilizes the Digital Data Communications Message Protocol (DDCMP). It also supports Unimation's VALII programming language for the robot.

To interface from a personal computer to the robot controller, the Unimation software package PCVAL is available. The software is menu driven and emulates the PDP-11 supervisory computer so commands can be entered directly from the personal computer keyboard instead of the robot controller terminal.

A separate program exists which sets up all messages from the PC in the correct protocol format to interface to the PDP-11 supervisor. This program is a terminate-stay-resident program which resides alongside the DOS operating system. It is loaded prior to running the PCVAL software.

PCVAL also supports user written application programs. This allows the user to create applications interface software written in a high level language to perform control functions. The user written application program must interface to the PDP-11 supervisor via the terminate-stay-resident program.

The user written control programs must follow a specified format and provide correct response sequences to the robot controller. The format is specified as structures and commands which provide the control functions. To send messages to the robot supervisor controller, structures are set up with a command code, status code, message buffer, and buffer length. Eight commands are supported using this application interface:

(1) read data
(2) write data
(3) open port
(4) send network start
(5) input status
(6) output status
(7) close port
(8) file input/output

These functions are similar to those discussed for the IBM 7545 robot.

6.2 NC Machines

Two numerical control devices were researched for the third class of cell devices: the HURCO BMC20 and the Dyna 2200/2400 milling machines. This class of devices supports only a limited amount of communications to a host computer.

6.2.1 DYNA 2200/2400 MILLING MACHINE

Communications between the host computer and the Dyna occur in a non-blocked format. No control characters signal the start or end of a transmission block. Only two types of control actions are supported by the communications: upload and download.

6.2.1.1 Upload

Uploading a program from Dyna memory is a simple process. The Dyna controller is manually put in an upload state. It then waits for the hardware lines to be initialized by the host computer. At this point, it dumps the program character by character through the serial port. No character communications are initiated by the host computer. The host computer can request a pause by lowering the hardware line control status and then raising it to restart communications.

6.2.1.2 Download

Downloading a program from a host computer requires a series of steps. First, the Dyna controller must be manually put in a download state. At this point, it is ready to receive the program. The host computer sends the program character by character without any control characters. No error checks such as parity or longitudinal checks are performed on the data. If the data received is not syntactically correct, the line is thrown out. No signal of this occurrence is sent to the host computer. When an END statement is received, the Dyna cuts off communications and no further information is accepted [9].

6.2.2 HURCO BMC-20 MILLING MACHINE

The Hurco milling machine also supports upload and download capabilities through the serial port but through a proprietary protocol. The vendor provides executable support software to perform the two operations at an additional cost. The specifics on the protocol are not available to the public. Tests done on communications show that the protocol is more sophisticated than the Dyna in that it does provide error checking. The proprietary nature of the communications protocol makes integration of the Hurco milling machine into a cell control environment difficult [19].

7.0 FRAMEWORK

The framework for the development of low level cell control functions consists of three areas: (1) control function categories, (2) code framework, and (3) data structure framework. Each category will be outlined and detailed for a user to follow in developing cell control code. These three areas are not mutually exclusive. Integrated together they form the whole cell control code structure. To develop the code and data format, aspects pertaining to the device and the operating environment were first explored. After explanation of the framework, an example is given which applies the framework concept to the PLC. The benefits achieved from following the framework are also discussed.

7.1 Cell Control Function Categories

The major control functions outlined in the literature and the functionality of the researched devices provided the background necessary to categorize cell control functions. Five control categories and one setup function were defined:

(0) initialize/shutdown port

(1) control

(2) upload

(3) download

(4) read

(5) write

The exact capabilities within each control category are dependent on the factory device. Also, some devices may not support every category, but their functions should fall into this grouping.

(0) **Initialize/shutdown** functions set up the cell control communications port which will interface with the factory device. Line control is set and any other environment specific necessities. Line control is set to support the attached device. The initialize/shutdown functions do not perform any active control or communications with the device, but are required for cell control operations.

(1) **Control** functions provide active control of the device. Included in this category are functions such as starting and stopping execution, changing the device mode, clearing memory, etc.

(2) **Upload** functions read the programs or recipes currently residing in device memory and store them in the cell controller's memory.

(3) **Download** functions load a new program or recipe from the cell controller's memory to device memory.

(4) **Read** functions gather information such as data values and status information from the device.

(5) **Write** functions send information such as data values and set up information to the device.

Each control function has parameters which are specific depending on the device and its capabilities. For example, in the read category, the data type would be a parameter associated with the specific read function. Other parameters include amount of data, location of data, and a data array to place the values read.

In addition to the parameters, the control code and data interpretation is also device specific. The data read from a PLC is interpreted differently than that read

From a robot. The control code to perform the functions is dependent on the device. Even though it seems as though all the code and parameters are specific, they can be structured in a general format.

7.2 Device vs. Environment

The cell control functions have two main variable factors: the operating environment and the factory device. The information (data and code) used in the cell control routines can usually be categorized as being device or operating system dependent. In these two categories, the information can further be broken down into generic or specific. Figure 7.1 categorizes environment and device dependent aspects of cell control communications.

7.2.1 ENVIRONMENT

The environment is the operating system and special peripherals utilized on the cell control computer to perform its control functions. The following aspects were determined to be environment dependent:

(1) port communications

(2) event management

(3) resource management

(1) **Port communications** refer to the way information is passed through the port. This encompasses port initialization and shutdown, as well read/write capabilities. The port could be serial, parallel, or utilize a special network, depending on the controller set up.

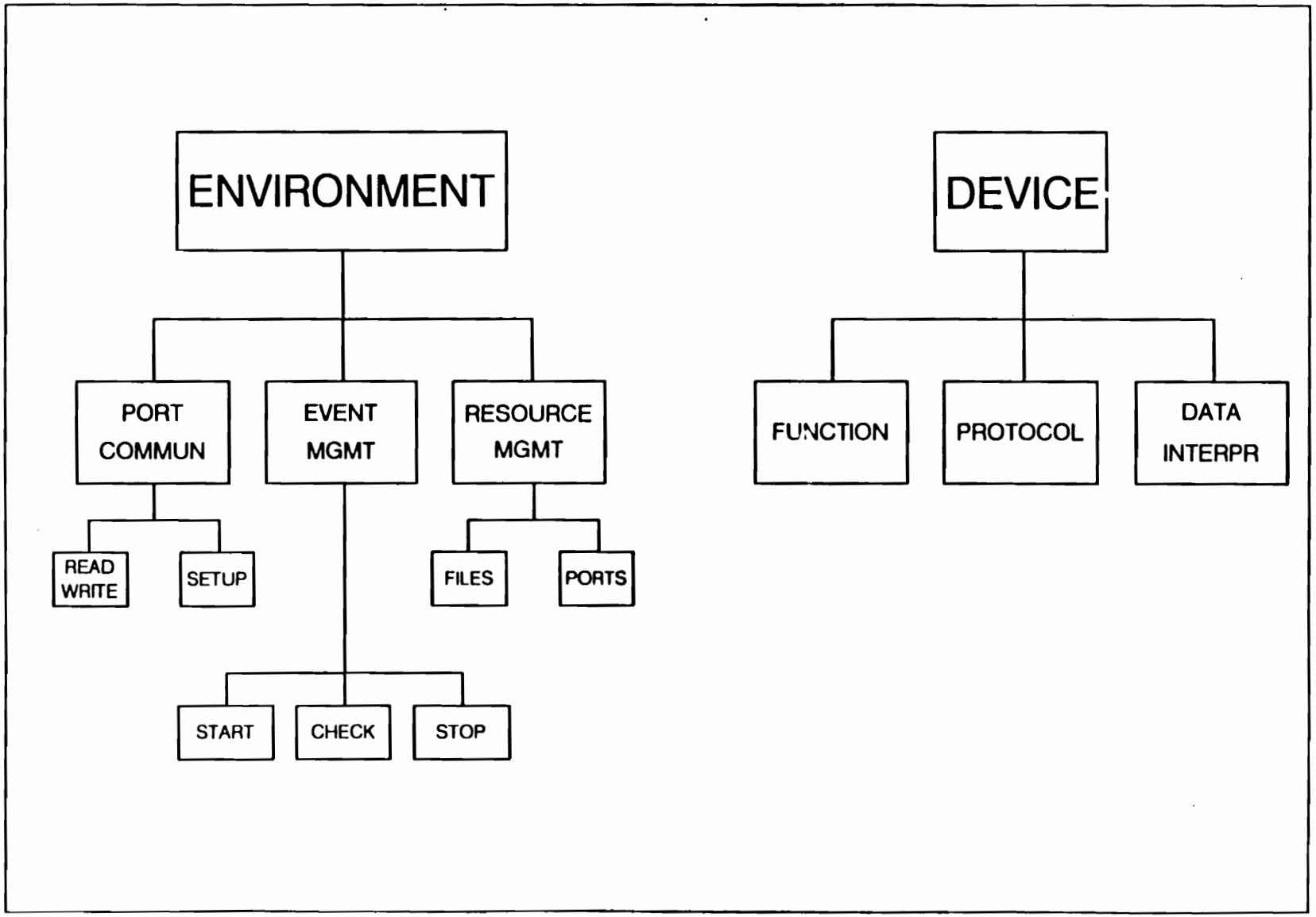


Figure 7.1 Environment vs. Device Dependent Aspects

(2) **Event management** refers to the manner in which the cell control functions are handled. This includes starting the cell control function, checking its status, and ending the operations when necessary. For example, in the OS/2 environment, cell control functions are set up as threads. Specific steps must be followed to set up and start threads in the OS/2 environment. Event management also handles how the status of the cell control routine is determined.

(3) **Resource management** refers to how resources such as files, ports, screens, etc. are allocated. Only one control function should have access to a port or a file at any time. The manner in which access is granted or denied is specific to the operating system. This is usually handled by some type of semaphore.

7.2.2 DEVICE

The device refers to the unique factory machine with which communications is taking place. Four aspects were considered to be device dependent:

(1) available functions

(2) scope of functions

(3) protocol

(4) line communications setup

(1) **Available functions** refer to the cell control categories defined above. Not all devices support the five classes of cell control functions.

(2) The **scope of functions** refers to the functionality available within a cell control function category. For example, the read function class can consist of numerous data types depending on the device or may be limited to only one type.

(3) The **protocol** is the supported communications format of the device. Protocols have been previously discussed in Chapter 4.

(4) **Line communications set up** refers to the line control parameters that are supported by the device. These include baud rate, parity, stop bits, and data bits.

7.3 Data Framework

Once the aspects pertaining to the environment and devices were determined, the data structure framework was developed. When looking at the aspects dependent on the device and environment, certain data is needed regardless of the type of device or the operating system. Other data is specific to these variables. The format for the data was set up to start with an overall generic format and migrate to specifics where necessary.

The C language supports this concept through the use of data structures. Structure types are defined which hold specific information. The structures are referred to either by name or a pointer to the structure. A pointer to the structure stores the address in memory where the structure resides as opposed to its actual values. This aspect of C is very useful when setting up the specific cell control data.

The data for the cell control functions is stored in structures. At the highest level, a generic structure type is used. Figure 7.2 shows the format of a typical high level structure used in the framework. The structure holds pointer to four other structures. Regardless of the device or environment, the same structure type is used. As more detailed information is needed, pointers provide access to the

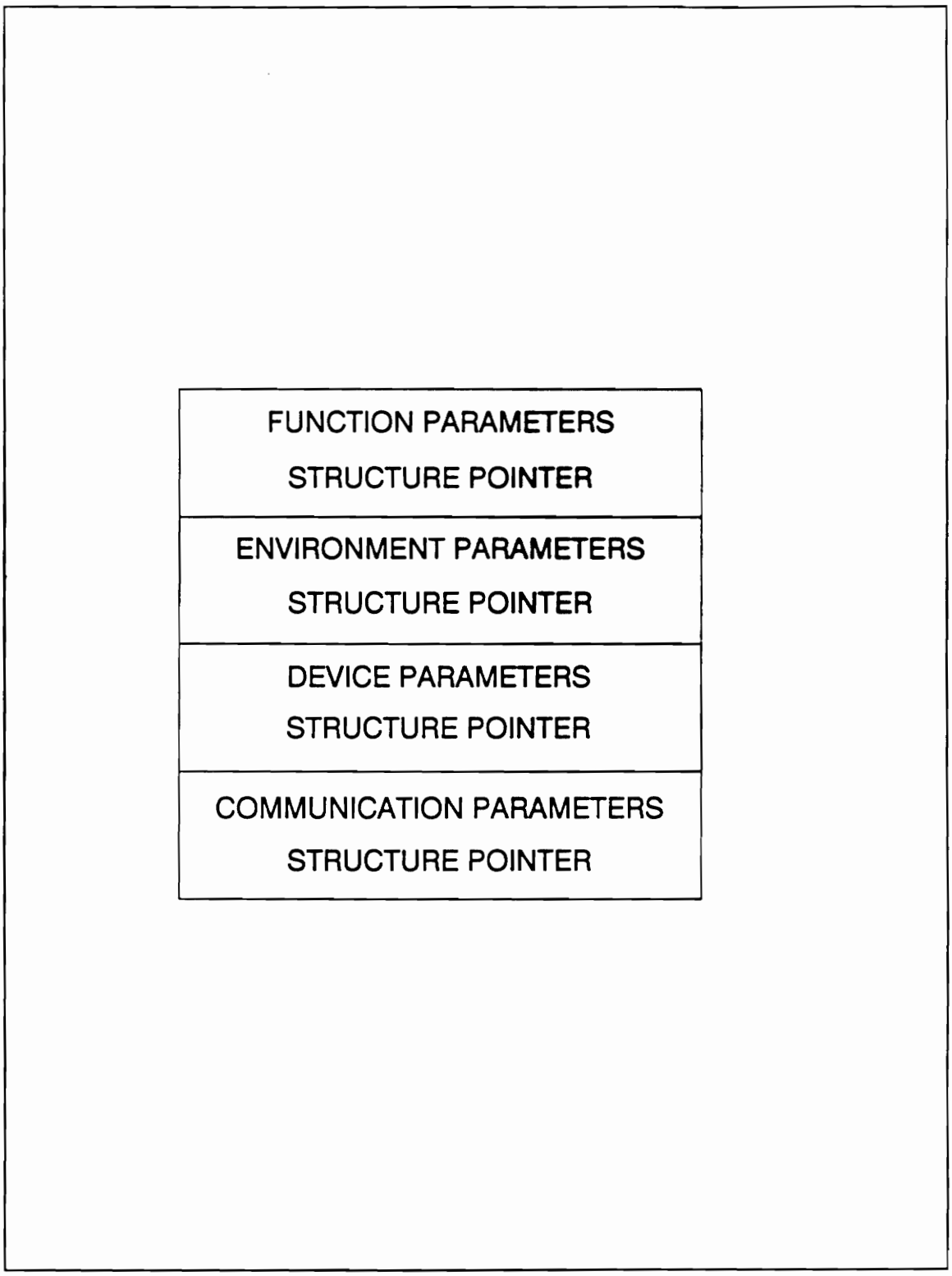


Figure 7.2 High Level Generic Structure

specific structures. This lets the higher level structure remain generic but access specific information.

Four generic data structures make up the main generic data structure: (1) function parameters structure, (2) environment parameters structure, (3) device parameters structure, and (4) communications parameters structure. These structures may utilize lower level generic structures. Each has an identifier and a pointer to another more specific structure. The code developer can easily add lower level structures to support any specific information necessary.

7.3.1 FUNCTION PARAMETERS STRUCTURE

The generic function structure holds two types of information: (1) command identifier, and (2) command parameters. The primary purpose of this structure is to provide the information for the device cell control functions. The format for the function structure is shown in Figure 7.3.

(1) The **command identifier** indicates the type of command to be carried out within the cell control category. Using the read function example, the command identifier specifies what type of data will be read. The command identifier specifies the particular function to be performed within the scope of the immediate category.

(2) The **command parameters** are the parameters necessary to execute the function specified by the command identifier. The function structure holds the pointer to the command parameters structure. Information such as data type, data length, and range values, is stored in the command parameter structure.

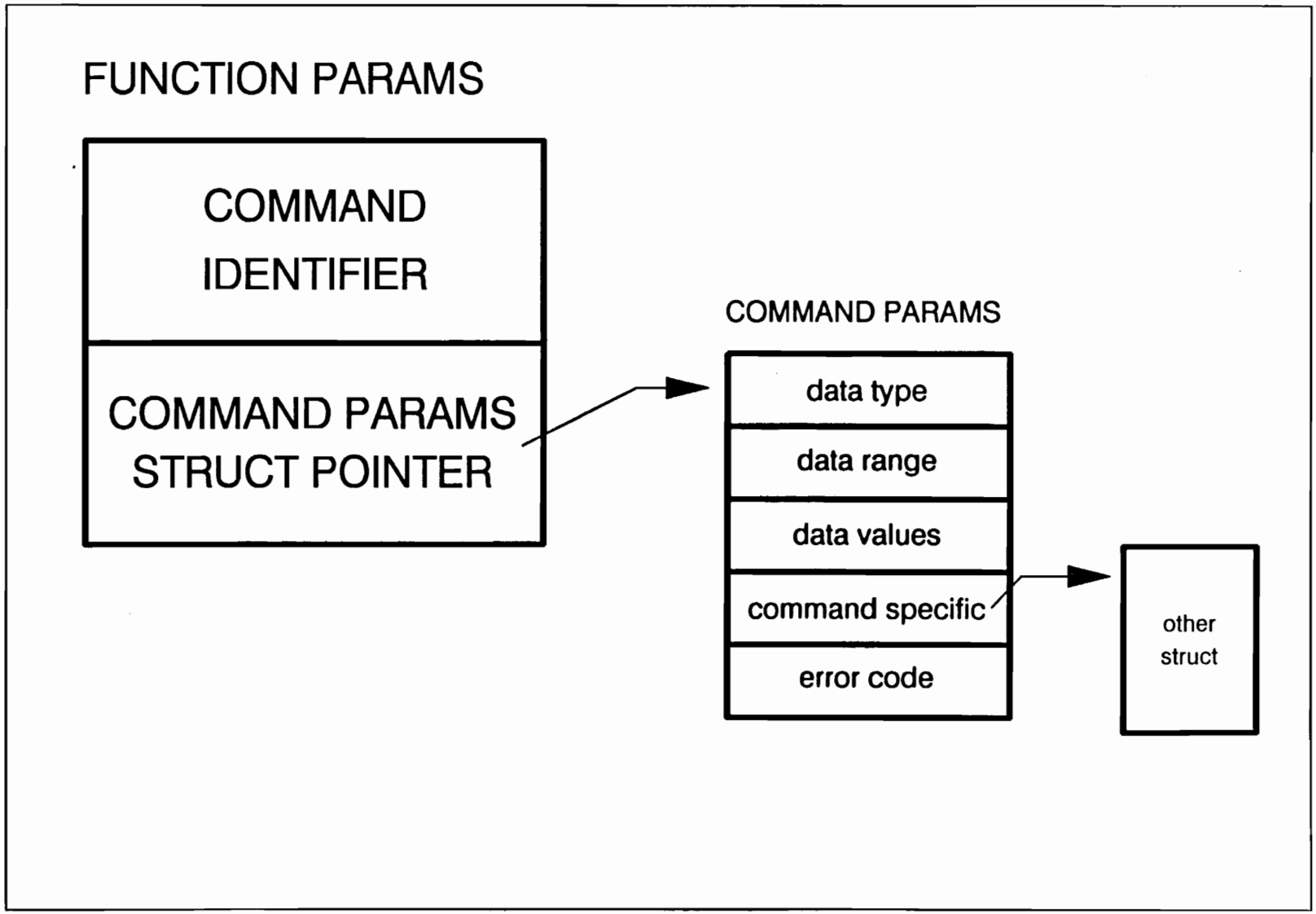


Figure 7.3 Function Parameters Structure

7.3.2 ENVIRONMENT PARAMETERS STRUCTURE

The environment structure holds two types of information: (1) operating system identifier, and (2) operating system parameters. If code is developed for multiple operating systems, this information can be used to identify the cell controller's current environment. This structure can be used to dynamically link the correct operating system specific code. If this ability is not necessary, the structure can be defined as NULL. Figure 7.4 displays the environment parameters structure.

(1) The **operating system** identifier acts similar to the command identifier. It identifies the current operating environment. Based on this identifier, specific code can be accessed or dynamically linked.

(2) The **operating system parameters structure** holds information about the selected operating system and its environment. The environment data structure holds the pointer to this structure.

7.3.3 DEVICE PARAMETERS STRUCTURE

The device structure holds three types of information: (1) device name, (2) device type information, and (3) line setup information. Figure 7.5 shows the device structure and its set up.

(1) The **device name** identifies the device within the cell. The device can be referred to by this name. For example, if two robots are attached in the cell, the respective structures could be name ROBOT1 and ROBOT2 to identify each one.

(2) The **device type structure** hold information about the device identified by the device name. This structure is specific for the device but follows a general

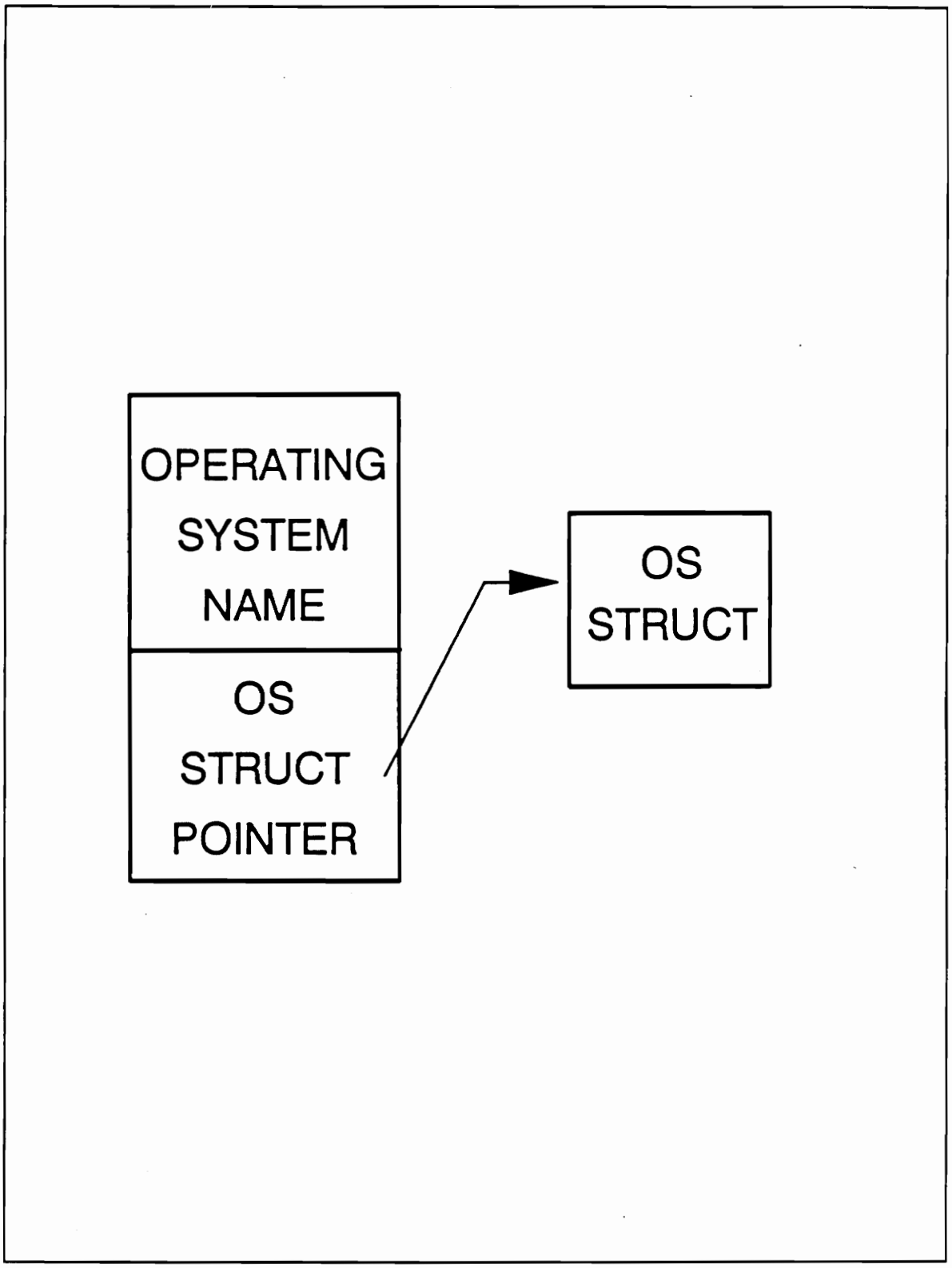


Figure 7.4 Environment Parameters Structure

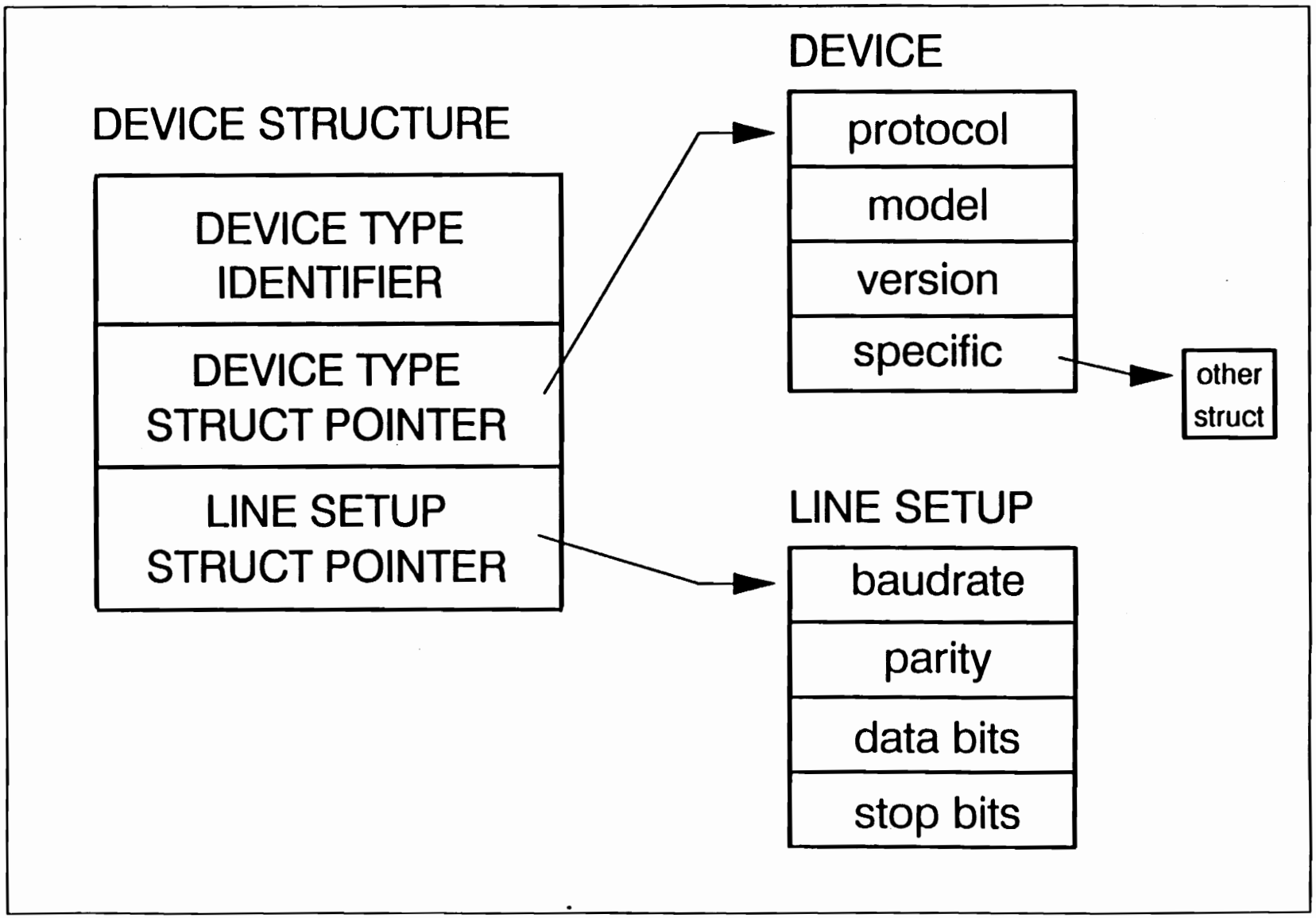


Figure 7.5 Device Parameters Structure

format within factory device classes. Typical information data available in this structure includes a protocol identifier, model, and version number. Other data can be set up and used as needed.

(3) The **line setup structure** stores the parameters used to initialize the port. This includes the baud rate, parity setting, data bits, and stop bits. This structure is included as part of the device structure because its settings are device dependent.

7.3.4 COMMUNICATIONS PARAMETERS STRUCTURE

The communications structure holds two types of information: (1) port identifier, and (2) a port identifier structure. This information, in addition to the line set up parameters, is needed to initialize the port. Figure 7.6 displays the communications structure and its supporting data.

(1) The **port identifier** identifies the port attached to a specific device. When referring to the port in communications functions, this value is used as opposed to any other port addresses which are operating system specific. In some environments, a handle is returned when the port is first initialized. This handle references the desired port in subsequent port functions. This identifier is not operating system specific but rather a user name given to reference the port.

(2) The **port identifier structure** provides the operating system specific information to identify the port. This may contain a port address, card number, etc.

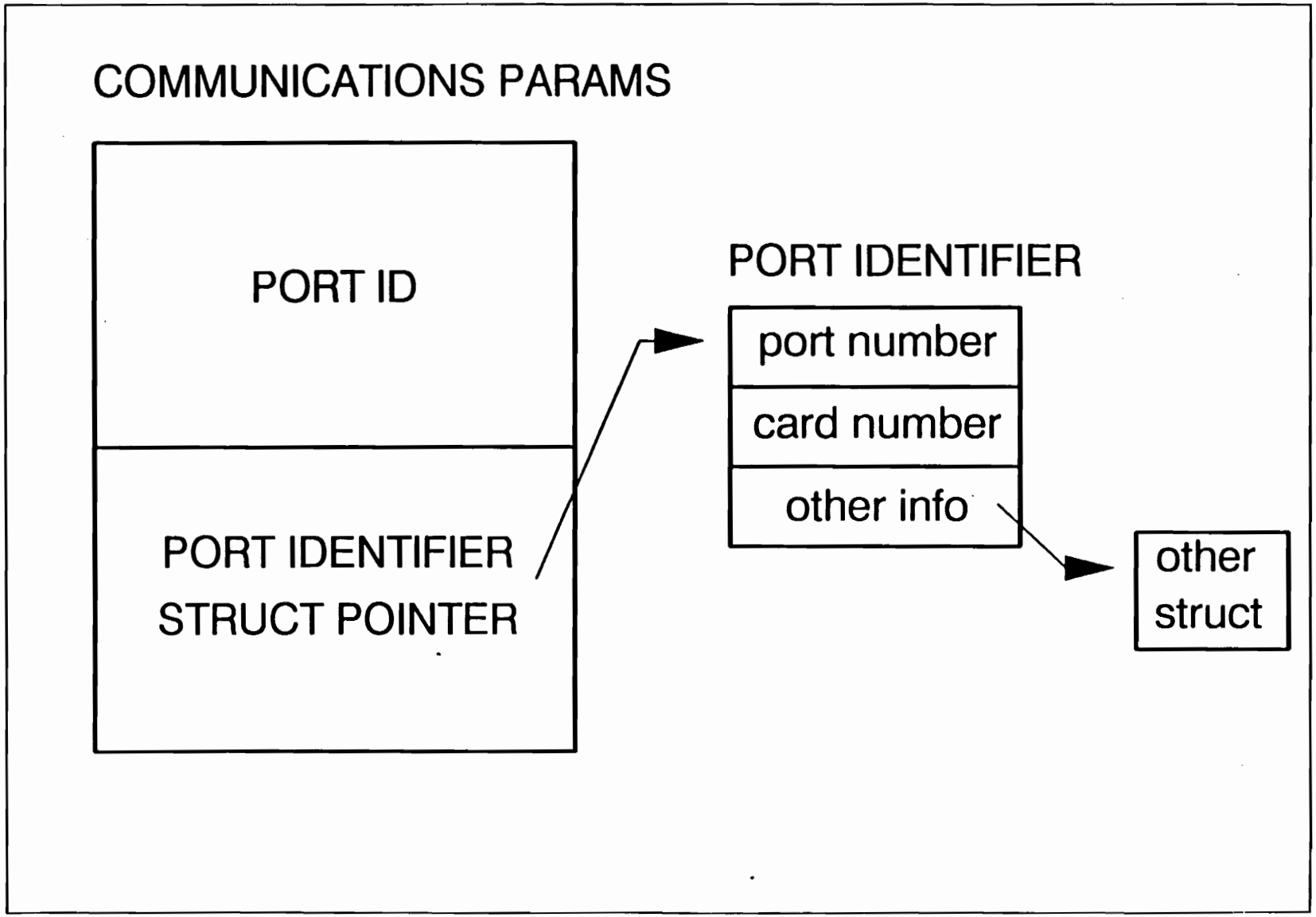


Figure 7.6 Communications Parameters Structure

7.4 Code Framework

The code framework was developed in a similar manner to the data structure framework. The device and environment dependent aspects were used in developing the code framework. Code must be specific to the device or the environment, but by creating it in a generic manner, the device code can be utilized in any environment and the environment code can be used for any device. This is the goal of the code framework.

7.4.1 CELL CONTROL CATEGORY FUNCTIONS

The defined cell control categories should have specific code associated with each one. This code follows the structure of the control software: a hierarchical manner. Each category has a function call which is generic regardless of environment or device. The parameter structures dictate the device control functions or environment functions called. Looking specifically at the cell control functions, the command identifier dictates the particular function within the scope. The cell control category functions are the only functions that interface to the user. These functions in turn call the environment, device, and other function code necessary to perform the operation.

7.4.2 ENVIRONMENT CODE

Environment dependent code covers the three areas identified above: (1) port communications, (2) event management, and (3) resource management.

The functions associated with each of these should be called in the same manner regardless of the environment to preserve a generic interface. If the operating system identifier is used, the specific code utilized is directed by this.

7.4.2.1 Port Communications Code

Port communications support the following functions: (1) initialize, (2) shutdown, (3) read, and (4) write. As stated earlier, the function call and parameter structure should be generic across environments. Table 7.1A identifies the generic and specific aspects of the communications code. Even though specific code and information is necessary for initialization and communication, generic names should be used. Typical function names are given in Table 7.1B. It is important at this level of code to keep the call format similar. This is the lowest level of code which many other functions utilize.

7.4.2.2 Event Management Code

The event management code handles management of the cell control function. Events such as starting the cell control function, checking its status, and ending the function fall into this category. Table 7.2A shows the generic and specific aspects of event management code. The operating system will dictate the manner of performing these tasks, the information, and the specific code. Again, the function call should be generic across environments. Table 7.2B lists typical function names for event management.

Table 7.1A Generic vs. Specific Communications Aspects

GENERIC:	**SPECIFIC:**
Parameters:	*Code*
line control: baudrate parity stop bits data bits	*Parameters:* port address additional cards
port name	
Function calls	

Table 7.1B Typical Communications Function Names

initialize	open port and set line control
shutdown	close port
write_byte	transmit one byte
write_array	transmit array
read_byte	read one byte
read_array	read array

Table 7.2A Generic vs. Specific Event Management Aspects

GENERIC:	**SPECIFIC:**
Function Calls	*Code:*
	cell control routine set up (function, task, thread)
	Parameters:
	information transfer routine status

Table 7.2B Typical Event Management Function Names

start_routine	start specific cell control routine
check_routine	check routine status
stop_routine	end routine

7.4.2.3 Resource Management Code

Resource management refers to handling the resources such as files, ports, or screens as described earlier. The operating system will dictate the code and some of the resources available. For the most part, the parameters can be fairly generalized for groups of resource management functions. The parameters for these functions can also be generalized. Table 7.3A identifies the generic and specific aspects of resource management code. Again, the calls for similar resource management functions can be the same across environments. Table 7.3B lists typical function names for this category.

7.4.3 DEVICE CODE

Device dependent code is very unique but can be structured in a similar manner as described above. This allows certain aspects of the code to be utilized across device boundaries. Code which is dependent on the device falls into three categories: (1) protocol, (2) data manipulation, and (3) control function.

7.4.3.1 Protocol Code

Protocol specific code handles the communications details of the device protocol. Although the protocol is device dependent, by isolating this code from any other device code or data interpretation, it can be used across device boundaries. A library of protocol functions could be set up for multiple devices. The protocol identified in the device structure signals the protocol functions to call.

Table 7.3A Generic vs. Specific Resource Management Aspects

GENERIC:	**SPECIFIC:**
Parameters:	*Code:*
file names port names	file handling port allocation semaphores
Function calls	

Table 7.3B Typical Resource Management Function Names

open_file	open file
read_file	read buffer from file
write_file	write buffer to file
close_file	close file
set_semaphore	set "in use" semaphore
clear_semaphore	clear "in use" semaphore
alloc_port	set port in use
release_port	release port in use

When structuring the protocol functions, the raw data is passed as a parameter. The specifics of the protocol are handled within the function and thus are transparent to the calling routine. Within the control function, lower level functions perform the steps associated with the specific protocol. The protocol code uses operating system port communications functions to pass information through the port. Because the port communications functions are accessed the same regardless of environments, no recoding needs to be done when the environment changes. Likewise, once a set of routines has been developed for a protocol, any device utilizing the same protocol can use the routines.

7.4.3.2 Data Manipulation Code

Because data is sent and received in many formats, functions are needed for manipulation and interpretation. The functions discussed such as protocol and read/write communications only pass data. In almost all cases, the data needs to be decoded. One example which emphasizes this is the use of ASCII coded hex data formatting. Many devices send and receive data in this format. Therefore, routines are necessary to convert the data. This particular function could be utilized as part of the protocol code, but the concept is applied to other situations. One device may store data in many different formats depending on the data's application. Therefore, the data must be converted to or interpreted into or from a particular format. These types of conversion functions should be created in an isolated format so that the code can be utilized across device and environment boundaries

7.4.3.3 Control Function Support Code

Each control routine requires numerous operations to perform its task. This is true when breaking down a control category, or when performing a specific control function. Separate functions should be created to execute these operations. Particular support functions may be used by several control functions. By structuring the code in a manner which facilitates a general parameter, the function can be used by any control routine which needs it.

7.5 Framework Overview

This section puts the pieces of the framework together in a general overview. The section discusses the three aspects of the framework: (1) control functions, (2) data, and (3) code. These three pieces come together to provide the software for cell control functions.

7.5.1 CONTROL FUNCTIONS

The first step in creating the cell control code is to determine the functional capability of the desired device. These should be easily assigned to one of the following categories: (0) intialize/shutdown, (1) control, (2) upload, (3) download, (4) read, and (5) write. Once the functions themselves are determined and grouped, the data format and code format can be tackled. The specific control software should follow the data and code framework described.

7.5.2 DATA OVERVIEW

Figure 7.7 displays the data structure from the high level generic structure to the specific structures which support it. By passing one high level structure, any information in the entire data framework can be accessed by any function. The information is readily available and easily passed.

The device, operating system, communications, and functions parameters should be isolated and structured according to the figure. On the operating system side, determine what parameters are necessary for communications, and resource and event management. On the device side, create structures which hold information pertaining to device type, communications line control, and specific data necessary for each control function within the cell control categories.

7.5.3 CODE OVERVIEW

When looking at the overall picture, the software structure follows a somewhat hierarchical format. Higher level commands are decomposed into smaller ones. These are similarly decomposed into specific tasks. Figure 7.8 shows how this decomposition takes place. Taking a top down look, the major cell control routines are broken into functions depending on the scope of the device. This function is further broken into smaller support operations. The protocol code is chosen dependent on the protocol specified in the device structure. The protocol uses operating system specific communications functions.

Taking a bottom up approach, the code can be structured in a format which builds on generic function calls. At the lowest level, communications routines have

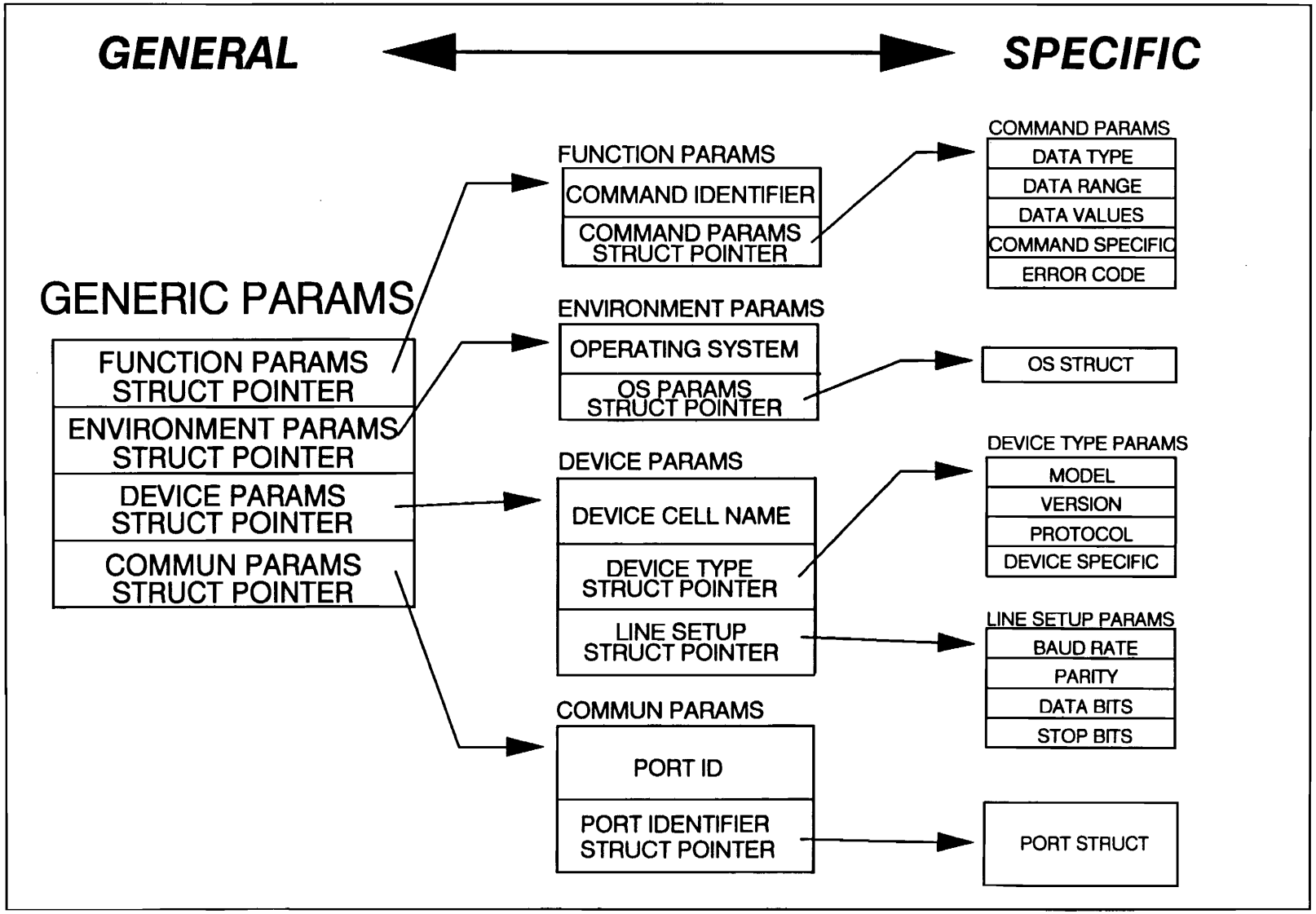


Figure 7.7 Data Framework

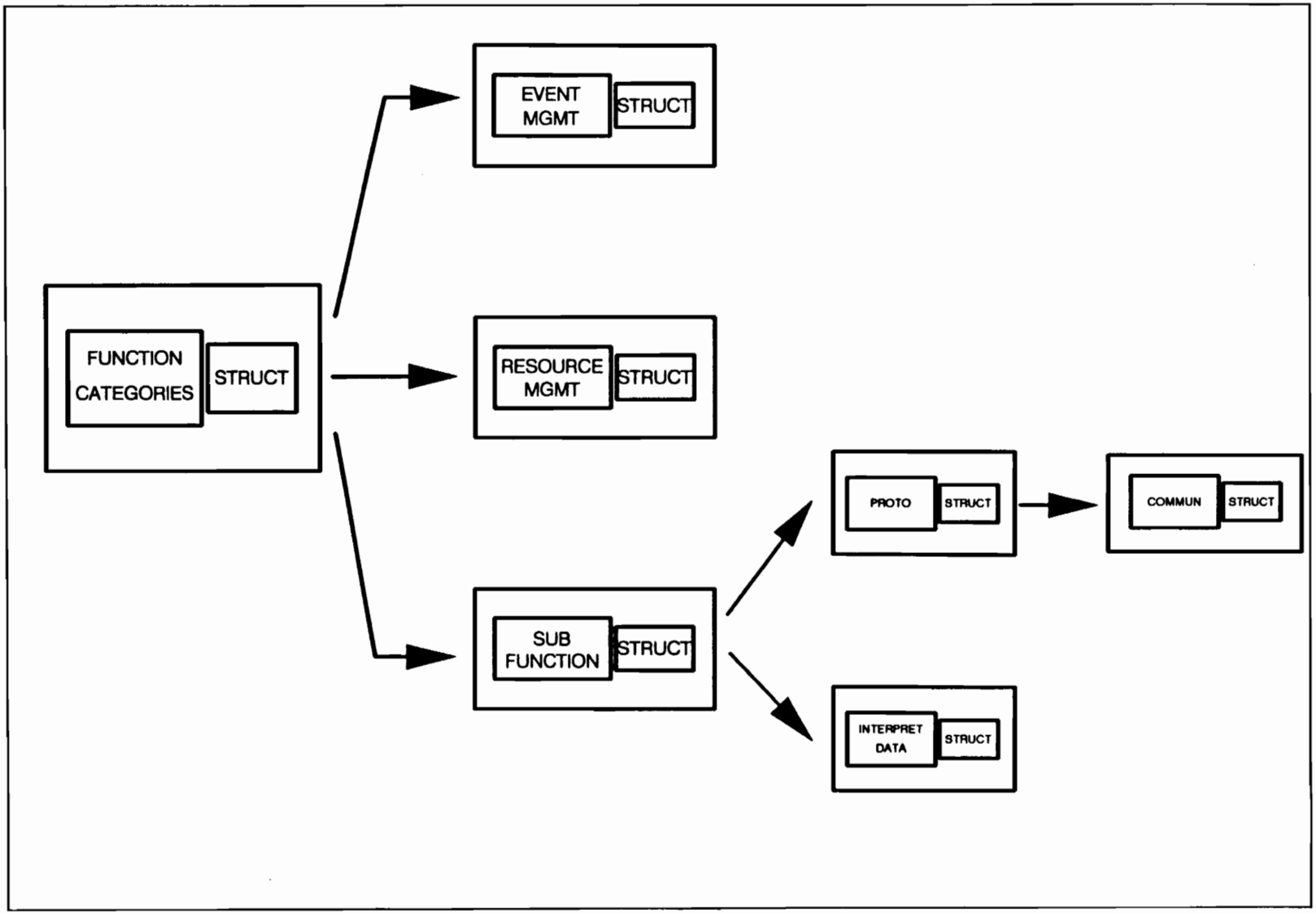


Figure 7.8 Code Framework

generic calls so that the protocol functions utilize the same call for communications regardless of the operating system. The code which corresponds to the operating system can be defined in an environment specific library and linked with the cell control software, or the operating system identification variable can dictate the specific code accessed. The protocol specific code has generic calls to the communications routines to perform the input/output functions. If the operating system changes, the protocol functions can be left unmodified. Also, protocol functions can be used across device boundaries.

The resource and event management code is written with generic calls for the same reason: the low level code can change without altering the code which calls them. The specific function, such as download, would utilize a generic function call to open the file to download. The underlying code structure is specific to the operating system, but the calling routine is unaware of this. Again, no changes would occur in the download function to accommodate an operating system change

7.6 Application Example

The framework will be applied in theory to the task of creating PLC cell control code. The actual code was not developed to follow this format, but this example strives to show the validity of the developed framework. The framework is applied to the PLC in the OS/2 operating environment. Each portion of the framework is applied, although all the functions of the PLC and OS/2 are not. In some cases, the capabilities of the PLC are not exhausted, only enough to convey the framework abilities. For the purposes of this example, the specific function referred to is reading timers.

7.6.1 CONTROL FUNCTIONS CATEGORIES

The twenty PLC cell control routines are grouped into the control and port function categories developed in the framework. This breakdown is shown in Table 7.4. The TI435 PLC supports all five function categories.

7.6.2 DATA FRAMEWORK

Most of the data structures can be set up independent of the control functions. Table 7.5 initializes the main function while Tables 7.5A - 7.5D detail the lower level structures.

The high level generic structure shown in Table 7.5 contains pointers to four structures. In turn, each of these has pointers or specific information. In Table 7.5A, the read timers function is assigned a command identifier value of 1. This would signal the read function to call the *read_tmr_ctr* function associated with this task. The parameters for the function are set in the *READ_T_C_PARAMS* structure. The structure also contains variables for the data read from the PLC and an error return code.

The environment parameters are relatively simple in OS/2. Table 7.5B shows the environment identified as "OS/2." The operating system parameters contains only the semaphore used to determine the status of the cell control routine.

The device type is specified in Table 7.5C. The name "PLC1" is given and the PLC type and model are identified in the *PLC_PARAMS* structure. The supported line control is also initialized in the device parameters structure.

Table 7.4 Example -- Control Function Categories

CATEGORY	*CONTROL FUNCTION*
0 initialize/shutdown	initialize shutdown
1 control	clear_ladder program_mode run_mode assign_password unlock_cpu lock_cpu last_error
2 upload	upload_TISOFT upload_user
3 download	download_TISOFT download_user
4 read	read_tmr_ctr read_TMRA_UDC read_io_range read_vmemory num_iopts
5 write	write_tmr_ctr write_TMRA_UDC write_io_range write_vmemory

Table 7.5 Example -- High Level Generic Structure

```
GENERIC_STRUCT:
    FUNCTION_PTR = FUNCTION_PARAMS
    ENVIRON_PTR = ENVIRON_PARAMS
    DEVICE_PTR = DEVICE_PARAMS
    COMMUN_PTR = COMMUN_PARAMS
```

Table 7.5A Example -- Function Parameters Structure

```
FUNCTION_PARAMS:
    CMD_ID = 1
    READ_TMR_CTR_PTR = READ_T_C_PARAMS

READ_T_C_PARAMS:
    type = "TMR"
    start = 10
    end = 20
    values
    error
```

Table 7.5B Example -- Environment Parameters Structure

```
ENVIRON_PARAMS:
    oper_sys = "OS/2"
    OPER_SYS_PTR = OS2_PARAMS

OS2_PARAMS:
    port_sem = "PORT1"
    funct_sem = "PLC1"
```

Table 7.5C Example -- Device Parameters Structure

```
DEVICE_PARAMS:
    DEVICE NAME = PLC1
    DEVICE_PTR = PLC_PARAMS
    LINE_SETUP_PTR = PLC1_SETUP_PARAMS

PLC_PARAMS:

    make = "TI"
    model = 435
    version = 1.6

PLC1_SETUP_PARAMS:
    baudrate = 9600
    databits = 7
    stopbits = 1
    parity = 0
```

Table 7.5D Example -- Environment Parameters Structure

```
COMMUN_PARAMS:
    port_id = "PORT1"
    PORT_ID_PTR = PORT1_PARAMS

PORT1_PARAMS:
    port = 1
    port_handle
```

The communications parameters are shown in Table 7.5D. The port is identified for subsequent operations as "PORT1." In the specific OS/2 environment, two variables are utilized: port and port handle. The port identifies which comm port is used. The port handle is returned from the operating system port open call.

7.6.3 CODE FRAMEWORK

The code is broken down into three groups: (1) control function categories, (2) environment, and (3) device. These specific function names for the TI435 PLC are shown in Tables 7.6A - 7.6C.

The control function names are the same as the category names. The high level generic structure is passed to each of these functions. The information within the specific function category would differ. Table 7.6A shows these five functions.

Environment specific code for OS/2 is shown in Table 7.6B. Because this is operating system dependent, it could be used with any factory device. The environment code covers resource management, event management, and port communications.

Resource management code in OS/2 would assure only one port user at a time. Also, any necessary semaphores, which were defined in the operating system parameters structure, can be set and cleared through resource management code.

Event management includes starting, checking, and ending the cell control routines. In the OS/2 environment, the *start_routine* function starts the routine in thread form. This function performs all the duties to accomplish this. The *check_routine* function queries a semaphore to check the status of the routine. This

Table 7.6A Example -- Function Category Code

```
initialize(GENERIC_STRUCT)
control(GENERIC_STRUCT)
upload(GENERIC_STRUCT)
download(GENERIC_STRUCT)
read(GENERIC_STRUCT)
write(GENERIC_STRUCT)
```

Table 7.6B Example -- Environment Code

RESOURCE MANAGEMENT:

allocate_port(GENERIC_STRUCT)

set_semaphore(GENERIC_STRUCT)

release_port(GENERIC_STRUCT)

close_semaphore(GENERIC_STRUCT)

EVENT MANAGEMENT:

start_routine(GENERIC_STRUCT)

check_routine(GENERIC_STRUCT)

end_routine(GENERIC_STRUCT)

PORT COMMUNICATIONS:

read(port_id, data)

write(port_id, data)

is necessary due to the multitasking nature of OS/2. The *end_routine* function takes care of releasing any resources tied up by the cell control routine. These could be ports, files, screens, etc.

The device specific code functions for the *read_tmr_ctr* function are shown in Table 7.6C. The control category would call the *read_tmr_ctr* function based on the function parameters structure command identifier. For this function, the protocol functions *enquiry*, *send_header*, and *read_block* are needed. The *ascii_to_hex* function is first needed to transform the data read from the PLC. In the TI435 PLC, the timers and counters are stored as binary coded decimal. Therefore, the *bcd_to_hex* function is needed to convert the data into the correct format for the user.

7.6.4 EXAMPLE SUMMARY

The above example shows the framework applied to the TI435 PLC. The functions were first categorized. Then the data and code were then set up in a generic structure. The functions outlined in the example do not necessarily correspond to the actual code developed. But the software could be structured in this format with minor modifications. The functions and parameters logically fit into the categories defined in the framework.

This example applies one device and one environment to the developed framework. Other devices follow a similar format. The robot functions easily fall into the function categories. The supported protocol could be broken down into functions. In reference to the milling machines, the support functions fall into the control categories, although not all of the categories are supported. The protocol

Table 7.6C Example -- Device Code

FUNCTION:

read_tmr_ctr(GENERIC_STRUCT)

PROTOCOL:

enquiry(specific_data)

send_header(specific_data)

read_block(specific_data)

DATA INTERPRETATION:

ascii_to_hex(specific_data)

bcd_to_hex(specific_data)

for the Dyna 2200/2400 is much simpler, but still follows a standard which fits the framework.

7.7 Benefits

The goal of the framework is to provide benefits from two perspectives of the software: user and developer. Some of the benefits achieved fall into both categories. The main emphasis of the framework is to provide a common interface. This is true for both the user and developer.

From the user's perspective, the interface to the software should be the same regardless of device or environment. The underlying code is very specific to the both, but the user is unaware. The user must provide specific parameter information for the environment, device, and function. The information is set up in the same format so the user can easily define the required information. This commonality is also seen when the software is expanded. The results of the expansion do not affect what is already in use. Performance should not be hindered by introducing the expansion or modification.

The developer's benefits are similar to the user's but at a more intricate level. By following the framework, development, expansion, and modification are done in an organized and configurable manner. The present code does not need restructuring to accommodate new devices or capabilities. New functions can be added to the libraries with no modification to old libraries. Through the use of structures and pointers, specific information for new devices and functions can be accessed without modifying the structure of any existing data.

The modular format of the software allows certain types of functions to be used across several boundaries. If code is configured the same way, upper level code can call lower level code without changing. The lower level specific code may be modified, but the higher level code can remain the same. For example, this allows the same high level control function code to be used in several operating system environments. Similarly, communications, protocols, and data conversion functions can be used across device boundaries.

8.0 CONCLUSION

8.1 Results

The results of this research fall into two categories. The first is a knowledge gained perspective. This refers to the experience achieved through the research. The second category refers to the knowledge provided from the research. This perspective focuses on what aspects make the research valuable.

By developing the TI435 PLC software, an extensive knowledge of generating PLC control communications code was gained. This includes details of the PLC in terms of its internal set up and its external communications techniques. In addition, two other device classes were researched: robots and NC machines. The communications capabilities and restrictions of the devices in each class provided the background necessary in developing the framework.

Communications control code was generated for the TI435 PLC. Although this code does not follow the framework, it acted as a learning tool for the framework development. This developed software provides functions to perform initialize, control, upload, download, read, and write cell control communications for the PLC. The code source listing is available in a separate document.

The above results provided the basis for developing the final result -- the framework. This framework provides a method to generalize communications interfaces to the above classes of devices. The framework is a valuable tool in view of the many different factory control devices which exist. The framework allows the

communications interface to the device to be specific. But the user interface remains general regardless of the device.

The advantage of the framework is that it does not adhere to any one standard or device. The framework could be applied to any standard of communications. In cell control, there is always a need to interface directly with a factory device. Even if the communication methods are standard, the interpretation of the information from each device is not. The framework allows each part of the communications and control aspects to be handled separately.

8.1 Recommendations

The most logical recommendation is that the framework be implemented when creating communications interfaces to devices. It is inevitable that the interface will take place. Therefore the framework should be applied to assure the developed code will be flexible. By following the framework, and additions to the system should not cause recoding. More software modules will be added, but the base code will not change.

The framework was a result of the research into several classes of factory devices and operating environment. The research provided the basis for the framework development. The framework should be applied to other classed of devices or operating platforms. Through its implementation, its validity should be apparent.

The framework provides a model for the development of low level communication interfaces for generic work cell control. The software developed from this framework should provide a base for a generic cell controller. The development of the cell controller itself is the next logical step. This research completes the interface for the lower level of the control hierarchy.

9.0 REFERENCES

[1] Bedworth, David D., Mark R. Henderson, and Philip M. Wolfe, Computer-Integrated Design and Manufacturing, New York:McGraw-Hill Inc., 1991.

[2] Bizzarri, Paul M. "Real-Time Communication Interfaces to Popular Brands of PLCs," 1987 PLC Conference, pp. 149-160.

[3] Bradley, Michael S. "Cell Control -- What Do Users Really Want," 1989 PLC Conference, pp. 329-340.

[4] Bresson, Matthew J. and Roger G. Rupp. "CELL CONTROL: How Can You Effectively Implement The Solution?' 1989 PLC Conference, pp. 293-303.

[5] Clark, Karen E. "The Role of Cell Control in Flexible Automation Environments," How To Achieve Integration Through Cell Controllers, April 24-25, 1990, Oak Brook, Illinois.

[6] Cobb, Peter. "Implementation of Cell Control in a Metal Fabricating and Body Assembly Plant," How To Achieve Integration Through Cell Controllers, April 24-25, 1990, Oak Brook, Illinois.

[7] Coflan, Barry F. "Control, Coordination, Communication," 1986 PLC Conference.

[8] Dror, Asael. The Waite Group's OS/2 Programmer's Reference, Indianapolis:Howard W. Sams & Company, 1988.

[9] Dyna Electronics, Inc., DM2400/2200 Programming Manual, 1984.

[10] Ekberg, Brandon. "Cell Control for Discrete Electronics Manufacturing," How To Achieve Integration Through Cell Controllers, April 24-25, 1990, Oak Brook, Illinois.

[11] Falk, Herbert, Ralph Mackiewicz, and Jonathan Martin. "Application and Maintenance Requirements of Cell Control," 1986 PLC Conference.

[12] FASTech Integration, Inc. "CellWorks" Marketing Brochure.

[13] Gofton, Peter W. Mastering Serial Communications, San Francisco:Sybex, 1986.

[14] Gokorsh, S. "National Electrical Manufacturers Association Classification System for Cell Controllers," Achieving Integration Through Cell Controllers, April 26-27, 1989, Detroit, Michigan.

[15] Greenwood, Nigel R., "Industrial Computers For Cell Control," 1987 PLC Conference.

[16] Groover, Mikell P., Mitchell Weiss, Roger N. Nagel, and Nicholas G. Odrey, Industrial Robotics, New York:McGraw-Hill, 1986.

[17] Harkrider, Eddie. "Flexible Manufacturing Cell Integration," Achieving Integration Through Cell Controllers, April 26-27, 1989, Detroit, Michigan.

[18] Honeywell Industrial Automation Systems. "Manufacturing Automation System Controller MAS/C" Marketing Brochure.

[19] HURCO Manufacturing Company, Inc., CNC BMC-20 User's Manual, 1984.

[20] IBM, Plant Floor Series Distributed Automation Edition General Information, 1989.

[21] IBM Manufacturing System, AML/Entry Version 4 User's Guide,1985.

[22] IBM Manufacturing Systems Products, Realtime Interface CoProcessor C Language Support Version 1.02 User's Guide, 1989.

[23] IBM Manufacturing Systems Products, Realtime Interface CoProcessor Developer's Kit Version 2.01, 1988.

[24] IBM Manufacturing Systems Products, Realtime Interface CoProcessor DOS Support Version 1.03 User's Guide, 1990.

[25] IBM Manufacturing Systems Products, Realtime Interface CoProcessor Extended Services User's Guide, 1990.

[26] IBM Manufacturing Systems Products, Realtime Interface CoProcessor Operating System/2 Support Version 1.03 User's Guide, 1989.

[27] ITP Enterprise Software, Inc. "MainstreamTM Application Integration Platform For Open Factory Systems."

[28] Johnson, David G. "Cell-Control - A Migration," 1988 PLC Conference, pp. 91 - 102.

[29] Jones, Albert T. and Charles R. McLean. "A Proposed Hierarchical Control Model For Automated Manufacturing Systems," Journal of Manufacturing Systems, Volume5/No. 1, pp. 15 - 25.

[30] Lafore, Robert and Peter Norton. Peter Norton's Inside OS/2, New York:Brady Books, 1988.

[31] Lafore, Robert. The Waite Group's Microsoft C Programming for the PC, Indianapolis:Howard W. Sams & Company, 1989.

[32] Larin, David J. "Cell Control: What We Have, What We'll Need," Manufacturing Engineering, January 1989, pp. 41-48.

[33] Mechler, Bob. "Cell Controller's Evolution to a Product," 1987 PLC Conference, pp. 59 - 73.

[34] Moskowiz, David, Kathy Ivens, and Noel Bergman. Converting Applications to OS/2, New York:Brady Books, 1989.

[35] Motorola. "MDO-68k Cell Controller" Marketing Brochure.

[36] Nguyen, Thuyen and Rober Moskal. Advanced Programmer's Guide to OS/2, New York:Brady Books, 1989.

[37] Northern Research and Engineering Corporation. "WorkCell MANAGER Computer Monitoring and Control for Discrete Manufacturing," Marketing Brochure.

[38] Quatse, Jesse T. "Requirements for Real-Time Cell Control" 1986 PLC Conference, pp. 99-113.

[39] Skukas, Paul A. "A Cell Control Application: The Wirebond Cell," How To Achieve Integration Through Cell Controllers, April 24-25, 1990, Oak Brook, Illinois.

[40] Stallings, William, Handbook of Computer-Communications Standards, Volume 2 Local Network Standards, Indianapolis: Howard W. Sams & Company, 1987.

[41] Texas Instruments Inc., 405 Series Language and Communication Protocol Specification, (2593568), Johnson City, Tennessee: Texas Instruments Incorporated, 1989.

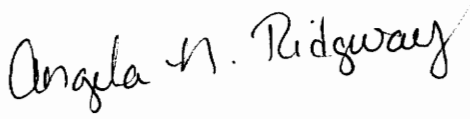
[42] Texas Instruments Inc., Series 405 Data Communication Module User Manual (2593572-0001), Johnson City, Tennessee: Texas Instruments Incorporated, 1990.

[43] Texas Instruments Inc., Series 405 User's Manual (405-8101), Johnson City, Tennessee: Texas Instruments Incorporated, 1989.

[44] Wuest, Robert E. "Trends in Cell Control," 1989 PLC Conference, pp. 305 - 314.

VITA

Angela Nadine Ridgway was born January 30, 1965 in Portsmouth, Virginia to Robert and Elizabeth Ridgway. She received a Bachelor's degree in Industrial Engineering and Operations Research from Virginia Polytechnic Institute and State University in 1987. After a brief experience as an engineer in industry, she returned to pursue a Master's degree. With this experience now complete, the next one begins.



THE DEVELOPMENT OF LOW LEVEL COMMUNICATION INTERFACES FOR GENERIC WORK CELL CONTROL

by

Angela Ridgway

Committee Chairman: M. P. Deisenroth
Department of Industrial and Systems Engineering

ABSTRACT

As the desire for factory automation increases, so has the need to integrate machinery within the factory. More specifically, this integration is gaining importance in the area of manufacturing work cells. Many ideas exist about what functions a cell controller should perform and how it should interact with its environment. The functions utilized by the cell controller many vary depending on the type of machinery but similar tasks are usually performed. The complexity of the cell controller increases due to the differences in functional capability caused by machine intelligence or vendors specifications.

The objective of this research was to create a framework which can be followed when developing low level machine specific cell control communications. The framework would assist the user in defining and structuring the information and functions associated with a particular device and operating environment. This framework will act as a guide in the creation of generic cell control communication functions. The purpose of the framework is to act as a guide in

the development of low level base routines which interface to various classes of factory devices. It is impossible to create a completely generic base which will interact with every device. However, it is possible to develop this base following a structured format which facilitates generic work cell control.