1-1-2005

Design and Implementation of a Networked Control System

Mi Chen
Eastern Illinois University

THESIS REPRODUCTION CERTIFICATE

TO: Graduate Degree Candidates (who have written formal theses)

SUBJECT: Permission to Reproduce Theses

The University Library is receiving a number of request from other institutions asking permission to reproduce dissertations for inclusion in their library holdings. Although no copyright laws are involved, we feel that professional courtesy demands that permission be obtained from the author before we allow these to be copied.

PLEASE SIGN ONE OF THE FOLLOWING STATEMENTS:

Booth Library of Eastern Illinois University has my permission to lend my thesis to a reputable college or university for the purpose of copying it for inclusion in that institution's library or research holdings.

[illegible]   08/11/2005

Author's Signature   Date

I respectfully request Booth Library of Eastern Illinois University **NOT** allow my thesis to be reproduced because:

____

____

____

______________________________   ______________

Author's Signature   Date

This form must be submitted in duplicate.

Design and Implementation of a Networked

Control System

(TITLE)

BY

Mi Chen

THESIS

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF

Master of Science in Technology

IN THE GRADUATE SCHOOL, EASTERN ILLINOIS UNIVERSITY
CHARLESTON, ILLINOIS

2005
YEAR

I HEREBY RECOMMEND THAT THIS THESIS BE ACCEPTED AS FULFILLING
THIS PART OF THE GRADUATE DEGREE CITED ABOVE

8/10/05	Peter P. Liu
DATE	THESIS DIRECTOR
Aug. 10, 2005	[illegible]
DATE	DEPARTMENT/SCHOOL HEAD

THESIS COMMITTEE

Peter P. Liu

Peter Ping Liu,
Ph.D., P.E., OCP, C.Q.E., and C.S.I.T.
Professor
Thesis Director
Graduate Coordinator
School of Technology

8/10/05
Date

Sam Guccione

Sam Guccione, Ed.D, C.S.I.T
Professor
School of Technology

August 10, 2005
Date

[illegible]

Larry D. Helsel, Ed.D.
Professor
School of Technology

8/10/05
Date

Abstract

One of the major challenges faced by modern control systems is to integrate the computing, communication, and control into different levels of operations and information processes. The current practice to tackle these control problems focuses on distributed implementation of control systems. A Networked Control System (NCS) is one type of distributed control systems where the control loop is operated over a communication network.

This thesis discusses the structure and hierarchical model of a typical NCS, The prototype combines both network and control theory, addresses the detailed procedures on design, configuration, integration and implementation of a process control via communication network. The implementation considerations in designed NCS include serial communication, network communication, human machine interface programming, client/server modeling, and conformance testing, etc. The designed system can be used for the further controller design to investigate the interaction between network configuration and control parameters.

Acknowledgements

I would like to take this opportunity to thank all of people who have given me helpful suggestions, motivation, and encouragement in my graduate study at the Eastern Illinois University.

I would first like to express my utmost gratitude to my advisor, Dr. Ping Liu. Dr. Ping Liu has continuously provided me with his inspiration, encouragement, and support since I came to the Eastern Illinois University. He has also enriched my research experiences and guided me from a person who was completely blind in the research area to a person who knows a little more on how to use technical method to perform a research work.

I would also like to thank Dr. Sam Guccione for his guidance, contributions and help with this thesis. He has been a constant source of helpful questioning, and a continual source of fresh ideas in the process of earning my degree.

I would also like to thank Dr. Larry Helsel for his valuable comments on this thesis and be on my committee.

Furthermore, I gratefully acknowledge the CimQuest Inc for their support of Serial DF1 ActiveX Control Component with this study.

Finally, but most importantly, I would like to express my deepest appreciation to my parents, Mr. Boan Chen and Mrs. Liru Fu, for their endless love, support, understanding, and encouragement, and many other professors, colleagues and friends that I could not mention them all here. Thank you!

Table of Contents

List of Tables

List of Figures

CHAPTER 1

Introduction

One of the major challenges faced by modern control systems is to integrate the computing, communication, and control into different levels of operations and information processes (Abdullah & Chatwin, 1994). These complex control systems may include a large number of devices interconnected together to perform the desired operations. Intelligence and decision making can be moved out of the central control units and distributed into controllers located near the controlled devices (Lian, Moyne, & Tilbury, 2000a).

For many years, the point-to-point architecture has been widely used in the industrial and manufacturing control systems. This centralized communication architecture had a single central control unit, in which each system controller is directly wired to other devices, such as sensors, or actuators. However, this centralized communication architecture is no longer suitable as it lacks a common communication protocol and the levels of interoperability are generally not defined. It is also difficult to handle expansion of physical setups and system functionality in complex control applications.

The current practice to tackle modern manufacturing control problems focuses on distributed implementation of control systems. Whereas a common control algorithm is generally defined across the manufacturing system, individual physical nodes can be operated independently (Lian, Moyne, & Tibury, 1999). These nodes cooperate with one another, communicating through a shared data network channel. These systems are able to accomplish various tasks with limited reconfiguration and to provide a way to improve the efficiency of diagnostics and maintenance (Altun, Topaloglu, Saygin, & Bayrak, 2001).

A Networked Control System (NCS) is one type of distributed control systems where the control loop is operated over a communication network. With NCS, decisions and control functions can be distributed among controllers on the network (Lian, et al., 1999). In contrast to the traditional point-to-point communication, NCS offers more efficient re-configurability, high system testability and better resource utilization. It reduces not only the installation and maintenance cost of the control system, but also the floor space needed for electrical cabinets. Moreover, applications connected through a network can be remotely controlled from a long-distance source, especially in environments where electronically controlled machines are not closely located.

1.1 Statement of the Problems

It has been realized that the traditional point-to-point control architecture has limited the development of modern manufacturing control systems. Implementing a complex control application over a point-to-point architecture may cause a number of problems, which can potentially degrade system effectiveness. Typical limitations in point-to-point architecture are as follows:

1. Adding, deleting or interchanging system components are difficult.

2. Remote access is not accessible because of physical constraints.
3. Higher intelligence devices, such as smart sensors and actuators are not supported.
4. Troubleshooting is time consuming.
5. Wiring work and maintenance costs are extremely high.

As an alternative to point-to-point architecture, Networked Control System has received more and more attention recently because of its ability to offer more efficient re-configurability, high system testability and better resource utilization.

1.2 Statement of the Purpose

The purpose of this research is to study the detailed procedures on designing and implementing a Networked Control System. The designed system can be used for the further controller design to investigate the interaction between network configuration and control parameters.

1.3 Definition of Terms

1. ASCII – American standard code for information interchange, an 8-bit code for character representation.
2. Client – A node or software program (front-end device) that requests services from a server.
3. Client/Server model – A common way to describe network services and model user processes of those services.

4. Ethernet – A baseband Local Area Network (LAN) specification invented by Xerox Corporation and developed jointly by Xerox, Intel, and Digital Equipment Corporation. Coaxial cable carries radio frequency signals between computers at a rate of 10 megabits per second.

5. Programmable Logic Controller (PLC) – A device used to automate monitoring and controlling of industrial processes. It can be used stand-alone or in conjunction with a Supervisory Control and Data Acquisition (SCADA) or other systems.

6. Human Machine Interface (HMI) – The user interface that allows interaction between the user and the machine. The term HMI is typically used in industrial and process control applications.

7. Binary Exponential Backoff (BEB) – In CSMA/CA (Carrier Sense Multiple Access with Collision Avoidance) networks and in CSMA/CD(Carrier Sense Multiple Access with Collision Detection) networks, the algorithm used to schedule retransmission after a collision such that the retransmission is delayed by an amount of time derived from the slot time and the number of attempts to retransmit.

8. ActiveX Control – A component program object that can be re-used by many application programs within a computer or among computers in a network. The technology for creating ActiveX controls is a part of

Microsoft's overall ActiveX set of technologies. Component Object Model (COM) is a major part of this technology.

1.4 Limitations

The following is a list of limitations in this study.

1. Nondeterministic nature of the Ethernet-based network does not guarantee that a particular message will not collide and eventually be dropped.
2. Time delay over the communication network may affect data transaction between the controller and the control process.

1.5 Delimitations

The following lists delimitations in this study.

1. Ethernet-based network was used as the communication network.
2. The networked control system prototype consists of one process, one server, and one client.
3. Operating environment of the server and client is based on Microsoft Windows 2000.
4. Human Machine Interface was programmed using Microsoft Visual Basic.
5. The program needed for the PLC controller was written in the RSLogix500 programming software.
6. The program controlling a robot was written in the PC robotics programming software.

CHAPTER 2

Literature Review

2.1 Modern Control Systems

The current trend of modern control systems focuses on integrating computing, communication, and control into different levels of operations and information processes (Abdullah, et al., 1996). When integrated, processes are able to share information, initiate actions, and accomplish various tasks with comparably small reconfiguration work. A simple example of control systems is a robot controller and a programmable logic controller (PLC) working together to control a single machine. A complex example is an entire manufacturing plant with hundreds of workstations, including computer-numerically-controlled (CNC) machines, computer-aided design (CAD) tools, supervisory controllers, and intelligent monitoring devices.

Figure 2-1 shows an altitude control system in aircraft control as another typical example of a complex control system. In order to maintain the altitude during an autopilot mode, various sensors and control surfaces among other control components are distributed over the aircraft. Three subsystems including flaps, the elevator, and the engine, must perform and collaborate together to achieve an overall system task.

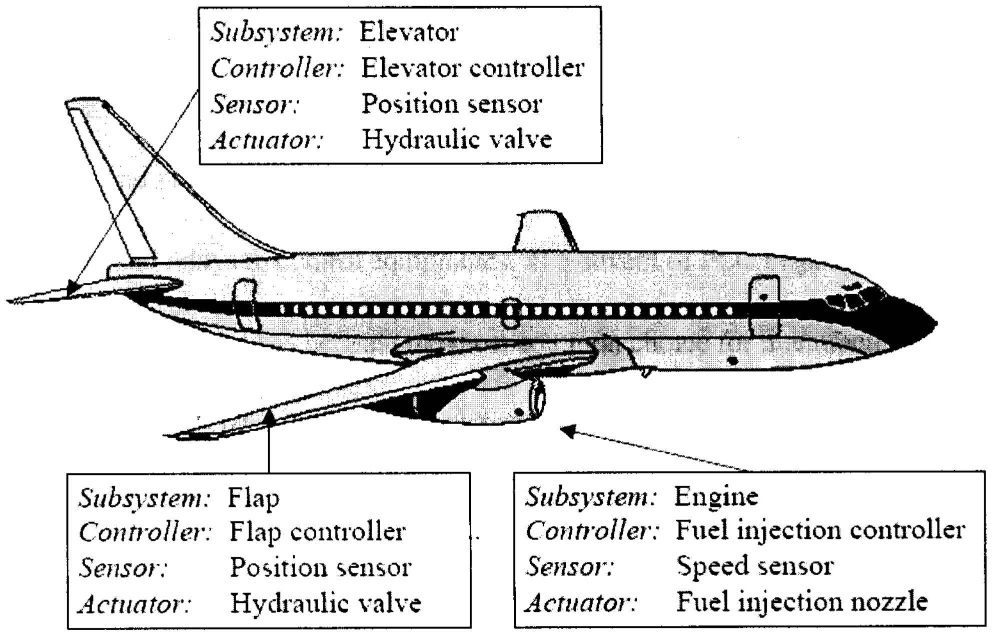

Subsystem: Elevator
Controller: Elevator controller
Sensor: Position sensor
Actuator: Hydraulic valve
Subsystem: Flap
Controller: Flap controller
Sensor: Position sensor
Actuator: Hydraulic valve
Subsystem: Engine
Controller: Fuel injection controller
Sensor: Speed sensor
Actuator: Fuel injection nozzle

Figure 2-1. Altitude control system of an airplane (Tipsuwan, Y., 2003, p. 17)

Common features of these complex control systems include a large number of devices interconnected together to perform desired operations, and a large physical area of coverage (Lian, 2001). Advanced control systems have been widely applied in the industrial automation, building automation, office and home automation, intelligent vehicle system, and advanced aircraft and spacecraft.

2.2 Programmable Logic Control

Prior to the development of electronic control solutions, electromechanical relays were the standard means of sequential control. These relays allow power to be switched on and off without a mechanical switch (Hugh, 2004). It is common to use relays to make simple logical control decisions. However, additional components and cross wiring could result in a substantial amount of downtime for the production line and a potentially high cost to implement.

Programmable Logic Controller (PLC) revolutionized the control by simplifying its process. PLCs are the control hubs for a wide variety of automated systems and processes. They contain multiple inputs and outputs to simulate switches and relays to control equipments. The advent of PLC began in the 1970s (Hugh, 2004), and it has become the most common choice for manufacturing controls, and will probably remain predominant for some time to come.

PLC I/O channel specifications refer to total number of points, number of inputs and outputs, ability to expand, and maximum number of channels. PLCs may be specified by any possible combination of these values. Expandable units may be stacked or linked together to increase total control capacity. Maximum number of channels refers to the total number of input and output channels in an expanded system.

PLC system specifications include scan time, number of instructions, data memory, and program memory. Scan time is the time required by a PLC to check the states of its inputs and outputs. Instructions are standard operations (such as math functions) available to PLC software. Data memory is the capacity for data storage. Program memory is the capacity for control software.

Possible inputs for PLCs may include DC, AC, analog, thermocouple, frequency or pulse, and interrupt inputs. The inputs are electrically isolated from the CPU power and the CPU data bus. Outputs for PLCs include DC, AC, relay, analog, frequency or pulse. As it is with inputs, outputs are electrically isolated from the CPU power and the CPU data bus. PLCs may be programmed through front panel, hand held interface, or a computer. They may also have computer interface options and network ability.

2.3 Human Machine Interface

Human machine interface (HMI) is where people and technology meet. Typically, the term HMI is used to refer to devices that display machine or process information and provide a means for entering data or commands. It retrieves information from machines, which allows operators to monitor record and control the system through interfaces such as image, keyboard, touch screen and so on.

The required functionality of an HMI will vary based upon the type and complexity of product, the type of machine, the skills of the operator, and the degree of automation of the machinery. Typical types of functionality stated by Weber (1999) are shown in Table 2-1. For example, "Graphic Displays" function provides information about machine operation and status to the operator.

Table 2-1

Typical Types of HMI Functionality

Functionality	Purpose
Graphic Displays	To provide information about machine operation and status to the operator in a format that allows for easy interpretation and determination of needs for action
User Input	To facilitate inputs from the operator to adjust machine operation, perform machine setups, and respond to events
Data Logging & Storage	To provide for the storage of historical machine operating data for part traceability and analysis of ways to improve quality, productivity, and uptime. Also used to store and retrieve machine setup data where needed.
Trending	To provide a means for visual analysis of data on current or past machine operation
Alarming	To provide notification to the operator of abnormal operating conditions and events.

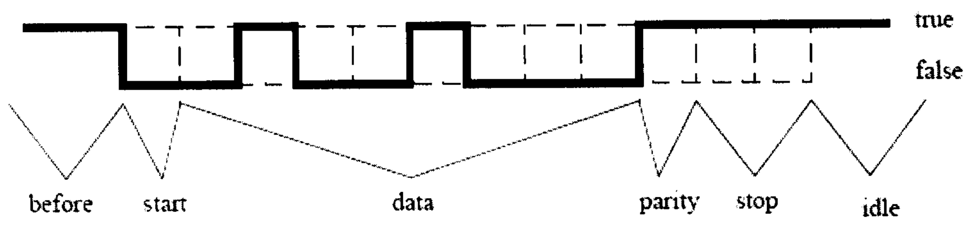
2.4 Serial Communication

Multiple control components may be used for complex processes. For these controllers to work together, they must communicate with each other. The simplest form of communication is a direct connection between two computers. Data can be transmitted one bit at a time in series, which is known as serial communication.

All IBM PC and compatible computers are typically equipped with standard serial ports. For example, keyboard port, RS-232 serial port, USB port, network port, and modem port are all serial communication ports.

In serial communications, one single bit of data is sent at a time over the medium. This only requires a single communication channel, as opposed to eight channels to send a byte using a parallel port. While it takes eight times as long to transfer each byte of data this way than parallel byte transfer, only a few wires are required. In fact, two-way (full duplex) communications is possible with only three separate wires - one to send, one to receive, and a common signal ground wire.

Serial communication is frequently characterized by parameters of baud rate, parity, data bits, and stop bit. For example, a set of serial parameter of (9600, N, 8, 1) indicates a baud rate of 9600 bps, no parity, 8 data bits, and 1 stop bit. A typical serial data byte equal to 00010010 is shown in Figure 2-2.



Descriptions:

before - this is a period where no bit is being sent and the line is true.
start - a single bit to help get the systems synchronized.
data - this could be 7 or 8 bits, but is almost always 8 now. The value shown here is a byte with the binary value 00010010 (the least significant bit is sent first).
parity - this lets us check to see if the byte was sent properly. The most common choices here are no parity bit, an even parity bit, or an odd parity bit. In this case there are two bits set in the data byte. If we are using even parity the bit would be true. If we are using odd parity the bit would be false.
stop - the stop bits allow a pause at the end of the data. One or two stop bits can be used.
idle - a period of time where the line is true before the next byte.

Figure 2-2. A serial date byte (Hugh, 1996, p.121)

RS-232C is the most common standard defined by Electrical Industries Association (EIA). RS-232 stands for Recommend Standard number 232 and C is the latest revision of the standard. Serial ports on most computers use a subset of the RS-232C standard. The full RS-232C standard specifies a 25-pin "D" connector of which 22 pins are used. Most of these pins are not needed for normal PC communications, and indeed, most new PCs are equipped with male D type connectors having only 9 pins.

A server computer is able to communicate with PLC through RS-232 serial port. For example, PLCs of Allen-Bradley's MicroLogix family are capable of serial communication at standard baud rates from 300 to 38400, with 9600 being the default values specified by the factory. In a typical connection between PLC and PC, the RS-232 cable may be connected to the channel serial connector on the PLC processor, and to the COM1 port on the computer.

2.5 Communication Protocol for Allen Bradley's PLCs

Based on the Open System Interconnection (OSI) model, physical layer is a set of cables and interface modules that provide a channel for communication between nodes. A node is a connection point onto a network, typically containing a unique address (Allen Bradley, 1996). At the physical layer level, Allen Bradley's MicroLogix family PLCs can be communicated with the server computer by means of RS-232 communication port. The framework of the data packet transmitted between a server and a PLC is constructed with ASCII control characters. In order to talk to the PLC, the protocol for messaging of the PLC must be understood. The protocol provides the structure that encapsulates the data bytes, ensuring that the data bytes arrive undistorted at the correct destination (Leonik, 2000).

Figure 2-3 shows a data communication model between server computer and PLC. Two layers of software are involved in this communication. They are data-link layer and application layer. Details of the two layers are explained as follows.

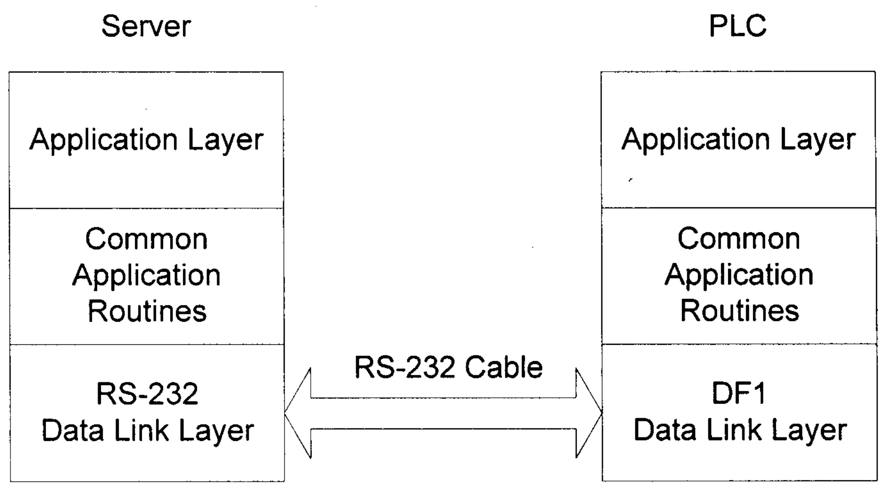


Figure 2-3. Mode of data communication between server and PLC

2.5.1 Data link layer

Data link layer controls the flow of communication over the physical link and determines the encoding of data on the physical medium (Allen Bradley, 1996). In order to connect a PLC to a computer, the proper protocol character sequences and program driver on the physical link must be understood. DF1 protocol is an Allen-Bradley data-link layer protocol that interprets signals transmitted over a physical link. DF1 protocol encapsulates the date bytes from one end of the link to the other, indicating failure with an error code. It has no concern for the content, function, or the ultimate purpose of the message (Allen Bradley, 1996). The transmission format is as follows:

Start bit—8 data bit (0~7)—no parity—stop bit

DF1 protocol provides two modes of communication: full-duplex and half-duplex. Full-duplex allows two-way simultaneous transmission over a point-to-point link. Half-duplex is a multidrop protocol for master/ multiple slaves' network, but communication only takes place in one direction at a time.

2.5.2 Application layer

Application layer controls and executes the actual commands specified in the communication between nodes (Allen Bradley, 1996). Application layer interfaces with user processes and databases, interprets commands, and formats user data into packets. The application layer depends upon the type of node the application is running on since it must interface with the user process and interpret the user database.

All messages on a network have the same fundamental structure, regardless of their function or destination. PLC messaging consists of two types of messaging bytes: protocol bytes and data bytes. There are two types of application programs in the application layer: command sender and command receiver (Allen Bradley, 1996).

Command sender sends command messages, specifying which command function to execute at a particular remote node. Figure 2-4 shows the software logic for implementing command message transaction. The command sender first sends a command, initializes the timer, and waits for acknowledgment (ACK) from the PLC. If an ACK is detected, the command sender then starts to process data. Otherwise, it proceeds to transmit out the next request.

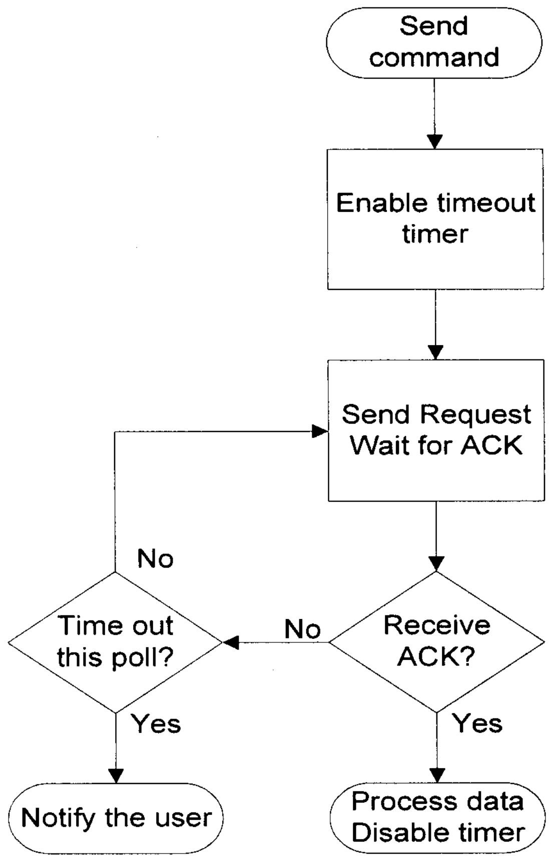


Figure 2-4. Flowchart for implementing command message transaction

The command receiver sends reply messages. It is responsible for interpreting and executing command messages. Each command message requires one reply message.

Figure 2-5 shows the software logic for implementing reply message transaction. The command receiver first sends a command to request the reply message and enable the timer. Upon receipt of a message, the integrity of the message is verified by checking the cyclic redundancy check (CRC). If the CRC checking is failed, the command receiver transmits out another request. If the CRC checking is passed, the command receiver proceeds to generate the response and wait for ACK. If an ACK is received, the command receiver routes reply into the database. Otherwise, it proceeds to wait for the next request.

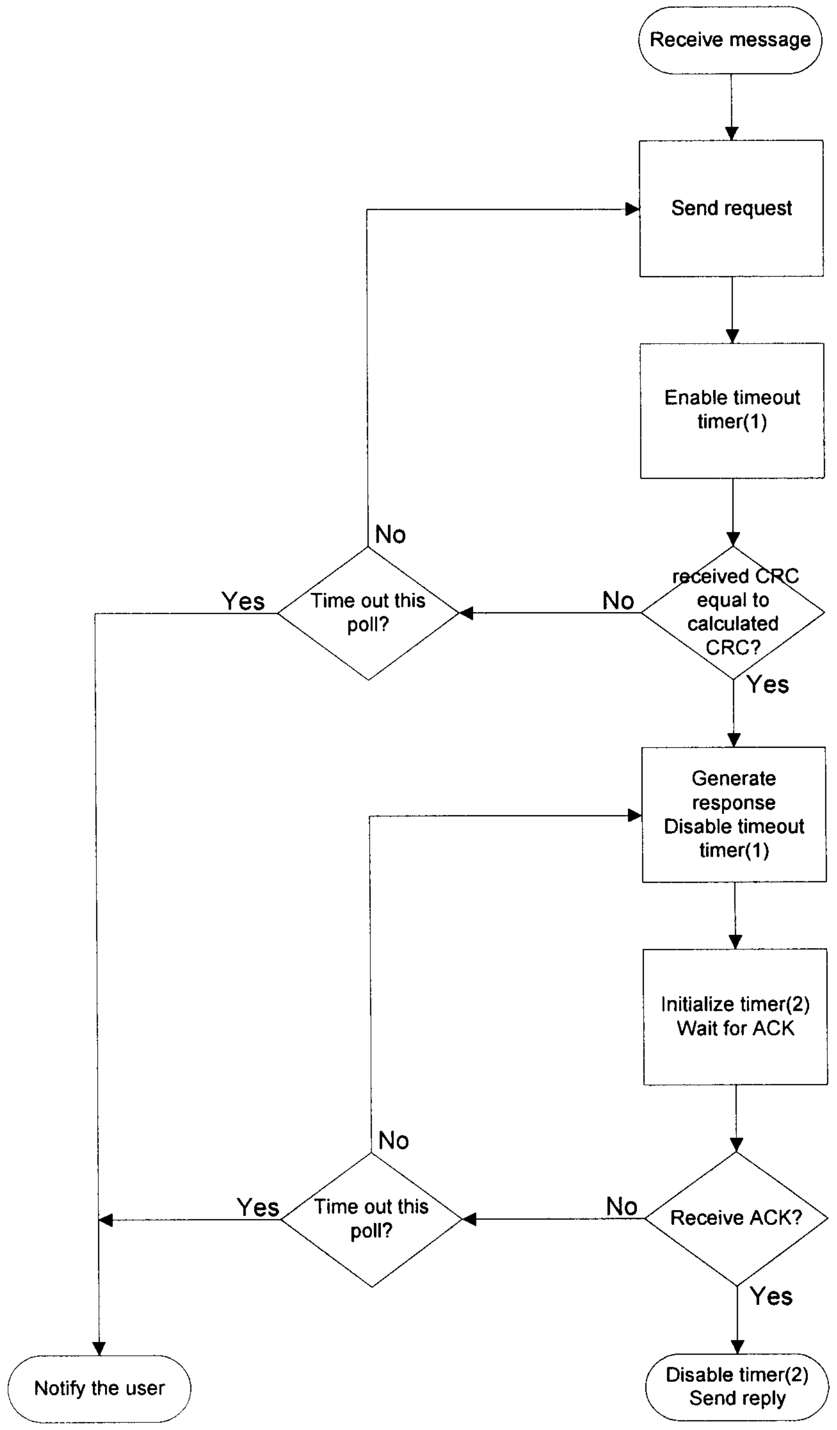

Receive message
Send request
Enable timeout timer(1)
received CRC equal to calculated CRC?
No
Yes
Time out this poll?
No
Yes
Generate response Disable timeout timer(1)
Initialize timer(2) Wait for ACK
Receive ACK?
No
Yes
Time out this poll?
No
Yes
Notify the user
Disable timer(2) Send reply

Figure 2-5. Flowchart for implementing reply message transaction

The command message and reply message provide extra data integrity by making sure that a required action always returns a proper reply. The application-layer protocol distinguishes a command from a reply. The unprotected write command and the unprotected read command (Allen Bradley, 1996) permit access to the PLC's data memory. The address fields of unprotected reads and unprotected writes are used as word addresses in MicroLogix Family.

Unprotected write command.

Unprotected write command writes data to a common interface file (CIF). This command is implemented as a protected file read by MicroLogix processors and is used by non- MicroLogix devices to read information from MicroLogix devices. The message packet is constructed as follows:

Write Command Message:

DST	SRC	CMD (8)	STS	TNS1	TNS2	ADDL	ADDH	DATA (max. 234 bytes)

Reply Message:

CMD (48)	STS	TNS1	TNS2

Where

DST is the destination node, containing receiving message.

SRC is the source node, containing sending message.

CMD is the command byte. For command message, CMD = 8 Hex; for reply message, CMD = 48 Hex.

STS indicates the status of the message transaction, which typically equals to zero.

TNS1 and TNS2 are bytes that contain a unique 16-bit transaction value. The protocol requires that each command have a unique transaction value that is different from the previously issued command. In order to generate this number, the developer can simply set up a 16-bit counter, increment the counter each time the command sender sends message, and store the value in the two bytes of TNS. TNS1 is the low byte of the 16-bit value; TNS2 is the high byte of the 16-bit value.

ADDL and ADDH are bytes that contain a 2-bytes address of memory location since the PLC register is 2-bytes wide. ADDL is the low order address byte. ADDH is the high order address byte.

DATA contains data values being transmitted by the message. The size is implied by the number of data bytes sent. For MicroLogix processors, the valid range is 0-234 bytes. The following example describes how to write a 2357 Hex into the contents of register N7:0. N7 is the PLC memory location which is 16 bits, or 2 bytes wide.

Write Command Message:

CMD	STS	TNS1	TNS2	ADDL	ADDH	DATA (max. 234 bytes)	
8	0	23	34	0	0	57	23

Reply Message:

CMD	STS	TNS1	TNS2
48	0	23	34

Unprotected read command.

Unprotected read command reads data from a common interface file (CIF). This command is implemented as a protected file read in MicroLogix processors and is used by non- MicroLogix devices to read information from MicroLogix devices. The message packet is constructed as follows:

Read Command Message:

DST	SRC	CMD (1)	STS	TNS1	TNS2	ADDL	ADDH	SIZE

Reply Message:

CMD (41)	STS	TNS1	TNS2	DATA (max. 234 bytes)

In contrast of write command message, read command message specifies the size field, not the data field at the end of the packet. The SIZE byte specifies the number of data bytes to be transferred by a message. This field should be an even value because PLC words are 2-bytes wide. The low order byte is always transferred first. In addition, for read command message, CMD = 1 Hex; for reply message, CMD = 41 Hex. The following example illustrates how to write two data bytes (2211) starting at the contents of register N7:1.

Read Command Message:

CMD	STS	TNS1	TNS2	ADDL	ADDH	SIZE
1	0	34	56	2	0	4

Reply Message:

CMD	STS	TNS1	TNS2	DATA	
41	0	34	56	11	22

2.6 Network Communication

2.6.1 Data network and control network

Generally speaking, computer networks can be divided into two categories, namely data networks and control networks (Raji, 1994). Data networks use large data packets and relatively infrequent transmission over a wide area with high data rates to support the transmission of large data files (Lian, et al., 2000a). Control networks, on the other hand, must shuttle countless small, but frequent packets among a relatively large set of nodes to meet the time-critical requirements (Lian, et al.). The key element that distinguishes control networks from data networks is the capability to support time-critical applications. Data networks do not have hard time-critical constraints (Raji). Hence, the communication of control systems should be implemented only through control networks.

A wide variety of networks are commercially available. Every type of network has particular strengths and weaknesses, depending upon their basic designs. Three types of control networks commonly implemented in industrial control systems are Token-passing, CAN-based, and Ethernet-based network.

2.6.2 Token-passing network

Token-passing network was originally developed by IBM. PROFIBUS, ControlNet, and MAP are typical examples of token-passing networks. In a Token-passing network, individual hosts are connected in a ring, as shown in Figure 2-6. A small data frame, called a token, circulates around the ring. During operation of the network, the station with the token is granted the right to transmit data. If a node runs out of data frames to transmit, or has no message to send, it passes the token to the successor on the network. Each station can hold the token for a certain period of time, depending on the token rotation time that has been implemented. The physical location of the next station on the ring is not important, because the token is sent to the logical neighbor. In this manner, only one station may have control on the network at a specific point of time. Other stations on the ring cannot transmit data at this moment. They must wait for the token to become available.

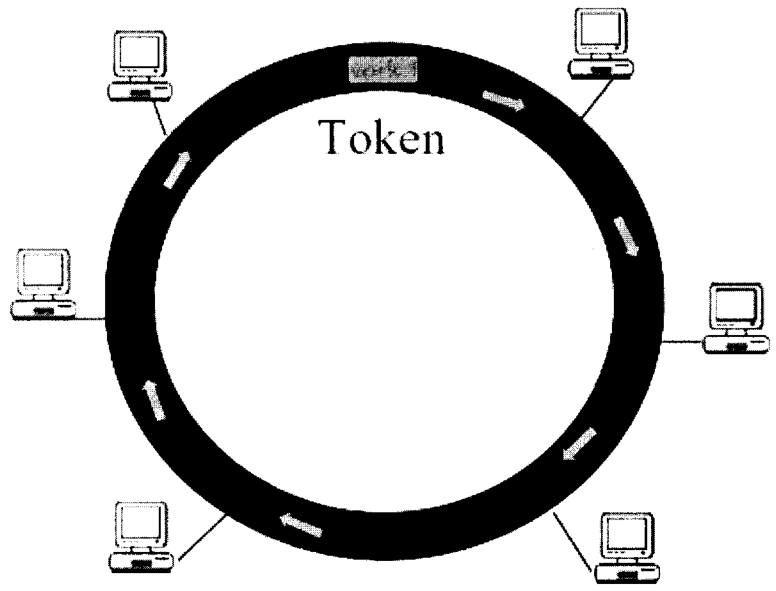


Figure 2-6. A token-passing network

<u>Priority system.</u>

Token-passing networks have a sophisticated priority system that permits certain stations with high priority to use the network more frequently. Priority is defined by the frame's priority and reservation fields. Only stations with a priority equal or higher than the priority field of the token can seize that token. After the token is captured and changed to an information frame, only stations with a priority higher than that of the transmitting station can reserve the token for the next pass around the network. When the next token is generated, it includes the higher reserving station. Stations must reinstate the priority back to its previous value after their transmission has been completed.

Management mechanisms.

In order to detect and correct network faults, Token Ring networks may dedicate a station for monitoring frames which are circling around without being dealt with. This monitor removes such frames and allows the network to function in a normal manner all over again.

Advantages and disadvantages.

Token-passing networks are deterministic. In other words, one can calculate the maximum time that will pass before any end station will be capable of transmitting (Cisco System, Inc., 2001). This feature provides excellent throughput and efficiency at high network loads, which makes Token-passing networks ideal for applications in which any delay must be predictable and robust network operation is important.

Token-passing systems, however, have some disadvantages. Token passing protocols are much more complex than contention-based protocols. It may require that a node with an urgent or critical message wait to receive the token. Furthermore, when there are a large number of nodes in one logical ring, a large percentage of the bandwidth has to be used in passing the token between nodes (Koubias & Papadopoulos, 1995).

2.6.3 CAN-based network

CAN (Controller Area Network) protocol was originally developed in 1983 by a German company Robert Bosch for use in automotive industries (Yodyium, 2003).

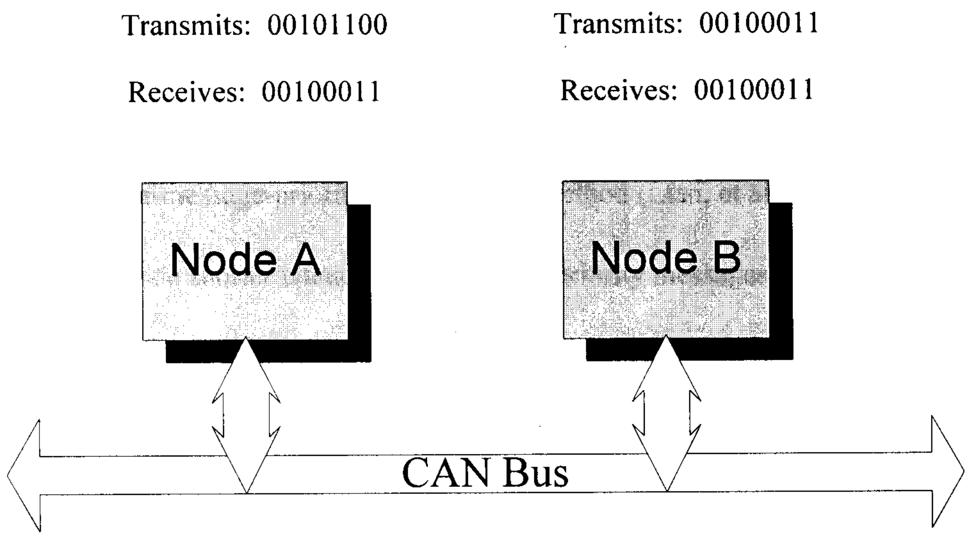
It is also being used increasingly in industrial automation technology and in a large number of embedded systems in various fields of applications - from coffee machines to kidney stone lithotripters. CANopen and DeviceNet are typical examples of CAN-based networks (Zeltwanger, 2000). CANopen is the dominant standard for applications in embedded networks whereas DeviceNet is used especially in industrial automation in the environment by Rockwell Automation. CAN-based networks have a series of special features compared with other system solutions.

Access and collisions.

Each node of a CAN network can initiate the transmission of a message as soon as the bus is free and then starts to send the identifier of its message bit by bit. As it may happen that more than one network node begins with the transmission of a message at the same time, an arbitration process is necessary, which ensures only one node actually continues with the transmission of its message.

Figure 2-7 shows the principle used for the bus arbitration. When the nodes start to transmit their respective messages, each bit of the identifier is written onto the bus and also read back by each node. If two nodes want to send message at the same time, they first continue to send the message frame and then listen to the network. If one of them receives a bit different from the one it sends out, it will stop transmitting its message and the other wins the arbitration. This arbitration is called

Carrier Sense Multiple Access with Collision Resolution (CSMA/CR). With this method, the process ensures "lossless" bus access.



Node B Wins Arbitration

Figure 2-7. A principle used for the bus arbitration

Multi-cast message transmission.

The CAN-based network was designed as a multi-master architecture with a maximum transfer rate of 1Mbit/sec, that is, all CAN nodes are able to transmit data and several CAN nodes can request the bus simultaneously. Unlike other types of networks, transmitted data in a CAN message does not necessarily contain addresses of either the source or the destination of the message. It is the data not the node that is given an identifier unique through the network. The message is broadcast to the network where other nodes pick up or reject message depending upon the configuration of mask filters for the identifier. For example, in a car, one node may be transmitting signals of wheel speed, which may be picked up simultaneously by the ABS unit, the instrument cluster and the engine management system. But none of these data receivers has knowledge of where the information came from.

Advantages and disadvantages.

The CAN-based network is optimized for short messages and utilizes an arbitration-on-message-priority media access method (Lian, et al., 1999). These features provide shorter latency times for high-priority messages and the ability to transmit even in environments with high fault-intensity.

One of major disadvantages of CAN, compared with the other networks, is the slow data rate (Lian, et al., 1999). The maximum data rate in DeviceNet is 500 Kbps. This also limits the maximum bus length (network extension) because the protocol requires different stations to be synchronized within a bit time. Moreover, a maximum of 8 bytes of data can be transmitted with one CAN message. Hence, the CAN network is not suitable for transmission of message of large data sizes, although larger data blocks can be transmitted by a series of consecutive CAN messages.

2.6.4 Ethernet-based network.

Ethernet is the most widely used local area network (LAN) technology. It was designed to fill the middle ground between long-distance, low-speed networks and specialized, computer-room networks carrying data at high speed for very limited distances (Cisco System, Inc., 2001). Because of its availability, high communication rates, and well-established infrastructure, Ethernet has become a primary network control candidate for control applications.

The first experimental Ethernet system was developed in the early 1970s by Bob Metcalfe and David Boggs of the Xerox Palo Alto Research Center (PARC). This was used as the basis for the Institute of Electrical and Electronic Engineers (IEEE) 802.3 specification released in 1980.

Access and collisions

Ethernet devices compete for access to the network using Carrier Sense Multiple Access with Collision Detection (CSMA/CD) mechanism for resolving contention on the communication medium. CSMA/CD is an access method that allows only one station to transmit at a time on a shared medium. Standard Ethernet using CSMA/CD takes into consideration all the transmission requests and determines what devices can transmit and when they can transmit for all devices to receive adequate service (Cisco Systems, Inc., 2001).

Figure 2-8 shows a typical process of CSMA/CD. Similar to the CSMA/CR used in the CAN-based network, when a node at Ethernet networks wants to initiate the transmission of a message, it first listens to the network to see if there are any signals on the networking media. After the node determines that the network is idle, it can begin to transmit data. If two network nodes begin with the transmission of a message simultaneously, a collision detection method will be applied. With CSMA/CR mechanism used in the CAN-based network, the high priority node will continue sending messages while the low priority node will stop initiating transmission and start listening to the network. With CSMA/CD mechanism used in Ethernet, in contrast, both nodes stop sending data and retry their transmission after a randomly chosen delay period. This random time is determined by the standard Binary Exponential Backoff (BEB) algorithm (Lian,, et al., 1999).

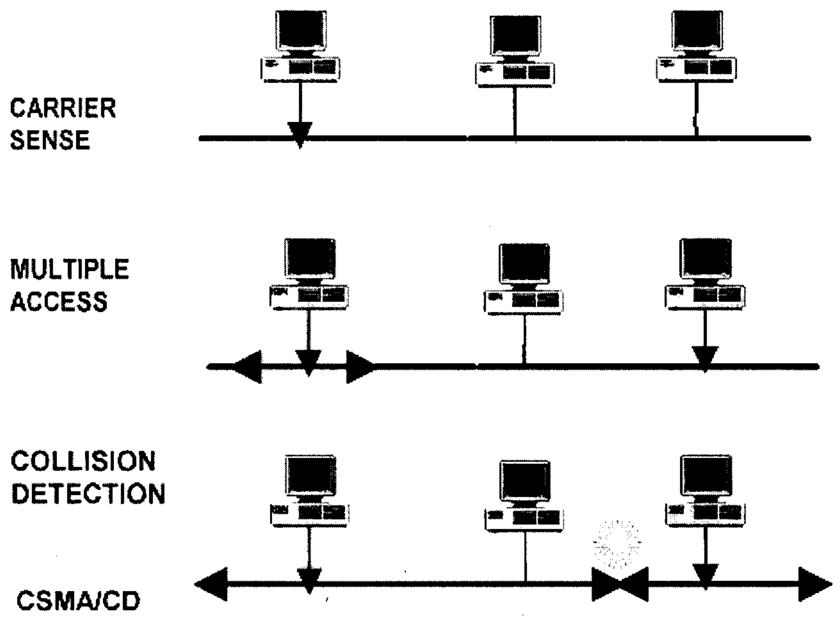


Figure 2-8. The process of CSMA/CD mechanism

Advantages and disadvantages.

There are many advantages of Ethernet with the most compelling advantages as its availability. Almost all personal computers sold today are equipped with Ethernet connectivity. This fact translates into low cost and high availability. Ethernet components are commodity items. Network cable, interface cards, network analysis tools, and management software offer significant lower cost compared with components used in other type of networks. Ethernet is well-established in office networks. Thus, extending it to the factory floor will be relatively easier than other types of network.

Another major advantage of Ethernet is its flexibility. Table 2-2 describes some critical characteristics of a standard Ethernet. It is able to transmit messages from a minimum size of 72 bytes to a maximum of 1500 bytes over a length of 2500 meters (Otanez, Parrott, Moyne, & Tilbury, 2000).

Table 2-2

Characteristics of Standard Ethernet

Configuration Parameters	Standard Ethernet
Data Rate (Mbps)	10
Bit Time(μs)	0.1
Max Length (meter)	2500
Max Data Size (byte)	1500
Min Message Size (byte)	72
Typical Tx Speed (m/s)	coaxial cable: 10

Because of low media access overhead, Ethernet uses a simple algorithm for operation of the network and has almost no delay at low network loads (Lian, Moyne, & Tilbury, 2000b). In addition, the multi-drop nature of Ethernet provides a straightforward expansion path on networks.

The major disadvantage of standard Ethernet with respect to control system networking is that it can not guarantee the delivery of time critical information because of its nondeterministic nature of communication. Network protocols employ CSMA/CD mechanisms to prevent message collisions from happening. When a collision occurs, the transmitting nodes wait a random length of time to retry transmission. This random time is determined by the standard BEB algorithm. Based on BEB algorithm, if sixteen collisions are detected, the transmitting node discards the message and reports an error. Therefore, there is no guarantee that a particular critical message will not collide and eventually be dropped.

Message collisions significantly compromise network performance by increasing time delays or causing message loss (Otanez, et al., 2000). These time delays come from the time sharing of the communication medium as well as the computation time required for physical signal coding and communication processing. The characteristics of time delays could be constant, bounded, or even random, depending on the network protocols adopted and the chosen hardware (Lian, 2001). For networks with low traffic and high data rates, time delays are relatively small and under some conditions can be neglected when the controller is designed for most networks (Walsh & Ye, 2001). However, the uncertainty in the magnitude of network and device delays hinders the performance of NCSs and has raised questions concerning what quality of control performance can be expected of distributed networked control systems (Otanez, 2002).

2.7 Overview of Networked Control System

2.7.1 Traditional communication architecture and its limitations

For many years, point-to-point architecture is the traditional communication architecture widely operated in industrial and manufacturing control systems. This centralized communication architecture had a single central control unit. Each system controller is directly wired to other devices, such as sensors, or actuators.

Complex control systems, however, cover large plant areas and connect a large number of components by means of computationally intense algorithms. Hence, a large amount of direct electrical wiring is required to connect system components like sensors, actuators, and controllers together. That usually results in a huge project, especially when many subsystems are not closely located.

From this point of view, it is unrealistic to implement the traditional point-to-point architecture in these complex systems. In the point-to-point architecture, there is a lack of common communication protocol and component interchangeability (Lian, 2001). Moreover, it is difficult to add, delete or interchange components in complex systems, which may make it difficult to expand physical setups and system functionality.

2.7.2 Introduction to networked control system

As an alternative to point-to-point, a Networked Control System (NCS) is one type of distributed common-bus control system where the control loop is operated over a communication network. Figure 2-9 illustrates a typical setup and information flow of a NCS. The defining feature of an NCS is its control components (i.e., sensors, actuators, and controllers) are distributed and interconnected by communication networks.

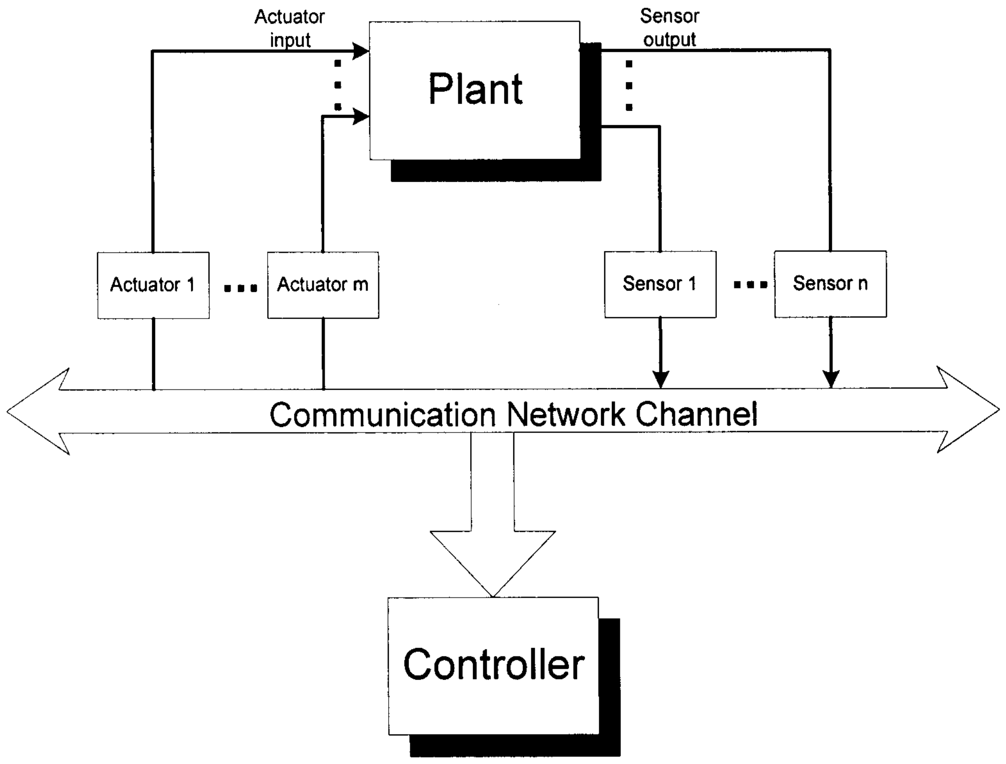


Figure 2-9. A typical setup and information flow of a NCS

In contrast to point-to-point communication, NCS can improve efficiency, offers better resource utilization, and also reduces installation, maintenance time and costs. Moreover, this type of architecture supplies higher intelligence at nodes for modularization of functionality and standard interfaces for interchangeability and interoperability (Koren, 1999). Intelligence and decision making can be moved out of the central control units and distributed into controllers located near the controlled devices (Lian, et al., 2000). Hence, the processing load on single central control unit can be assigned into several small processors.

Traditionally, communication networks used in NCS applications are specific industrial networks. Controller Area Network (CAN) protocol, for example, was one type of industrial network protocol, which was originally developed in 1983 for automobile industry. Profibus is a broadcast bus protocol that operates as a multi-master/slave system, developed by six German companies and five German institutes in 1987. Many other industrial network protocols including Foundation, Fieldbus and DeviceNet were also developed about the same period (Yodyium, 2003). Meanwhile, technologies on general computer networks especially Ethernet have progressed very rapidly. With the decreasing price, increasing speed, widespread usages, numerous software and applications, and well-established infrastructure, Ethernet networks have become major competitors to the industrial networks for control applications (Kaplan, 2001).

Depending on the network protocol, a network system may also provide attractive features with respect to different communication models supported. These include client/server, master/slave, and publisher/subscriber (Lian, 2001). Because all devices are interconnected by a common-bus network, information generated by a single unit is easily shared by other devices.

Regardless of the type of network used, however, implementing control applications over a communication network may cause a number of problems, which can potentially degrade control system effectiveness and possibly cause system instability. One challenging problem is network delay. The time to read a sensor measurement and to send a control signal to an actuator through the network depends on network characteristics (i.e., topologies, routing schemes, etc). These time delays come from the time sharing of the communication medium as well as additional functionality required for physical signal coding and communication processing (Lian, et. al., 2001). These inevitable time delays, if not dealt with properly, can greatly influence system performance and even cause system instability.

The study of NCS combines both network and control theory. Lian (2001) represented a NCS research methodology model as shown in Figure 2-10. In this model, control application is utilizing networks for communication and computers for processing. Thus, three major issues should be considered in the study of any NCS, including: network systems, networked devices and control systems. In a summary, "the overall NCS study should first research the characteristics of the network application systems and networked devices, and then parameterize all key factors that impact the stability and performance of both network and control systems" (Lian, 2001, p.19).

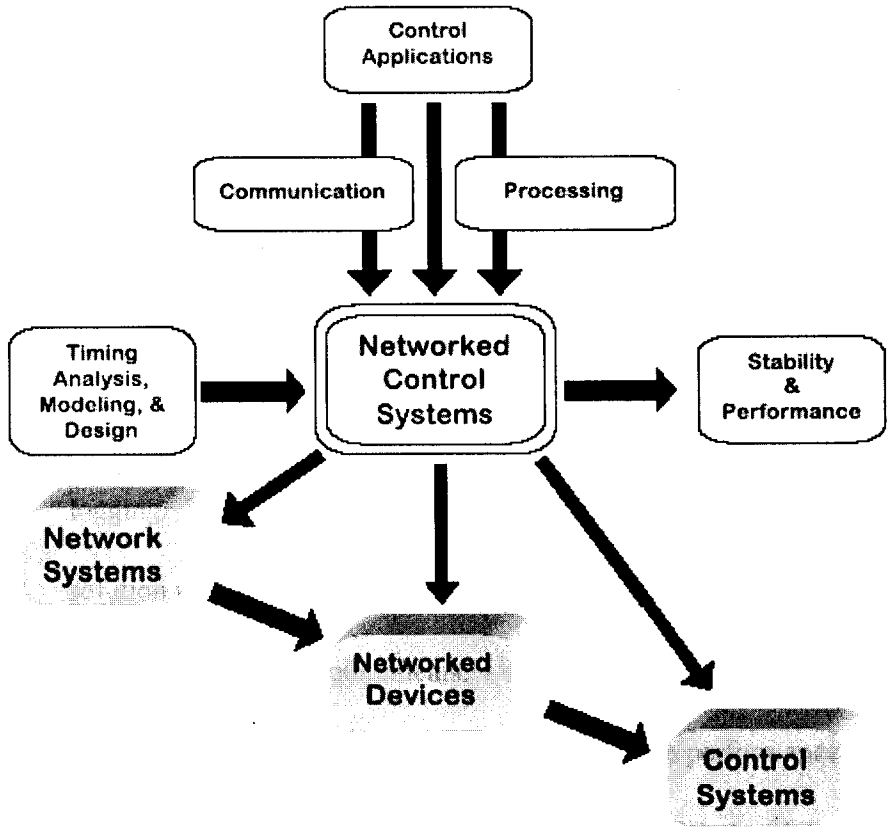


Figure 2-10. NCS research methodology (Lian, 2001, p. 4)

2.8 Software Tools Implemented in the Designed NCS

2.8.1. Visual Basic for HMI applications

Visual Basic (VB) is an application development tool designed specially for the Microsoft Windows operating systems. The language is designed for a broader audience with less formal programming experience and training than lower level languages such as C or C++. VB provides a powerful and flexible environment where the developer is able to write diverse programs for a wide range of application with minimal effort.

The primary goal of a HMI is to assist the operator in running a machine and managing a process. A HMI will increase the productivity of the operator and machine, increase uptime, and assist in providing consistent product quality. With many off-the-shelf HMI software packages available, Visual Basic can be used effectively for creating HMI applications for the following reasons.

1. Many HMI applications only require a few user screens and simple data logging. In the case of these applications, VB provides the optimum solution with additional flexibility and choice that may be required to meet further operational requirements. Until recently, the use of Visual Basic for industrial HMI applications was limited by the performance of interpreted code at runtime and the lack of tools for industrial applications such as control system hardware connectivity and data visualization (Weber, 1999). However, new technologies based on ActiveX have removed these barriers by extending the suite of tools and objects available to VB developers. With these barriers gone, VB can be used to quickly create simple HMI applications.
2. From the standpoint of maintenance and budget, VB is the clear winner over proprietary industrial automation software. By creating applications in VB, developers can distribute their compiled application royalty free. No per-machine runtime licenses are required for the actual VB code or the built-in objects (text boxes, command buttons, etc). Although developers

do use some third party plug-ins, those controls have either only require nominal per machine fees or no distribution licensing fees at all.

3. VB is one of the most widely used development environment in the world. More than three million programmers worldwide know Visual Basic language. Developers are able to easily find someone with the aptitude and skills to understand the application source code and minimize the learning curve.

2.8.2. ActiveX Control for communication

An ActiveX control is a component program object that can be added to Visual Basic. The main technology is based on the Component Object Model (COM). ActiveX controls can be used for any common task by an application program in the latest Windows and Macintosh environments. In implementation, an ActiveX control is a dynamic link library (DLL) module.

A number of ActiveX controls have emerged in the general marketplace for the Visual Basic developer to handle connectivity. Two different ActiveX controls will be introduced in this section. The first is Winsock control provided by Microsoft, and the other is known as Serial DF1 ActiveX Control provided by CimQuest Inc.

<u>MS Winsock for network communication.</u>

The Winsock control shipped with Visual Basic acts as the middleware between Windows applications (such as ftp, a Web browser, Telnet, and so forth) and the Internet protocol. It is the lowest level network programming protocol, and defines a network programming interface for Microsoft Windows. The Winsock control allows the user to handle the network protocol issues, and background processing without having to worry about the details of TCP/UDP standard or to call low level of application programming interface (API).

The latest version of Winsock is Winsock 2. The Windows Open System Architecture (WOSA) compliant WinSock 2 architecture is illustrated in Figure 2-11. It defines two interfaces: an API which shields application developers from underlying layers, and a service provider interface (SPI) which allows transparent extensions to a Winsock stack. With this architecture, it is no longer necessary for each stack vendor to supply their own implementation of WinSock 2 DLL since a single WinSock 2 DLL must work across all stacks. The WinSock 2 DLL should thus be viewed in the same light as an operating system component (International Center for Theoretical Physics, 1996).

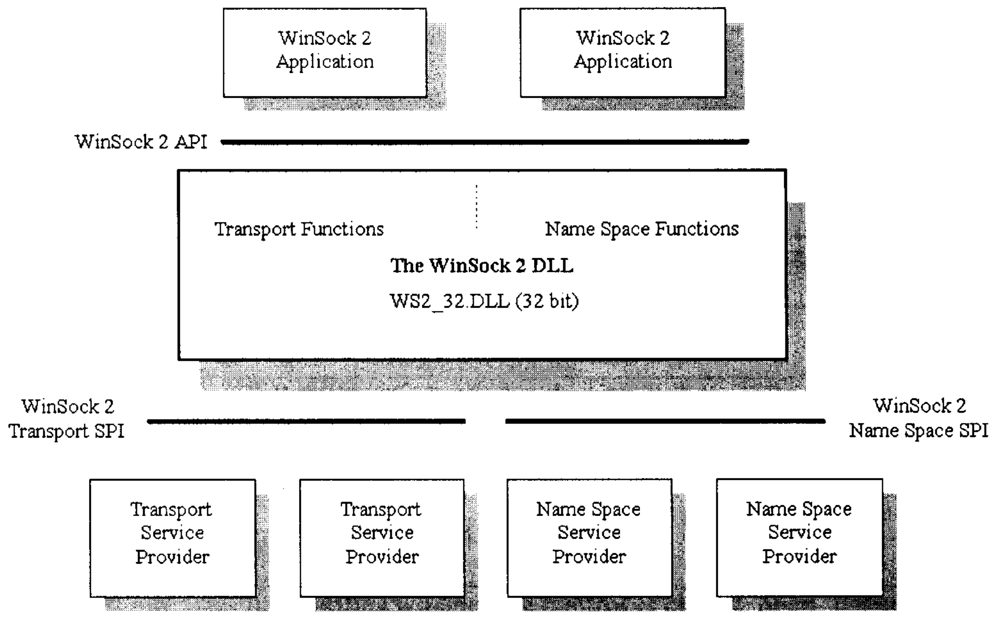


Figure 2-11. Winsock 2 architecture (ICTP, 1996, ¶3)

By setting properties and invoking control methods, a remote machine can be connected. It is possible to exchange data in both directions over various network transport protocols and client-server implementations.

Some important Winsock control properties are described in Table 2-3. Those properties make it possible to set the communication protocol, the IP address of the remote unit, remote and local port to use, and the current state of control when establishing a communication link. For example, "RemoteHost" property can be set as a server IP address "139.67.139.21".

Table 2-3

Important Winsock Control Properties

Winsock Property	Description
RemoteHost	Returns the name/IP address of the server computer.
RemotePort	Returns or sets the remote port number
LocalPort	Returns or sets the local port number
Protocol	Returns or sets the protocol, either TCP or UDP, used by the Winsock control.
State	Identifies whether the control is closed, open, listening, connected, error,etc.

Table 2-4 describes the important methods of the Winsock control. For example, “Connect” method is used to establish a connection to the server. The orders they appear in the table are typically the order in which they are used. Generally, an application will establish connection using “Connect” method before it is able to listen or accept requests. “Close” method will be used to terminate the application.

Table 2-4

Important Winsock Control Methods

Winsock Method	Description
Connect	Requests a connection to a remote server (invoked on the client application only).
Listen	Waits for a TCP/UDP request for connection from a client system (executed by the server application only).
Accept	Accepts the request for connection from the client system (used in the server application only).
SendData	Dispatches data to the remote computer (used for both the client and server application).
GetData	Retrieves data from remote commuter (used for both the client and server application).
Close	Close an open socket (used for both the client and server application).

Winsock enables developers to create a client-server application. The client program here is the one that requests a network connection, and the server program is the one that listens for a connection request.

Figure 2-12 illustrates a typical client-server model which indicates a series of events that occur during the lifetime of a client-server application. The left column refers to the client application, and the right column refers to the server application. Time proceeds from the top of the diagram to the bottom. Both client and server applications start with creating a socket and end up with the "Close" method.

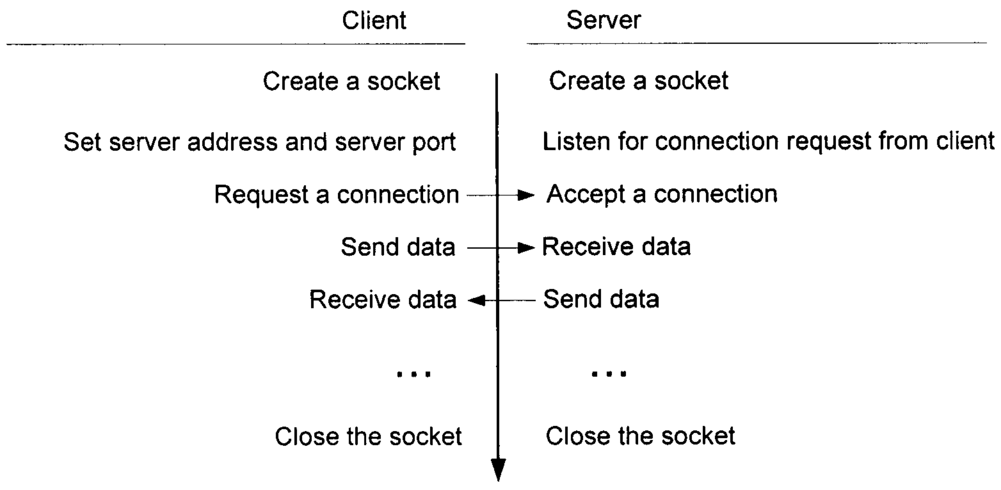


Figure 2-12. A typical client-server model

<u>DF1 ActiveX Control for communication with PLCs.</u>

The framework of the data packet transmitted between server PC and PLC is constructed with ASCII control characters through RS-232 serial port. Serial DF1 ActiveX Control provided by CimQuest Inc. can be used to handle this connectivity. The control application handles all low level protocol formatting, hardware interfacing, and error checking needed to communicate to a PLC. Similar to Winsock control, DF1 ActiveX control for VB also has properties and methods. Properties define the exact functionality the control will perform.

Some important control properties are listed and briefly described in Table 2-5. For example, "Host" property can be set as a specific IP address "139.67.139.1" for PLC.

Table 2-5

Important Serial DF1 ActiveX Control Properties

Property	Description
Host	Sets or returns the IP Address of the PLC Module
Adapter	Sets or returns the communication adapter number to be used.
Function	Sets or returns the type of data transaction the control will perform
FileAddr	Sets or returns the starting data point to read or write in the PLC
Size	Sets or returns the number of data table elements the control will read or write

Table 2-6 illustrates several critical methods that a developer invokes to make the control perform its functions. For example, "Trigger" method can be used to invoke a PLC read function. The orders they appear in the table are typically the order in which they are used.

Table 2-6

Important Serial DF1 ActiveX Control Methods

Property	Description
Trigger	Invokes the control to read or write to PLC
AutoPoll	Automatically invokes the Trigger Method at the given interval
WordVal	Used to reference the controls internal data array as 16-bit values.
ClearControl	Clears the contents of the controls internal data array

CHAPTER 3

Design Methods

In order to gain an understanding on the implementation of networked control system (NCS), a prototype was used to present the structure and hierarchical model of a typical NCS. The designed prototype combined both network and control practice, addressed the detailed procedures on design, configuration, integration and implementation of a process control via communication network.

3.1 System Configuration

The prototype was composed of three parts: (a) client , (b) server, and (c) process controller, as illustrated in Figure 3-1. The client and the server were physically located at different locations and they were linked by Ethernet.

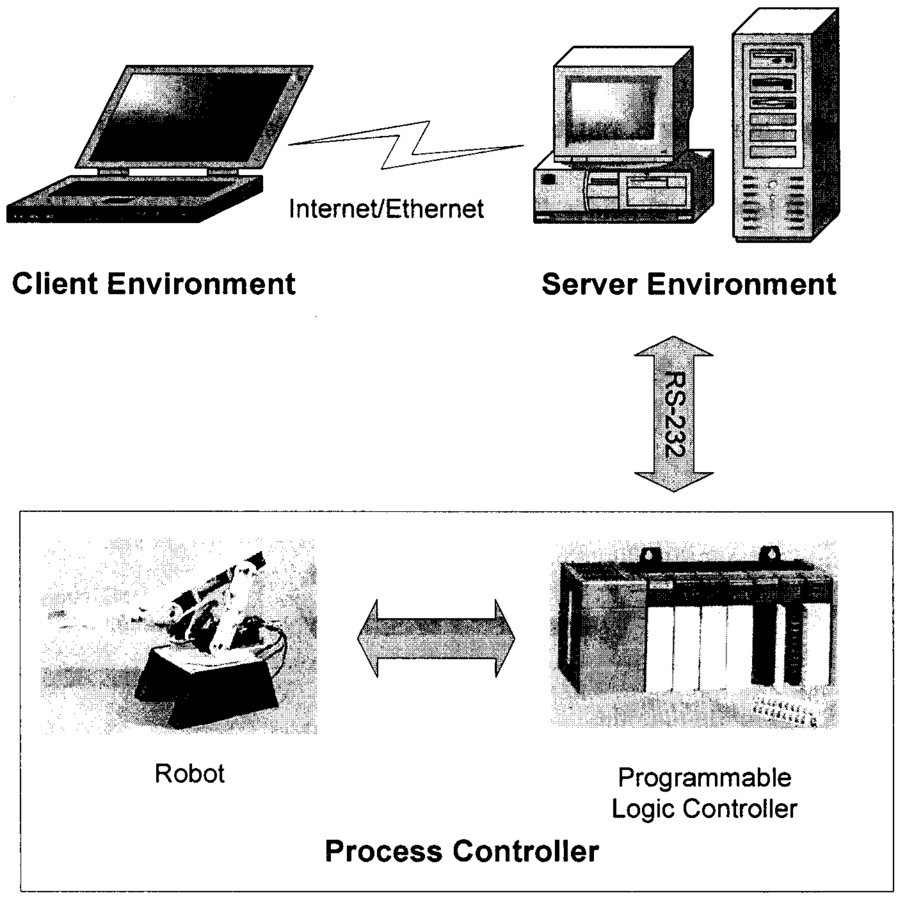


Figure 3-1. System configuration of a NCS

In this environment, the role of the client was to convey the control related commands and user messages to the server. The server was used as an interface between client and process, which processed the incoming process related data and sent them to the client. It also transmitted the user and/or client-originated commands to the process controller. A programmable logic controller (PLC) was used to communicate with the server and regulate a robot to perform desired tasks.

3.2 Software Structure

A high level functional model of software structure is shown in Figure 3-2. A user is able to regulate the process through a HMI, which is a Visual Basic-based interface. Communication between client and server was established using Microsoft Winsock control via Ethernet. The server controls the PLC via RS-232 serial port with the help of DF1 ActiveX control.

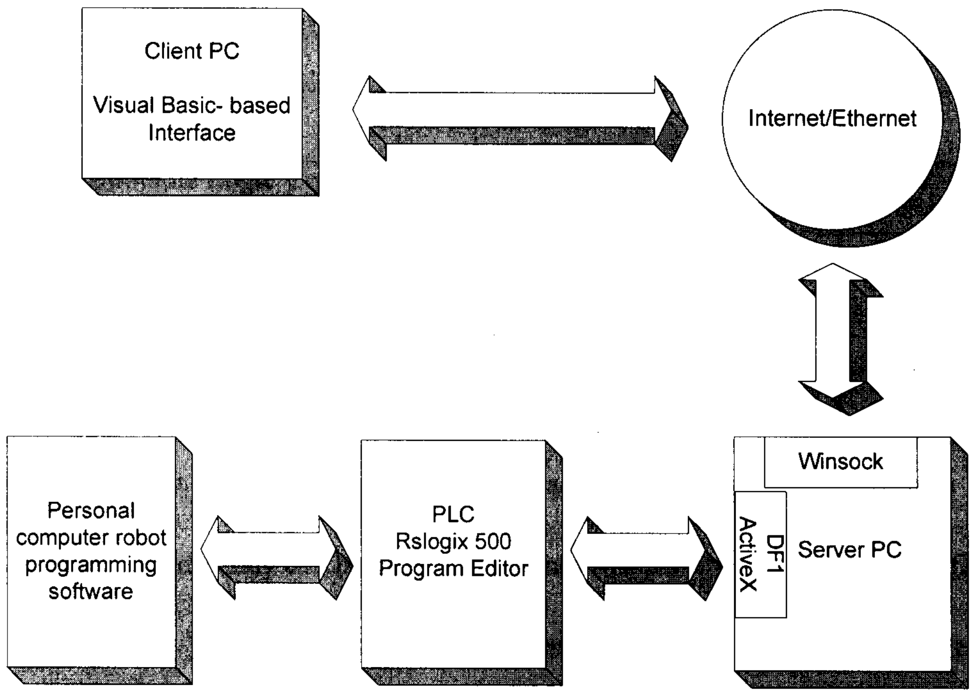

Client PC
Visual Basic- based Interface
Internet/Ethernet
Winsock
DF1 ActiveX
Server PC
Personal computer robot programming software
PLC Rslogix 500 Program Editor

Figure 3-2. Software structure of the designed NCS

3.3 Process Controller

A basic material sorting process was designed, with the hardware and software listed in Table 3-1. An Amatrol Pegasus robot driven by a PLC was used to execute the desired actions. The robot was assembled from a pre-existing kit and some modification was made in order to add sensory capability and to process the control related commands from the server.

Table 3-1

Configuration in the Process Controller

Component	Model	Quantity
Pegasus robot	880-RA2	1
Robot operator controller	88-A1	1
Parts feeder	88-F1	1
Parts bin	88-P1	2
Programmable Logic Controller	MicroLogix 1000	1
Personal Computer (PC) robot programming software	Robot programming	1
RSLogix500 program editor	PLC programming	1

The robot was programmed to perform a task of material sorting. Robot programs in this study can be broken down into two parts: program sequence file and point file. The program sequence file contains the commands used to create the robot's sequence of operation. Figure 3-3 describes the flowchart of this sorting application. The robot driven by the PLC gets an object from parts feeder, and then places the object into the inspection station to test for the presence of a hole. If a hole is detected in the object, the object is accepted and moved into the bin for accepted parts. Otherwise the object is rejected and moved into the reject bin. For each step or operation the robot performs, a separate command was used.

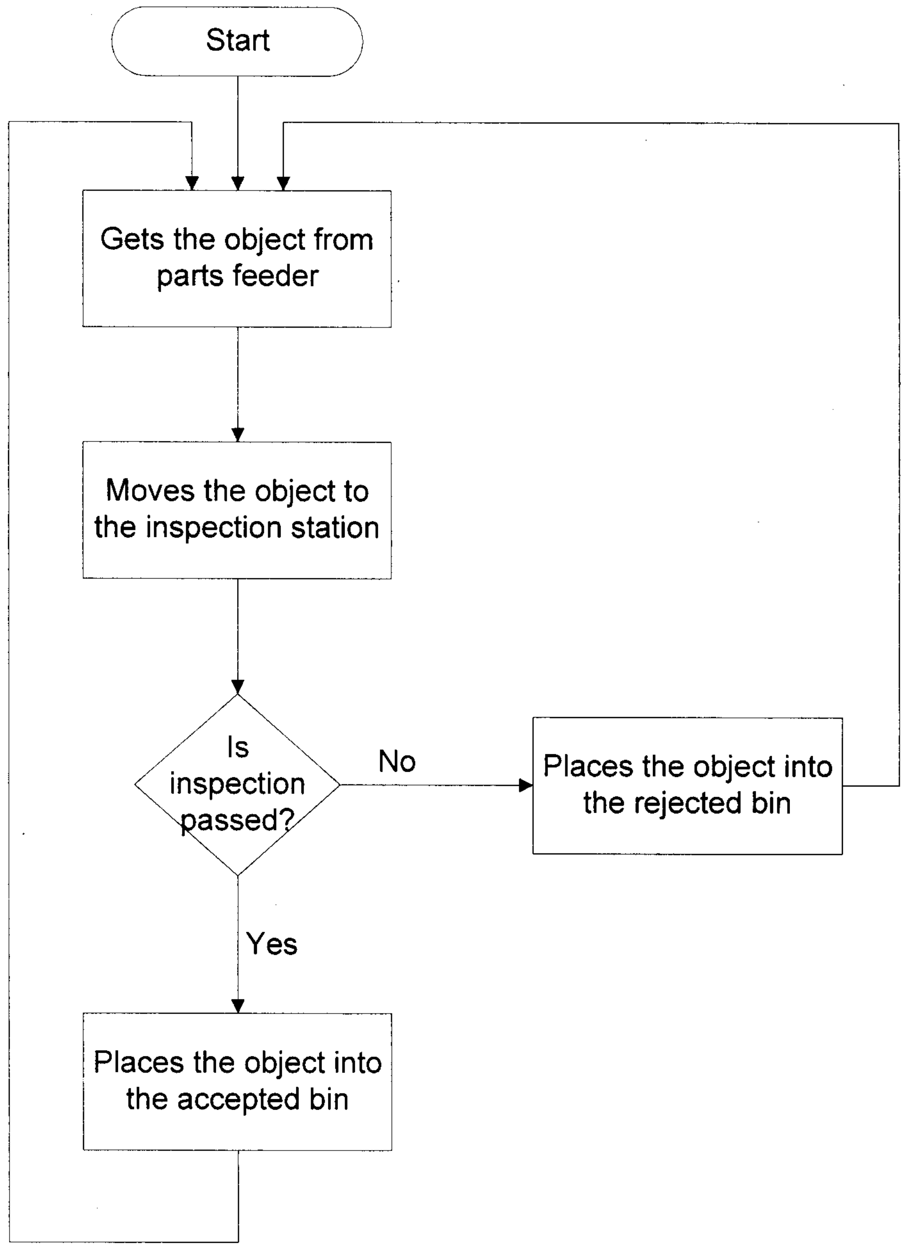

Start
Gets the object from parts feeder
Moves the object to the inspection station
Is inspection passed?
No
Places the object into the rejected bin
Yes
Places the object into the accepted bin

Figure 3-3. Flowchart for material sorting process

Before a robot program sequence can be executed, it is important to have robot program know the location of each point to which it will move. The point file stores the location of each point. Seven location points were defined in the sorting process.

An Allen-Bradley MicroLogix 1000 PLC was used to communicate with the robot. Firstly, PLC conveys the control related commands from the server and signals from the sensor to the Pegasus robot. In order to perform this application, the PLC outputs were directly wired to the terminals of the robot input. Secondly, the PLC processes incoming process related data and sends them back to the server. Robot outputs were used to transmit control feedback to the PLC. A limit switch resided in the inspection station was used to test for the presence of a hole in the object and to send the result signals to the PLC. Moreover, a built-in RS-232 port on the PLC was used for an operator interface/programming port to provide connectivity to the server computer.

3.4 Server Environment

The server computer acts as an interface between client and the controlled process. The hardware and software to be used in the server environment are listed in Table 3-2.

Table 3-2

Configuration of the Server Environment

Component	Description
CPU	Intel Pentium processor 500 MHz
RAM	128 MB
RS-232 COM port	
Ethernet adapter	3Com EtherLink XL 10/100 PCI
Operating system	Microsoft Windows 2000 Service Pack 4
Programming software	Microsoft Visual Basic 6.0

The role of the server computer is to retrieve data from process, to control process, and send response from process to the client. Therefore, communication is the major function of the server environment. The communication implemented in this study included two parts: one was the communication with the PLC, the other with the client.

3.4.1 Communication with PLC

The PLC communicates with the server computer through a RS-232 communication port, which was handled by DF1 ActiveX Control. This ActiveX control provides the ability to write control related commands to the PLC and read response data from the PLC. To perform the connectivity, the communication driver was configured by a built in utility provided by DF1 ActiveX control. Although it is applicable to perform “Write” and “Read” function in the same object, two instances of DF1 ActiveX control were used on this application for “Write” and “Read” functions respectively, which left “Read” function uninterrupted.

Basic steps of retrieving data from the PLC are as follows:

1. Create a DF1 control.
2. Set the function to read using Function property.
3. Set the starting point that the server will read from the PLC using FileAddr property.
4. Set the number of data elements the server will read using Size property.
5. Invoke Trigger method at the given interval using AutoPoll method.
6. Clear the contents of internal data array in the DF1 control using ClearControl method.

To write data to the PLC requires the same actions but in reverse. The following steps were performed:

1. Create a DF1 control.
2. Set the function to write using Function property.
3. Set the starting point that the server will write from the PLC using FileAddr property.
4. Set the number of data elements the server will write using Size property.
5. Set the data value the server will write using WorVal method.
6. Invoke control to write the PLC using Trigger method.

3.4.2 Communication with client

The communication between client and server can be established by using Microsoft Winsock control. Two separate applications were developed in this study. One application resided in the server computer and the other in client computer. Both client application and server application interacted with each other to exchange data.

Client application.

The client application requests a connection to the server. It also provides ability to send and receive data from the server. Specific steps of implementing the client application are as follows:

1. Create a Winsock control.
2. Specify server address and server port using RemoteHost and RemotePort properties, respectively.
3. Request a connection using Connect method.
4. Receive and send data using GetData and SendData methods, respectively.
5. Close the Winsock control using Close method.

Server application.

The server application listens for client and connects it to the application. Similar to the client application, the server application is capable of getting data from the client and returning the request data. Basic steps in the server application are as follows:

1. Create a Winsock control.
2. Listen for connection requests on the specified port using Listen method and LocalPort perperty.
3. Accept connection using Accept method.
4. Send and receive data from the client using SendData and GetData method, respectively.
5. Close the Winsock control using Close method.

3.5 Client Environment

The client environment was implemented as an interface that not only connects the user to the server via Internet but also allows user to control online process remotely through server. The software and hardware used in the client environment are listed in Table 3-3.

Table 3-3

Configuration of the Client Environment

Component	Description
CPU	Intel Pentium processor 500 MHz
RAM	128 MB
Ethernet adapter	3Com EtherLink XL 10/100 PCI
Operating system	Microsoft Windows 2000 Service Pack 4
Programming software	Microsoft Visual Basic 6.0

The design and implementation of the client environment was based on the algorithm shown in Figure 3-4. At the client site, a Visual Basic based human machine interface (HMI) asks for the address of the remote server to be connected. After the user verifies connection, the control program takes over to execute the remote control process.

Two control modes were designed, which can be selected by the user. A manual mode allows the user to remotely control the process step by step. The auto mode executes the entire sorting process sequence automatically. The control program is terminated when the user shuts down the connection.

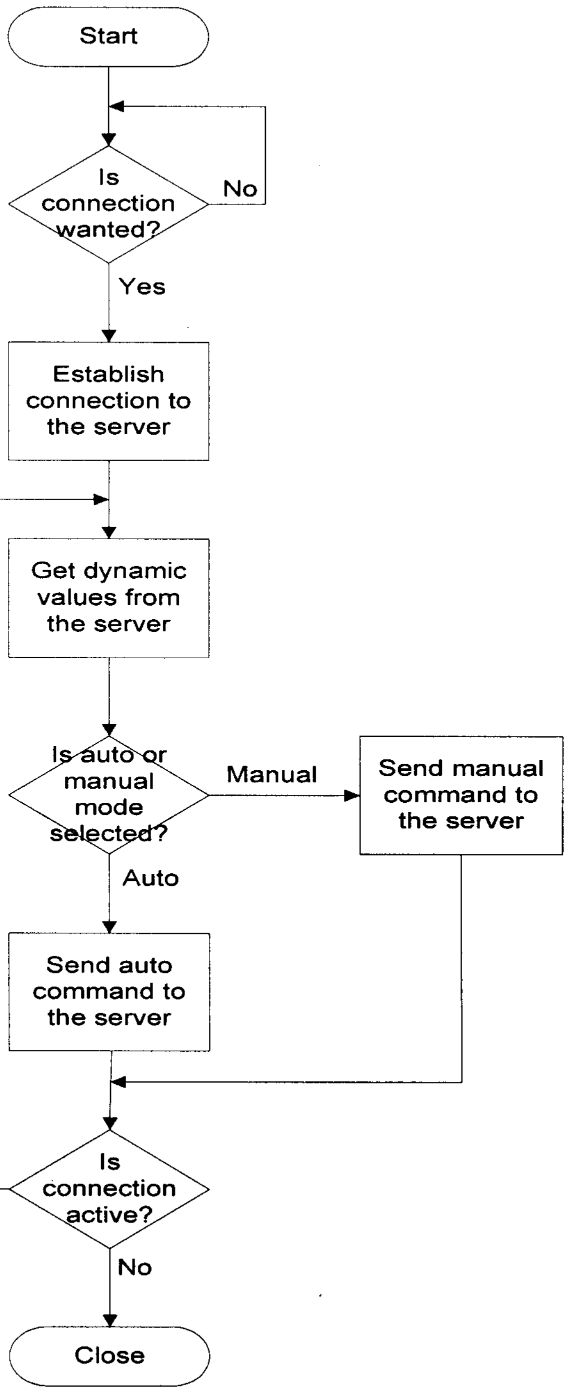

Start
Is connection wanted?
No
Yes
Establish connection to the server
Get dynamic values from the server
Is auto or manual mode selected?
Manual
Send manual command to the server
Auto
Send auto command to the server
Is connection active?
Yes
No
Close

Figure 3-4. Client environment algorithm

CHAPTER 4

Implementation

This chapter presents details of a typical Networked Control System designed for this research. The prototype can be technically decomposed into three parts: (a) process controller, (b) server, and (c) client.

4.1 Process Controller

To perform the designed material sorting process, a Pegasus robot was used for this study, which is shown in Figure 4-1. The Pegasus robot was assembled from a kit and some modification was made in order to add sensory capability and process the control related commands from the server.

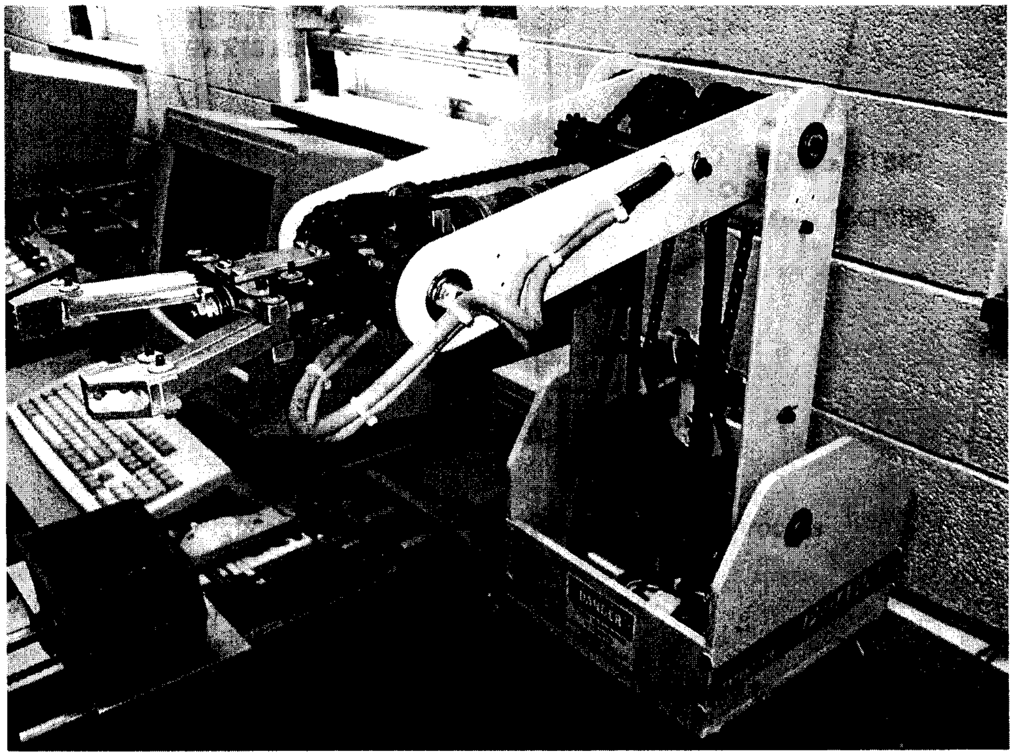


Figure 4-1. Pegasus robot

The detailed procedure implemented in the process controller is as follows:

4.1.1 Process schematic

Figure 4-2 illustrates a schematic for the material sorting process. It provides an overview that shows the robot, equipments it interacted with, and parts to be handled.

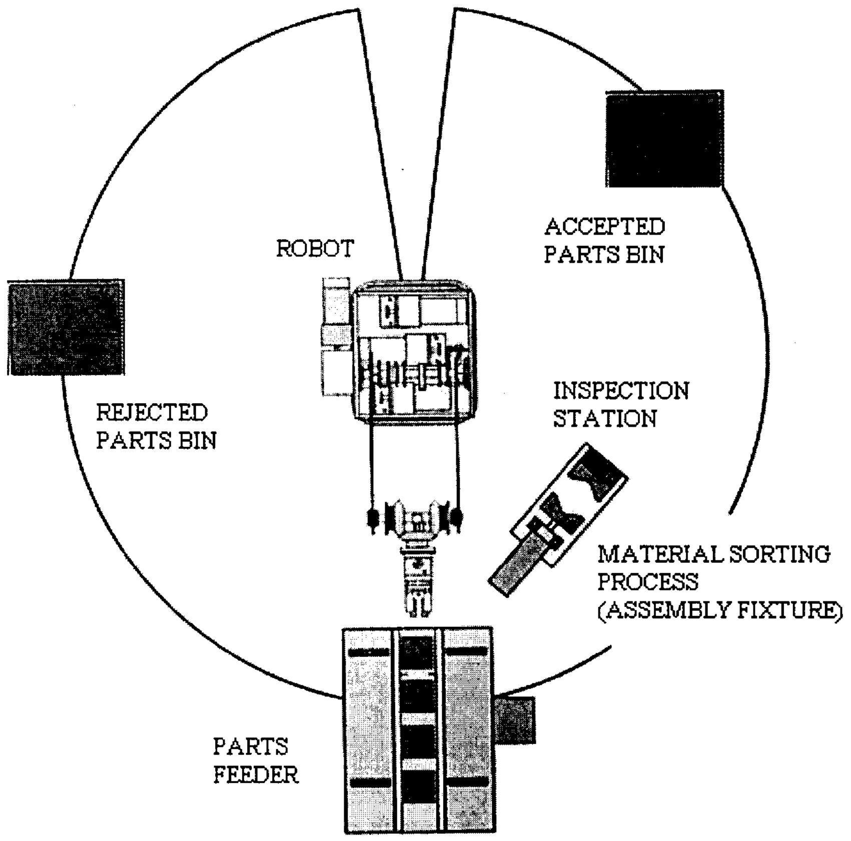


Figure 4-2. Schematic for a material sorting process

4.1.2 General sequence of operations

The program sequence file contains the commands used to create the robot's sequence of operation. Table 4-1 lists the series of steps to perform the designed material sorting process. The robot picks up, moves, and drops off the objects according to commands sent from the PLC. A separate command was used for each step or operation the robot performs.

Table 4-1

General Sequence of Operation for Material Sorting Process

Step	Description
1	Robot waits for command to start operation
2	Robot picks up the block from part feeder.
3	Robot places the block into the inspection station to test for the presence of a hole
4a	If the block has a hole, it passes the inspection and is placed into the bin of accepted parts.
4b	If the block has no hole, it fails the inspection and is placed into rejected parts bin
5	The cycle repeats as long as the stop command is not triggered.

4.1.3 List of points

The point file stores the location of each point by storing each encoder value in memory. The points used in this material sorting process are listed in Table 4-2. For example, “Point 1” indicates the robot is at “wait position”.

Table 4-2

Point List for Robot Points File

Point	Description
1	Robot wait position
2	Feeder approach/avoidance point
3	Feeder pickup point
4	Inspection station approach/avoidance point
5	Inspection station pickup/dropoff point
6	Accepted parts bin dropoff point
7	Rejected parts bin dropoff point

4.1.4 List of input/output terminals

An Allen-Bradley MicroLogix 1000 PLC was used for this research, as shown in Figure 4-3. This MicroLogix 1000 has ten discrete 24V DC inputs, six relay outputs.

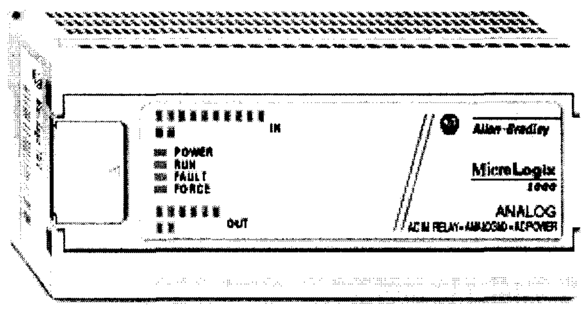


Figure 4-3. Allen-Bradley MicroLogix 1000 PLC

The PLC was used to control the overall operation, which interfaced with the robot, as shown in Figure 4-4. The Pegasus robot uses relay contact type outputs. This allows the PLC inputs to be wired directly. Moreover, the PLC outputs can be directly interfaced to the terminals of the robot inputs because PLC's outputs are relay contacts.

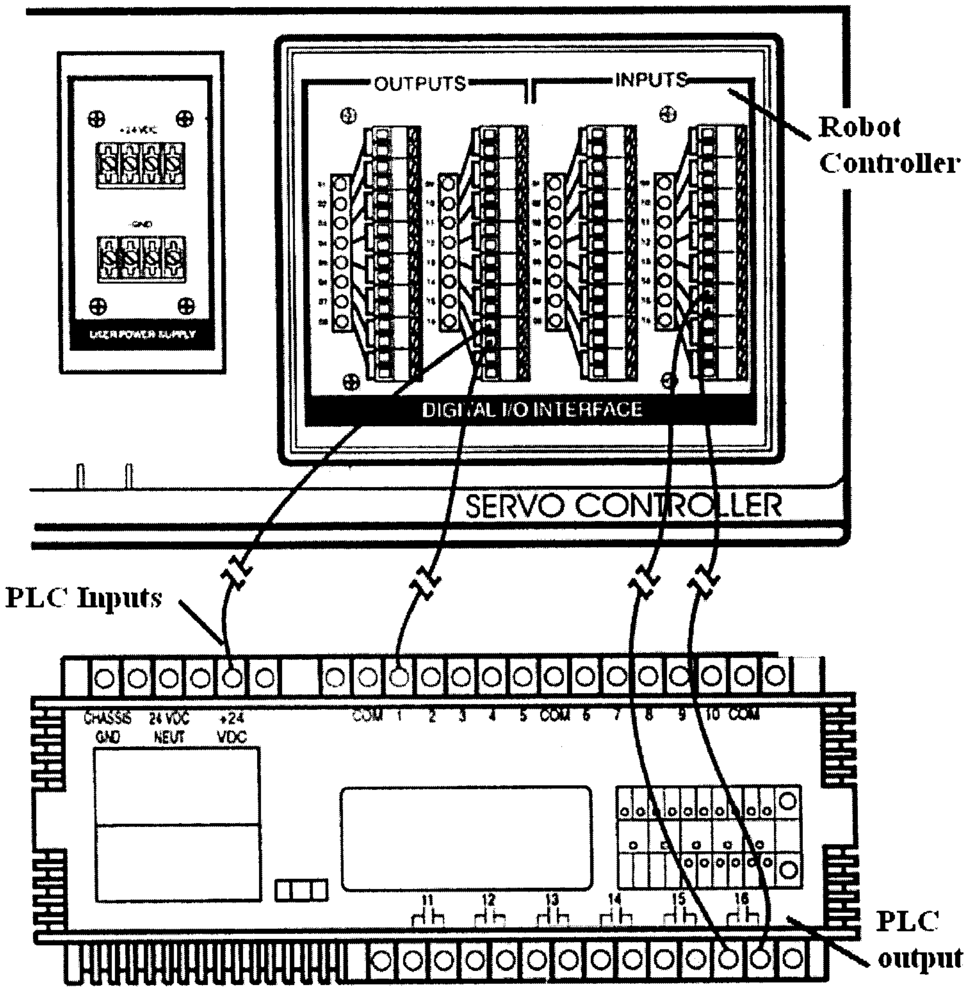


Figure 4-4. Interface between PLC and Pegasus robot

The PLC outputs were used to transmit the control related commands from the server to the robot and signals from the sensor to the robot as well. Table 4-3 briefly describes the I/O list between PLC outputs and robot inputs.

Table 4-3

I/O list between PLC Output and Robot Input

PLC Output	Robot Input	Description
O0	I6	Identify inspection passed/failed status
O1	I1	Switch auto/manual mode
O2	I2	Start getting the block from the part feeder
O3	I3	Start moving the block to the inspection station
O4	I4	Start moving the block to the rejected bin
O5	I5	Start moving the block to the accepted bin

PLC inputs were used to process incoming equipment related data and send them back to the server. Table 4-4 list those digital inputs the PLC monitored. The robot outputs were used to transmit control feedback as the PLC inputs. A limit switch resided in the inspection station was used to test for the presence of a hole in the block and to send the result signals to the PLC.

Table 4-4

Digital Inputs Monitored by the PLC

PLC Input	Sources		Description
	Robot Output	Limited Switch	
I0	O2		The status of step 2
I1	O4		The status of step 3
I2	O6		The status of step 4b
I3	O5		The status of step 4a
I4	O10		The status of step 1
I5		LS1	The inspection result

Note. Detailed information regarding particular steps can be found in Table 4-1

4.1.5 Robot and PLC programming

After the robot and controller are wired correctly as described above, programs need to be written to implement the process sequence. The general sequence of operations (see Table 4-1) listed each action or communication that took place in order for the robot to perform the designed task. The specific programming code along with the corresponding sequence of operations can be found in Appendix A.

In this system, the PLC acted as an input/output interface between robot and server computer. Each discrete input signal was directed to the N7:1 register for Visual Basic program to read. Table 4-5 summarizes the inputs and descriptions of their corresponding locations. N7 is a type of integer data table file, which is used to store numeric values or bit information. N7:1 is an element address, where the colon separates the file type and number from the element. Since N7 files have 1-word elements, the address N7:1 points to word #1 in integer file #7. In this manner, six discrete input signals were occupied in N7:1 register from Bit 0 to Bit 5.

Table 4-5

N7:1 Register Table

	Read Registers							
N7:1	Bit 0	Bit 1	Bit 2	Bit 3	Bit 4	Bit 5	Bit 6	Bit 7
	I0	I1	I2	I3	I4	I5		
	Step 2 Status	Step 3 Status	Step 4b Status	Step 4a Status	Step 1 Status	Inspection Result	Not Used	Not Used
	Bit 8	Bit 9	Bit 10	Bit 11	Bit 12	Bit 13	Bit 14	Bit 15
	Not Used	Not Used	Not Used	Not Used	Not Used	Not Used	Not Used	Not Used

Similar to the input operations, control commands have been written to their corresponding PLC outputs from Register N7:0. Table 4-6 summarizes the PLC outputs and descriptions of their corresponding locations. Six output signals were occupied in N7:0 register from Bit 0 to Bit 5.

Table 4-6

N7:0 Register Table

	Write Registers							
N7:0	Bit 0	Bit 1	Bit 2	Bit 3	Bit 4	Bit 5	Bit 6	Bit 7
	O0	O1	O2	O3	O4	O5	Not Used	Not Used
	Bit 8	Bit 9	Bit 10	Bit 11	Bit 12	Bit 13	Bit 14	Bit 15
	Not Used	Not Used	Not Used	Not Used	Not Used	Not Used	Not Used	Not Used

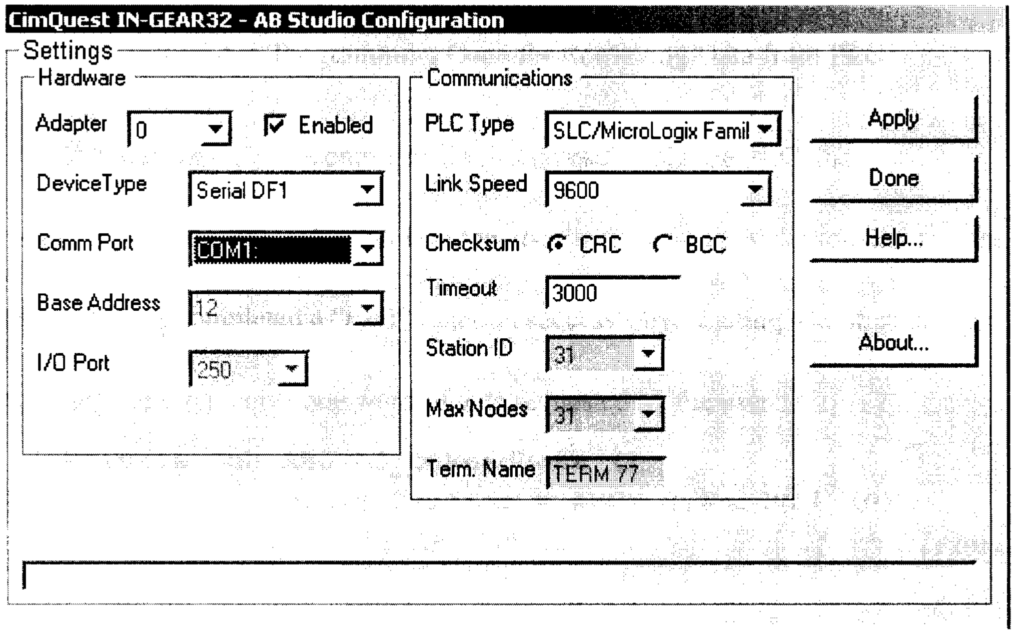
4.2 Server Environment

4.2.1 Communication with PLC

The communication between server and PLC is conducted in ASCII format through RS-232 serial port, which was enabled by DF1 ActiveX Control. The fundamental idea in this part was to retrieve data from a PLC and to send user related command back to the same controller. Two instances of DF1 ActiveX control by the name of "ABCTL1" and "ABCTL3" were used on this application for "Write" and "Read" functions, respectively. Following software components were defined to perform PLC connectivity function.

Driver configuration.

In order for DF1 ActiveX Control to function, the driver must be configured. Figure 4-5 illustrates the configuration of the communication driver. It provides the capability to apply hardware and communication characteristics which is independent of the ActiveX. In this study, SLC/MicroLogix Family PLC with baud rate of 9600 bps at COM 1 port was selected.


CimQuest IN-GEAR32 - AB Studio Configuration

Settings

Hardware

Adapter	0 ☑ Enabled
DeviceType	Serial DF1
Comm Port	COM1:
Base Address	12
I/O Port	250

Communications

PLC Type	SLC/MicroLogix Famil
Link Speed	9600
Checksum	● CRC ○ BCC
Timeout	3000
Station ID	31
Max Nodes	31
Term. Name	TERM 77

Figure 4-5. Communication driver configuration

Read data from the PLC.

In order to retrieve data from the PLC, code was created in the Visual Basic editor as shown in Listing 4-1. It sets the user input data for the read function (value 0), memory address, number of words, and poll rate and maps them into the appropriate properties in the ABCTL3 ActiveX control. Then it invokes the AutoPoll method and passes the poll rate.

```
ABCTL3.Function = 0         ' Set the function to read
ABCTL3.FileAddr = "N7:1"    'Start at register N7:1
ABCTL3.Size = 1              'Read one word(16 bits)
ABCTL3.AutoPoll 500          'Poll every 500ms
ABCTL3.ClearControl
'Clear the content of control arrays
```

Listing 4-1. Programming Code for Retrieving Data from PLC

When the ABCTL3 control receives data back from the PLC, the "ABCTL3_OnReadDone" event is automatically triggered which indicates that the controller has completed a "read" function successfully. Listing 4-2 illustrates the code segment to receive one word of data requested by the user.

```
    Private Sub ABCTL3_OnReadDone ()
        lblReadResult.Caption = ABCTL3.WordVal (0)
    'Append data in the ABCTL3 to the caption of the label
"lblReadResult"

    End Sub
```

Listing 4-2. Programming Code for Onreaddone Method

Write data to the PLC.

In order to leave Read function uninterrupted, a different instance of DF1 ActiveX control by the name of ABSTL1 was used on the form for Write function. To write data to the PLC requires the same actions as described above but in reverse.

The code segment used to write data to the PLC is listed in Listing 4-3. This code retrieves the address that the data will write to, sets the "Function" property to write (value 1), puts the data value into control's data array, and invokes the "Trigger" method to write data one time.

```
ABCTL1.Function = 1                        'Write to PLC
ABCTL1.FileAddr = "N7:0"   'Write data to register N7:0
ABCTL1.Size = 1                     'Set size to one word
ABCTL1.WordVal (0) = 1                     'Set data value
ABCTL1.Trigger       'Invokes the control to write PLC
```

Listing 4-3. Programming Code for Writing Data to PLC

Similar to "OnReadDone" event, the "ABCTL1_OnWriteDone" event is automatically triggered, indicating that the control has completed successfully function "Write" after the ABCTL1 control writes data back to the PLC.

4.2.2 Communication with client

The communication between client and server was established by using Microsoft Winsock control. Two separate applications were created for the communication. The first application was resided in the server computer, and the other in client computer. Both client and server applications interacted with each other to exchange data.

Client application.

The code segment required to connect to the server is listed in Listing 4-4. "RemotePort" and "RemoteHost" properties are initialized when form is loaded. The port on which the server program will be listening is set in "RemotePort" property. Port number 1600 was used in this study. "RemoteHost" property can be set as either the host name or IP address of the server computer. In this case, an IP address of "LocalHost" (192.67.1.1) was used. Once the "Connection" button is pushed, the state of the Winsock control is first checked. If it is not closed, "Close" method is called to terminate prior connection.

```
Private Sub Form_Load()
   WinsockClient.RemotePort = 1600
'Set client port number to 1600
   WinsockClient.RemoteHost =192.67.1.1
'Set server IP to 192.67.1.1
End Sub

Private Sub cmdConnect_click()
  If    WinsockClient.State    <>    sckClosed    Then
      WinsockClient.Close
'If winsock is not closed, then close it
        WinsockClient.Connect
'Invoke connect method
End Sub
```

Listing 4-4. Programming Code for Connecting to Server Application

Next, data needs to be sent to the server application using "SendData" method. Code segment in this procedure is listed in Listing 4-5. All data received will be initiated with the "Data Arrival" event. To copy the data into a variable, "GetData" method is used. The "Click" event procedure for the "cmdExecute" button is used to accomplish the task of sending a "CommandData" with value 5. The "Data Arrival" event is used to fire the "GetData" method to retrieve all data into "ResponseData" variable. These data are then appended to the caption of the "lblStatus" label.

```
Private Sub cmdExecute_Click()
   Dim CommandData As String
      CommandData = 5
   'Set "CommandData" with value 5
      WinsockClient.SendData CommandData
   'Invoke "Senddata" method to send "CommandData"
   End Sub

Private Sub Winsockclient_DataArrival(ByVal bytesTotal As
Long)
   Dim RsponseData As String
      WinsockClient.GetData ResponseData
   'Invoke "GetData" method to retrieve all data into
"ResponseData" variable.
       lblStatus.Caption = ResponseData
   'Append all data to the caption of the label
"lblReadResult"

   End Sub
```

Listing 4-5. Programming Code for Sending/Receiving Data to Server Application

<u>Server application.</u>

The server listens for a connection request on its assigned port. Codes in this action are listed in Listing 4-6. The "LocalPort" property is initialized when form is loaded. In this research, the initialization code sets the server port to 1600 and invokes "Listen" method.

```
    Private Sub Form_Load()
       WinsockServer.LocalPort = 1600
    'Set server port number to 1600
       WinsockServer.Listen
    'Invoke listen method to listen for a connection
request
    End Sub
```

Listing 4-6. Programming Code for Connection to Client Application

When a client program executes "Connect" method with a server port number, the server's "ConnectionRequest" event procedure is executed to fire the "Accept" method with the "requestID" passed to the "ConnectionRequest" event procedure. Codes implemented in this operation are listed in List 4ing-7. The state of the Winsock will be checked first, making sure that any prior connection is terminated. The connection is accepted by invoking "Accept" method.

```
    Private Sub Winsockserver_ConnectionRequest(ByVal
requestID As Long)
   If WinsockServer.State <> sckClosed Then
        WinsockServer.Close
   'If winsock is not closed, then close it
          WinsockServer.Accept requested
    'Invoke "Accept" method to accept connection request
    End Sub
```

Listing 4-7. Programming Code for Connection to Client Application

Receiving and sending data in the server is identical to the client. Codes listed in Listing 4-8 accomplish this task.

```
Private Sub Winsockserver_DataArrival(ByVal bytesTotal As
Long)
    Dim buffer As String
    WinsockServer.GetData   buffer
    Text1.Text = buffer
    'Invoke "GetData" method to retrieve data to "buffer"
variable
    WinsockServer.SendData temper
    'Invoke "SendData" method to send the value of "temper"
variable
End Sub
```

Listing 4-8. Programming Code for Sending/Receiving Data to Client Application

Once the client terminates the connection, "Close" event procedure of the server is fired, which is listed in Listing 4-9. This event is used to reinitiate listening.

```
Private Sub WinsockServer_Close()
   If WinsockServer.State <> sckClosed Then
   WinsockServer.Close
   'If winsock is not closed, then close it
End Sub
```

Listing 4-9. Programming Code for Reinitiating Listening in Server Application

The detailed codes for the server programming can be found in Appendix B.

4.3 Client Environment

Client environment allows user to monitor or control the networked process remotely through a server. Using Visual Basic programming, a human machine interface (HMI) was designed and developed on the client for this research. The data display and input, as well as animations of the control performance were accomplished using built in objects such as text boxes, labels, lists, and command buttons.

Figure 4-6 illustrates the human machine interface running on the client computer. Details of this layout are explained as follows.

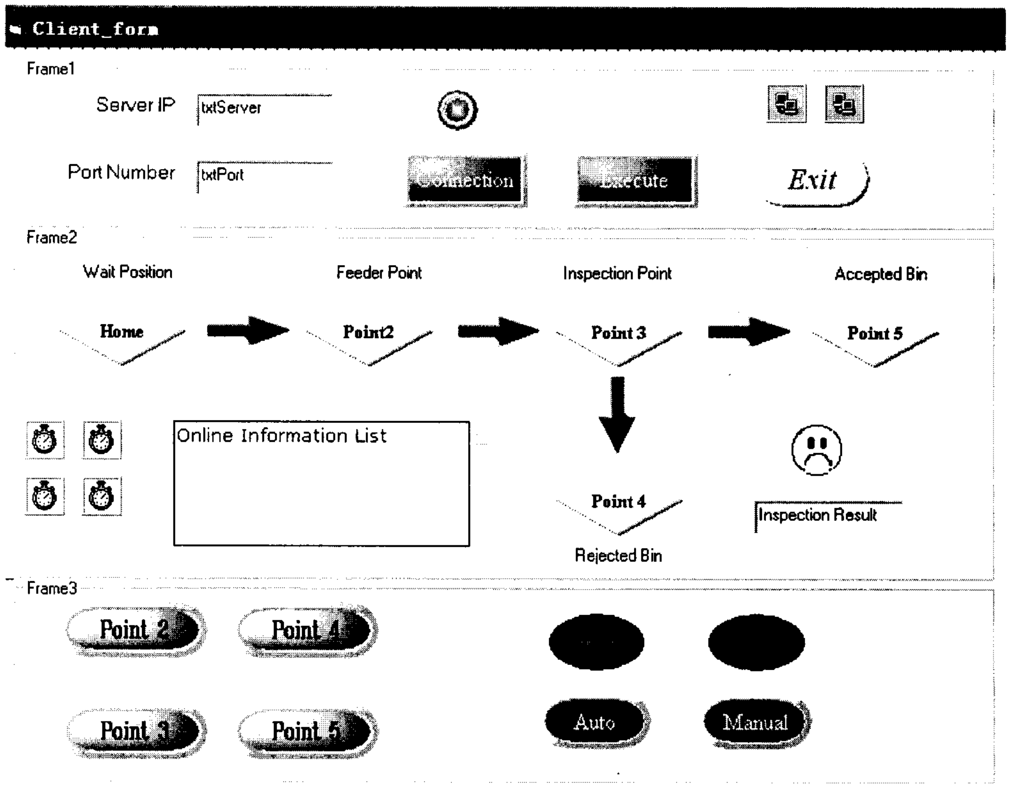


Figure 4-6. Human machine interface (HMI) on client

Frame 1 was used to set the communication parameters. Servers IP address and port numbers are defined in the “txtServer” and “txtPort” textbox, respectively. “Connection” button is used to send connection request to the server. After “Execute” command button is pushed, the control program takes over to execute the designed process remotely.

Frame 2 allows users to monitor the whole process step by step. " Home", "Point2", "Point3", "Point4", "Point5" labels correspond to the "Wait Position," "Feeder Point," "Inspection Point," "Accepted Bin," and "Rejected Bin" points in the Robot Point File, respectively. The color of the label changes from grey to green when robot moves into the corresponding position. Arrows activated by timer flash in the process of a specific action to help the user better understand the current status. For example, when the robot is moving from "Wait Position" to "Feeder Point", the arrow between these two points flashes until the desired task is completed.

"Inspection Result" text box will pop up and indicate the inspection result when the robot drops the object into the inspection station. A smile face icon and an upset face icon express the results of "Inspection Passed" and "Inspection Failed," respectively. "On-Line Information List" provides dynamic text information for the status of the robot, and on-line help. For example, after "Execute" button is pushed, the content of text in the "On-Line Information List" will changed to "Loading program to PLC, the robot is ready for your command. Please select auto or manual mode."

Frame 3 was used to allow user to control online process remotely through server. The user can change system control mode by pushing either "Auto" or "Manual" command button. The robot can be manually regulated with the use of commands through "Point2", "Point3", "Point4", and "Point5" command buttons.

Indicators and color-coding were used to reflect the state changes of a particular command button.

The detailed programming code for the client environment can be found in Appendix C.

CHAPTER 5

Analyses

This study discusses the structure and hierarchical model of a networked control system (NCS). Issues addressed in this research include serial communication, network communication, human machine interface programming, and client/server programming.

5.1 System Functionality

A NCS prototype was designed and implemented in this study. The functions can be accomplished with designed system are as follows:

1. An operator is able to remotely regulate a robot to perform a material sorting process through a Human Machine Interface (HMI).
2. Position, along with the current action of the robot can be monitored.
3. The server computer is able to receive and process commands transmitted from the client and subsequently forward instructions to the robot.

5.2 Advantages of NCS

Based on the prototype above, major advantages of NCS in contrast to traditional point-to-point communication can be identified as below:

1. Remote control of processes is accessible and controllable. The idea proposed here can be applied to industrial automation under health-critical or dangerous conditions.
2. Adding and deleting a client is feasible since Internet/Ethernet has been

used as the communication channel. In fact, the client can be resided at anywhere as long as an Internet connection is accessible.

3. In stead of direct electrical wiring between one and another, control components can be interconnected by a common network channel, which offers better resource utilization, and also reduces installation, maintenance time and costs.

5.3 Implementation Experience

The designed prototype combined both network and control practices, providing knowledge in hardware, software, control design and implementation. Some of major knowledge gained during the designing and implementing process are summarized below:

5.3.1 Control process design

A basic material sorting process was designed and implemented in this study. A PLC was used as the process controller to communicate with server and regulate a Pegasus robot to perform the desired task.

The procedure used in the implementation of this control process is as below:

1. Draw a process schematic for designed process.
2. Write the general sequence operation.
3. Specify inputs/outputs terminal.
4. Programming.
5. Module and Integrating test.

5.3.2 PLC communication

In order to talk to the PLC, the protocol for messaging of the Allen Bradley's PLC was studied. Two layers of software were involved in this communication including data-link layer and application layer. Data link layer handles the flow of communication over the physical link. The transmission format is as follows:

Start bit—8 data bit (0~7)—no parity—stop bit

Application layer interprets commands, and formats user data into packets. The unprotected write command and the unprotected read command have been used to access to the PLC's data memory. The address fields of unprotected reads and unprotected writes were used as word addresses.

5.3.3 Communication network

Because of its low cost, availability, and higher communication rates, Ethernet was selected as the communication network. The communication between client and server was established using Microsoft Winsock control.

By setting properties and invoking control methods, a client-server application has been applied in this study. The client application is the one that requests a network connection, and the server listens for a connection request. Both client and server applications interacted with each other to exchange data.

5.3.4 HMI Programming

On the client side, a Human Machine Interface (HMI) was programmed to retrieve information from controller, which allowed operators to monitor and control the system. In this study, Visual Basic was used as the programming language for HMI. The data input and process status display were accomplished using built in objects such as text boxes, labels, lists, and command buttons.

5.4 General Procedure in NCS Design and Implementation

In general, a NCS design shall proceed through the following phases: (a) requirements definition, (b) design, (c) implementation, and (d) testing. It is not necessary to fully complete one phase before beginning the next. It may, for example, be useful to implement certain key features of the system before the entire design is complete. In addition, it may be necessary to reiterate to an earlier phase to make modifications if additional information is discovered during a subsequent phase. A general procedure used for NCS design and implementation is described below:

1. System's operational requirements. In this step, the operational requirement for the designed system has been indentified. It is generally defined with the reference to: Functionality and Interfaces.
2. System definition. In this step, the requirements of the system are analyzed. Overall system architecture design is performed and behavior specifications are made.
    a) Hardware configuration. In this step, the controller hardware along with control platform, machine tools, and communication network to be used are specified.

   b) Software configuration. In this step, a high level functional model of software structure including operation system, programming software is defined.

3. Implementation. Software and hardware are developed to meet the design objectives at this stage.

4. Module test and system integration. In this step, all the individual components are tested, and then they are combined and integrated together to ensure that the original design requirements are met.

CHAPTER 6

Summary

This study provides details on designing and implementing a Networked Control System (NCS). This prototype combines both network and control practices, encompassing an expansive understanding in hardware, software, control design and implementation.

A client/server prototype was designed to perform a remote process control via Ethernet. Through the human machine interface, an operator is able to remotely regulate and monitor a material sorting process.

Major issues considered in NCS design and implementations include:

1. Control process. The detailed procedure implemented in the process controller along with hardware configuration, application software programming have been discussed.
2. PLC communication. The communication between server and PLC which was enabled by DF1 ActiveX Control was explored.
3. Network communication. The communication between client and server which was established by Microsoft Winsock control has been presented.
4. Human Machine Interface (HMI) programming. A HMI developed with Visual Basic program has been introduced.

CHAPTER 7

Recommendations for Future Work

Study of NCS (Networked Control System) is related to both network and control system. This research provided a prototype for a typical NCS. The designed system can help guide future controller design to investigate topics within control system designs and interaction between network configuration and control parameters. Some potential topics may include:

1. Key timing parameters stemming from the network architecture and the major impact of time delay on control applications.
2. Impact of network architecture and device performance on control performance in NCSs.
3. Design considerations related to control quality of performance as well as network quality of service.

References

Abdullah, H. A., & Chatwin, C. R. (1994). Distributed c3 environment for small to medium-sized enterprises. *Integrated Manufacturing Syst.*, 5(3), 20–28.

Allen Bradley. (1996). *DF1 protocol and command set.* [Reference Manual]. Mayfield Hts., OH: Author.

Altun, Z. G., Topaloglu U. M., Saygin A. V., & Bayrak, C. (2001). Process control via internet. *Integrated Design and Process Science*, 5(2), 111-122.

Cisco Systems, Inc. (2001). Layer 2: Technologies. In J. Wait (Eds.), *Cisco networking academy program: first-year companion guide* (pp. 239-280). Indianapolis, IN: Cisco Press.

Etkin, B., & Reid. (1996). *Dynamics of flight: stability and control.* New York: Wiley.

Hugh, J. (1996). Integration and automation of manufacturing systems. Retrieved Mar. 27, 2005, from Books Page Web site: http://claymore.engineer.gvsu.edu/~jackh/books/integrated/.

Hugh, J. (2004). Automating manufacturing systems with PLCs. Retrieved Mar. 27, 2005, from Books Page Web site: http://claymore.engineer.gvsu.edu/~jackh/books/plcs/.

International Center for Theoretical Physics, (1996). Windows sockets 2 application programming interface. Retrieved Mar. 28, 2005, from Networking and Radio communications Web site: http://www.ictp.trieste.it/~radionet/nuc1996/ref/winsock/wsapi22.htm.

Kaplan, G. (2001). Ethernet's winning ways. *IEEE Spectrum*, *38*(1), 113-115.

Koren, Y. (1999). The third year report: 1998-1999 (Tech. Rep.). University of Michigan, NSF-Engineering Research Center for Reconfigurable Machining Systems.

Koubias, S., & Papadopoulos, G. (1995). Modern fieldbus communication architectures for real-time industrial application. *Computers in Industry*, *26*(3), 243-252.

Leonik, T. E. (2000).Serial communication basics. In Gurdian, W. (Ed.), *Home automation basics: Practical application using visual basic 6* (pp. 37-76). Indianapolis, IN: Prompt Publication.

Lian, F. (2001). *Stability analysis of networked control systems.* Unpublished doctoral dissertation, University of Michigan, Ann Arbor.

Lian, F, Moyne, J. R., & Tibury, D. M. (1999). Performance evaluation of control networks: Ethernet, controlnet and devicenet. *ASME Dynamic Systems and Control Division, Vol. 67., pp.853-860,* Nashville, TN.

Lian, F, Moyne, J. R., & Tibury, D. M. (2000a, July). Implementation of networked machine tools in reconfigurable manufacturing systems. *Proceeding of Japan-USA Symposium on Flexible Automation,* Ann Arbor, MI.

Lian, F, Moyne, J. R., & Tibury, D. M. (2000b). Performance evaluation of control networks: ethernet, controlnet, and devicenet., . *IEEE Control Systems*, *21*(1), 66-83.

Lian, F, Moyne, J. R., & Tibury, D. M. (2001, November). Time delay modeling and sample time selection for networked control systems. *Proceeding of ASME Dynamic Systems and Control Division, Symposium conducted at the meeting of International Mechanical Engineering Congress and Exposition,* New York, NY.

Otanez, P. (2002). *Performance optimization of networked control systems.* Unpublished doctoral dissertation, University of Michigan, Ann Arbor.

Otanez, P.G, Parrott, J. T., Moyne, J.R. & Tilbury, D.M. (2000).*The implications of ethernet as a control network* (Tech. Rep.). University of Michigan, Engineering Research Center for Reconfigurable Manufacturing Systems.

Raji, R. (1994). Smart networks for control. *IEEE Spectrum, 31*(6), 49–55.

Tipsuwan, Y. (2003). *Gain scheduling for networked control system.* Unpublished doctoral dissertation, North Carolina State University, Raleigh.

Yodyium, T. (2003). *Gain scheduling for networked control system.* Unpublished doctoral dissertation, North Carolina State University.

Walsh, G., & Ye, H. (2001). Scheduling of networked control systems . *IEEE Control Systems Magazine, 21*(1), 57-65.

Weber, J. (1999, April). *Applying visual basic for human machine interface applications.* Presented on the International Manufacturing Software Show, Orlando, FL.

Zeltwanger, H. (2000, Octobers). S*tate-of-the-art can applications and future requirements*. Presented on the 7th International CAN Conference, Amsterdam, Netherlands.

Appendixes

Appendix A

Robot Programming Code

```
LABEL1:
DELAY 100
WRITEO 10,1
WRITEO 2,0
WRITEO 4,0
WRITEO 5,0
WRITEO 6,0


IF INP(1) = 1 THEN
CALL AUTO
ENDIF
IF INP(1)=0 AND INP(2)=1 THEN
CALL GETPART
WAITI 2,0
CALL INSPECT
WAITI 3,0
IF INP(4)=1 THEN
CALL SOLIDBIN
WAITI 4,0
ENDIF
IF INP(5)=1 THEN
CALL HOLEBIN
WAITI 5,0
ENDIF
ENDIF
BRANCH LABEL1


SUB AUTO
SPEED 100
DELAY 100
PMOVE TP[31]
PMOVE TP[32]
SPEED 50
PMOVE TP[33]
```

```
GRASP
PMOVE TP[32]
SPEED 100
PMOVE TP[31]
WRITEO 10,0
WRITEO 2,1
WRITEO 4,0
WRITEO 5,0
WRITEO 6,0

SPEED 100
DELAY 100
PMOVE TP[31]
PMOVE TP[34]
SPEED 30
PMOVE TP[35]
RELEASE
WRITEO 10,0
WRITEO 2,0
WRITEO 4,1
WRITEO 5,0
WRITEO 6,0

IF INP(6)=1 THEN
SPEED 30
DELAY 100
PMOVE TP[35]
GRASP
PMOVE TP[34]
WRITEO 14,0
SPEED 50
PMOVE TP[37]
RELEASE
WRITEO 2,0
WRITEO 4,0
WRITEO 5,0
WRITEO 6,1
PMOVE TP[31]
DELAY 100
WRITEO 10,1
WRITEO 6,0
DELAY 200
```

```
ELSE
SPEED 30
DELAY 100
PMOVE TP[35]
GRASP
PMOVE TP[34]
SPEED 50
PMOVE TP[31]
PMOVE TP[36]
RELEASE
WRITEO 2,0
WRITEO 4,0
WRITEO 5,1
WRITEO 6,0
PMOVE TP[31]
DELAY 100
WRITEO 10,1
WRITEO 5,0
DELAY 200
ENDIF
RETURN



SUB GETPART
SPEED 100
DELAY 100
PMOVE TP[31]
PMOVE TP[32]
SPEED 50
PMOVE TP[33]
GRASP
PMOVE TP[32]
SPEED 100
PMOVE TP[31]
WRITEO 10,0
WRITEO 2,1
WRITEO 4,0
WRITEO 5,0
WRITEO 6,0
RETURN
```

```
SUB INSPECT
SPEED 100
DELAY 100
PMOVE TP[31]
PMOVE TP[34]
SPEED 30
PMOVE TP[35]
RELEASE
WRITEO 10,0
WRITEO 2,0
WRITEO 4,1
WRITEO 5,0
WRITEO 6,0
RETURN

SUB HOLEBIN
SPEED 30
DELAY 100
PMOVE TP[35]
GRASP
PMOVE TP[34]
SPEED 50
PMOVE TP[31]
PMOVE TP[36]
RELEASE
WRITEO 2,0
WRITEO 4,0
WRITEO 5,1
WRITEO 6,0
PMOVE TP[31]
DELAY 100
WRITEO 10,1
WRITEO 5,0
DELAY 200
RETURN

SUB SOLIDBIN
SPEED 30
DELAY 100
PMOVE TP[35]
GRASP
PMOVE TP[34]
```

```
WRITEO 14,0
SPEED 50
PMOVE TP[37]
RELEASE
WRITEO 2,0
WRITEO 4,0
WRITEO 5,0
WRITEO 6,1
PMOVE TP[31]
DELAY 100
WRITEO 10,1
WRITEO 6,0
DELAY 200
RETURN
```

Appendix B

Programming Code for the Server Environment

```
'*******************************************************
***************
Private Sub Form_Load()
    lblHostID.Caption = WinsockServer.LocalHostName
    lblAddress.Caption = WinsockServer.LocalIP


    WinsockServer.LocalPort = 1600 'sets or returns the
local port number
    WinsockServer.Listen  'wait for a TCP request

    Winsock1.LocalPort = 1300
    Winsock1.Listen

End Sub

Private Sub BtnClose_Click()
    End
End Sub

'*****************************************************
'winsock programming section

Private Sub WinsockServer_Close()
    WinsockServer.Close          'close
End Sub

Private Sub Winsockserver_ConnectionRequest(ByVal
requestID As Long)

    'judge if the connectin is closed, before make a new
connectin
    If WinsockServer.State <> sckClosed Then
WinsockServer.Close
    WinsockServer.Accept requestID

    ABCTL3.Function = 0  'read PlC for one word(16 bits)
    ABCTL3.FileAddr = "N7:1" 'start at N7:1
    ABCTL3.Size = 1 '
    ABCTL3.AutoPoll 500 'poll every 500ms
    ABCTL3.ClearControl
End Sub

Private Sub Winsockserver_DataArrival(ByVal bytesTotal As
Long)
    Dim buffer As String
    WinsockServer.GetData buffer
    Text1.Text = buffer
    Select Case buffer
        Case 0       'send getpart command
            ABCTL1.FileAddr = "N7:0"
            ABCTL1.Function = 1  'Write to PLC
            ABCTL1.Size = 1 'write one word
            ABCTL1.WordVal(0) = 1 ' decimal= 00001
            ABCTL1.Trigger  'trigger
            'lblResult.Caption = "Light is On!"
        Case 1
            ABCTL1.FileAddr = "N7:0"
            ABCTL1.Function = 1  'Write to PLC
            ABCTL1.Size = 1 'write one word
            ABCTL1.WordVal(0) = 2 ' decimal= 00010
            ABCTL1.Trigger  'trigger
            'lblResult.Caption = "Light is On!"
        Case 2
            ABCTL1.FileAddr = "N7:0"
            ABCTL1.Function = 1  'Write to PLC
            ABCTL1.Size = 1 'write one word
            ABCTL1.WordVal(0) = 4 ' decimal= 00100
            ABCTL1.Trigger  'trigger
            'lblResult.Caption = "Light is On!"
        Case 3
            ABCTL1.FileAddr = "N7:0"
            ABCTL1.Function = 1  'Write to PLC
            ABCTL1.Size = 1 'write one word
            ABCTL1.WordVal(0) = 8 ' decimal= 01000
            ABCTL1.Trigger  'trigger
            'lblResult.Caption = "Light is On!"
        Case 4
            ABCTL1.FileAddr = "N7:0"
            ABCTL1.Function = 1  'Write to PLC
            ABCTL1.Size = 1 'write one word
            ABCTL1.WordVal(0) = 16 ' decimal= 10000
            ABCTL1.Trigger  'trigger
            'lblResult.Caption = "Light is On!"
        Case 5
            ABCTL1.FileAddr = "N7:0"
            ABCTL1.Function = 1  'Write to PLC
            ABCTL1.Size = 1 'write one word
            ABCTL1.WordVal(0) = 0 ' decimal= 00000
            ABCTL1.Trigger  'trigger
            'lblResult.Caption = "Light is On!"
        Case Else
            ABCTL1.FileAddr = "N7:0"
            ABCTL1.Function = 1  'Write to PLC
            ABCTL1.Size = 1 'write one word
            ABCTL1.WordVal(0) = 0 ' decimal= 00000
            ABCTL1.Trigger  'trigger
            'lblResult.Caption = "Light is On!"
    End Select
End Sub

' see if command is sent to PLC successfully

Private Sub ABCTL1_OnErrorEvent(ByVal nErrorCode As
Integer)
    lblWriteResult.Caption = "Write Error" & nErrorCode
End Sub

Private Sub ABCTL1_OnWriteDone()
     lblWriteResult.Caption = "Write Done!"
End Sub

Private Sub lblWriteResult_Change()
    WinsockServer.SendData lblWriteResult.Caption
'senddate method
End Sub

'*******************************************************
*************
' Read data from PLC

Private Sub cmdRead_Click()
```

```
    ABCTL3.Function = 0  'read PlC for one word(16 bits)
    ABCTL3.FileAddr = "N7:1" 'start at N7:1
    ABCTL3.Size = 1 '
    ABCTL3.AutoPoll 500 'poll every 500ms
    ABCTL3.ClearControl

End Sub

Private Sub ABCTL3_OnReadDone()
    ' Refer to Data Access Methods/BitVal
    lblReadResult.Caption = ABCTL3.WordVal(0)
End Sub
Private Sub ABCTL3_OnErrorEvent(ByVal nErrorCode As
Integer)
    lblReadResult.Caption = "Read Error" & nErrorCode
End Sub



'*******************************************************
******
'Send data to Client

Private Sub Winsock1_Close()
    Winsock1.Close          'close
End Sub

Private Sub Winsock1_ConnectionRequest(ByVal requestID As
Long)
      'judge if the connectin is closed, before make a new
connectin
    If Winsock1.State <> sckClosed Then Winsock1.Close
    Winsock1.Accept requestID
End Sub

Private Sub lblReadResult_Change()

    Winsock1.SendData lblReadResult.Caption  'senddate
method
End Sub
```

Appendix C

Programming Code for the Client Environment

```
Private Sub txtMoniter_Change()
    txtMoniter.Refresh
    'Timer1.Enabled = True
    'If lblBmp.Visible = False Then
        'lblBmp.Visible = True
    'Else
        'lblBmp.Visible = False
    'End If
End Sub


'******************************************
' Test Program

'*****************************************************
'Arrow animation
Private Sub Timer2_Timer()
 Timer2.Interval = 200
 Call lblArrow2.UpdateValue(Not lblArrow2.Value)
End Sub

Private Sub Timer3_Timer()
 Timer3.Interval = 200
 Call lblArrow3.UpdateValue(Not lblArrow3.Value)
End Sub

Private Sub Timer4_Timer()
 Timer4.Interval = 200
 Call lblArrow4.UpdateValue(Not lblArrow4.Value)
End Sub

Private Sub Timer5_Timer()
 Timer5.Interval = 200
 Call lblArrow5.UpdateValue(Not lblArrow5.Value)
End Sub
```

```
'****************************************************

Private Sub Form_Load()
    WinsockClient.RemotePort = 1600  'winsockclient
handles sending commands to client
    WinsockClient.RemoteHost = txtHost.Text
    Winsock1.RemotePort = 1300  'winsock3 handles getting
response from client
    Winsock1.RemoteHost = txtHost.Text

    lblBlock.Visible = False
    lblBlockStatus.Visible = False

    Timer3.Enabled = False
    Timer2.Enabled = False
    Timer4.Enabled = False
    Timer5.Enabled = False


    cmdExecute.Enabled = False
    cmdManual.Enabled = False
    cmdAuto.Enabled = False
    cmdPT2.Enabled = False
    cmdPT3.Enabled = False
    cmdPT4.Enabled = False
    cmdPT5.Enabled = False

    txtMoniter.Text = "This system is designed to remotely
control a pegasus robot by the nickname of Chico.  Please
start the system by pushing Connection button above."
End Sub

Private Sub cmdExit_click()
    End
End Sub

'*********************************************
' Send command to server

Private Sub cmdConnect_click()
    'If WinsockClient.State <> sckClosed Then
WinsockClient.Close
```

```
    WinsockClient.Connect  'request a connectin to the
remote computer
    Winsock1.Connect


    cmdConnect.Enabled = False
End Sub



'WinsockClient--show if command is sent to PLC succefully

Private Sub Winsockclient_Connect()
   lblConnect.OnColor = &HFF00&
   cmdExecute.Enabled = True

   txtMoniter.Text = "Connection to Server is set up!"
End Sub

Private Sub WinsockClient_Close()
    WinsockClient.Close
End Sub

Private Sub Winsockclient_DataArrival(ByVal bytesTotal As
Long)
    Dim RsponseData As String
    WinsockClient.GetData RsponseData
    lblStatus.Caption = RsponseData
End Sub

Private Sub cmdExecute_Click()
    cmdManual.Enabled = True
    cmdAuto.Enabled = True
    cmdExecute.Enabled = False

    cmdPT2.Enabled = False
    cmdPT3.Enabled = False
    cmdPT4.Enabled = False
    cmdPT5.Enabled = False

    Dim CommandData As String
```

```
        CommandData = 5   'set motor to home postion, wait
for command
        WinsockClient.SendData CommandData

     txtMoniter.Text = "Loading program to PLC, Chico is
ready for your command. Please select auto or manual mode."
End Sub

Private Sub cmdAuto_Click()
    If lblManual.Value = True Then  'when system is at
manual mode
        lblAuto.Value = True
        lblManual.Value = False
        cmdAuto.Enabled = False
        cmdManual.Enabled = True
        cmdPT2.Enabled = False
        Timer2.Enabled = True

        Dim CommandData As String
        CommandData = 4    ' set motor to auto mode,start
move
        WinsockClient.SendData CommandData

        txtMoniter.Text = "Chico is grabbing a block from
part feeder..........."
    Else

    cmdManual.Enabled = True
    cmdAuto.Enabled = False
    'cmdPT2.Enabled = False
    'cmdPT3.Enabled = False
    'cmdPT4.Enabled = False
    'cmdPT5.Enabled = False

    lblAuto.Value = True
    lblManual.Value = False

    Timer2.Enabled = True
    'Timer3.Enabled = False
    'Timer4.Enabled = False
    'Timer5.Enabled = False
    'Dim CommandData As String
    CommandData = 4    ' set motor to auto mode,start move
    WinsockClient.SendData CommandData

     txtMoniter.Text = "Chico has been changed to the Auto
mode!"
    End If
End Sub

Private Sub cmdManual_Click()
    If lblAuto.Value = True Then
        s$ = MsgBox("Chico will be switched to the Manual
mode after this aumation period is finished")
        'If Home <> 1 Then
            'cmdPT2.Enabled = False
        'End If
        'If Home = 1 Then
            'cmdPT2.Enabled = True
        'End If
        cmdManual.Enabled = False

    Else
        lblAuto.Value = False
        lblManual.Value = True

        cmdManual.Enabled = False
        'cmdAuto.Enabled = True
        cmdPT2.Enabled = True
        cmdPT3.Enabled = False
        cmdPT4.Enabled = False
        cmdPT5.Enabled = False

        Dim CommandData As String
        CommandData = 5    'set motor to home postion, wait
for command
        WinsockClient.SendData CommandData
    End If


End Sub
```

```
Private Sub cmdPT2_Click()
    'button animation
    cmdPT2.FaceColor = &H80FF80
    cmdPT2.CaptureColor = &H80FF80
    cmdPT2.FocusColor = &H80FF80

    'enable/disble other buttons
    cmdAuto.Enabled = False
    cmdPT2.Enabled = False
    'cmdPT3.Enabled = True
    'cmdPT4.Enabled = False
    'cmdPT5.Enabled = False

    'enable/disable timer to control arrow animation
    Timer2.Enabled = True
    Timer3.Enabled = False
    Timer4.Enabled = False
    Timer5.Enabled = False
    txtMoniter.Text = "Chico is grabbing a block from part
feeder..........."

    'send control command
    Dim CommandData As String
    CommandData = 0
    WinsockClient.SendData CommandData
End Sub

Private Sub cmdPT3_Click()
    cmdPT3.FaceColor = &H80FF80
    cmdPT3.CaptureColor = &H80FF80
    cmdPT3.FocusColor = &H80FF80

    'cmdPT2.Enabled = False
    cmdPT3.Enabled = False
    cmdAuto.Enabled = False
    'cmdPT4.Enabled = True
    'cmdPT5.Enabled = True

    Timer3.Enabled = True
    txtMoniter.Text = "Chico is moving the block to the
inspection station..........."
```

```
    Timer2.Enabled = False
    Timer4.Enabled = False
    Timer5.Enabled = False


    Dim CommandData As String
    CommandData = 1
    WinsockClient.SendData CommandData
End Sub

Private Sub cmdPT4_Click()
    cmdPT4.FaceColor = &H80FF80
    cmdPT4.CaptureColor = &H80FF80
    cmdPT4.FocusColor = &H80FF80

    'cmdPT2.Enabled = True
    'cmdPT3.Enabled = False
    cmdAuto.Enabled = False
    cmdPT4.Enabled = False
    cmdPT5.Enabled = False

    Timer4.Enabled = True
    txtMoniter.Text = "Chico is moving the rejected block
to the rejected bin..........."

    Timer2.Enabled = False
    Timer3.Enabled = False
    Timer5.Enabled = False

    Dim CommandData As String
    CommandData = 2
    WinsockClient.SendData CommandData
End Sub

Private Sub cmdPT5_Click()
    cmdPT5.FaceColor = &H80FF80
    cmdPT5.CaptureColor = &H80FF80
    cmdPT5.FocusColor = &H80FF80

    'cmdPT2.Enabled = True
    'cmdPT3.Enabled = False
    cmdAuto.Enabled = False
    cmdPT4.Enabled = False
    cmdPT5.Enabled = False

    Timer5.Enabled = True
    txtMoniter.Text = "Chico is moving the accepted block
to the accepted bin..........."

    Timer2.Enabled = False
    Timer4.Enabled = False
    Timer3.Enabled = False

    Dim CommandData As String
    CommandData = 3
    WinsockClient.SendData CommandData
End Sub



'*********************************************
'Read data from server
' winsock1--read robot response
Private Sub Winsock1_Close()
    Winsock1.Close
    End
End Sub


Private Sub Winsock1_DataArrival(ByVal bytesTotal As
Long)
    Dim buffer As String
    Dim reply As String * 6
    Dim PT2, PT3, PT4, PT5, Home, block As String
    Winsock1.GetData buffer

    txtDecimal = buffer
    txtBinary.Text =
Format(DecimalToBinary(CLng(buffer)), "000000")

    reply = DecimalToBinary(CLng(buffer))

        'mid$(y,n,m) mid$("abcde",2,1) = b
    PT2 = Mid(Format(reply, "000000"), 6, 1) ' get I0
    PT3 = Mid(Format(reply, "000000"), 5, 1) ' get I1
    PT4 = Mid(Format(reply, "000000"), 4, 1) ' get I2
    PT5 = Mid(Format(reply, "000000"), 3, 1) ' get I3
    Home = Mid(Format(reply, "000000"), 2, 1) ' get I4
    block = Mid(Format(reply, "000000"), 1, 1) ' get I5

    If PT2 = 1 Then                'Getpart done
        lblPT2.FaceColor = &HC000&
        lblPT2.CaptureColor = &HC000&
        lblPT2.FocusColor = &HC000&

        txtMoniter.Text = "Chico has finished grabbing the
block from part feeder."

        Timer2.Enabled = False

        'Dim CommandData As String
            'CommandData = 5
            'WinsockClient.SendData CommandData

    Else
      lblPT2.FaceColor = &HE0E0E0
      lblPT2.CaptureColor = &HE0E0E0
      lblPT2.FocusColor = &HE0E0E0
    End If

    If PT3 = 1 Then               'Inspect done
        lblPT3.FaceColor = &HC000&
        lblPT3.CaptureColor = &HC000&
        lblPT3.FocusColor = &HC000&

        lblBlock.Visible = True
        lblBlockStatus.Visible = True

        Timer3.Enabled = False
            If Timer4.Enabled = False Then
                txtMoniter.Text = "Chico has finished
moving the block to the inspection station."
            End If

        'CommandData = 5
        'WinsockClient.SendData CommandData
    Else
      lblPT3.FaceColor = &HE0E0E0
      lblPT3.CaptureColor = &HE0E0E0
      lblPT3.FocusColor = &HE0E0E0

      lblBlock.Visible = False
      lblBlockStatus.Visible = False
    End If

    If PT4 = 1 Then                    'solidbin done
        lblPT4.FaceColor = &HC000&
        lblPT4.CaptureColor = &HC000&
        lblPT4.FocusColor = &HC000&

        txtMoniter.Text = "Chico has finished moving the
rejected block to the rejected bin"

        'CommandData = 5
        'WinsockClient.SendData CommandData

        Timer4.Enabled = False

        lblBlock.Visible = False
        lblBlockStatus.Visible = False

    Else
      lblPT4.FaceColor = &HE0E0E0
      lblPT4.CaptureColor = &HE0E0E0
      lblPT4.FocusColor = &HE0E0E0
    End If

    If PT5 = 1 Then
        lblPT5.FaceColor = &HC000&
        lblPT5.CaptureColor = &HC000&
        lblPT5.FocusColor = &HC000&

        txtMoniter.Text = "Chico has finished moving the
accepted block to the accepted bin."

        'CommandData = 5
        'WinsockClient.SendData CommandData
        Timer5.Enabled = False

        lblBlock.Visible = False
        lblBlockStatus.Visible = False

    Else
      lblPT5.FaceColor = &HE0E0E0
      lblPT5.CaptureColor = &HE0E0E0
      lblPT5.FocusColor = &HE0E0E0
    End If

    If Home = 1 Then
        lblHome.FaceColor = &HC000&
        lblHome.CaptureColor = &HC000&
        lblHome.FocusColor = &HC000&

        txtMoniter.Text = "Chico is back to wait postion."

        'CommandData = 5
        'WinsockClient.SendData CommandData
    Else
      lblHome.FaceColor = &HE0E0E0
      lblHome.CaptureColor = &HE0E0E0
      lblHome.FocusColor = &HE0E0E0
    End If

    If block = 0 And Timer4.Enabled = False And PT3 = 1 Then
        lblBlock.Value = True
        lblBlockStatus.Caption = "Inspection passed"
    End If

    If block = 1 And PT3 = 1 Then
        lblBlock.Value = False
        lblBlockStatus.Caption = "Inspection Failed"
    End If

   'Auto mode******************************


    If lblAuto.Value = True Then
        If Home = 1 And Timer2.Enabled = False And
cmdManual.Enabled = False Then
```

```
            Dim CommandData As String
            CommandData = 5
            WinsockClient.SendData CommandData

            lblAuto.Value = False
            lblManual.Value = True
            cmdAuto.Enabled = False
        End If

        If Home = 1 And lblManual.Value = False Then
            Timer2.Enabled = True
            txtMoniter.Text = "Chico is grabbing a block
from part feeder..........."
        End If
        If PT2 = 1 Then
            Timer2.Enabled = False
            Timer3.Enabled = True
            txtMoniter.Text = "Chico is moving the block
to the inspection station..........."
        End If
        If lblBlock.Visible = True And lblBlock.Value =
True And Timer4.Enabled = False Then
            Timer5.Enabled = True
            txtMoniter.Text = "Chico is moving the
accepted block to the accepted bin..........."
        End If
        If lblBlock.Visible = True And lblBlock.Value =
False Then
            Timer4.Enabled = True
            txtMoniter.Text = "Chico is moving the
rejected block to the rejected bin..........."
        End If

    End If

    'manual mode*************************************

    If lblManual.Value = True Then     'Manual mode

        If Home <> 1 Or PT3 = 1 Or PT4 = 1 Or PT5 = 1 Or
PT2 = 1 Then
            cmdPT2.Enabled = False
            cmdPT3.Enabled = False
            cmdPT4.Enabled = False
            cmdPT5.Enabled = False
        End If

        If Home = 1 And PT3 <> 1 And PT4 <> 1 And PT5 <>
1 Then
            cmdPT2.Enabled = True
        'cmdPT3.Enabled = False
        'cmdPT4.Enabled = False
        'cmdPT5.Enabled = False
        End If

        If Timer3.Enabled = False And PT2 = 1 And PT3 <>
1 And PT4 <> 1 And PT5 <> 1 And Home <> 1 Then
        'cmdPT2.Enabled = False
            cmdPT3.Enabled = True
        'cmdPT4.Enabled = False
        'cmdPT5.Enabled = False
        End If

        If PT3 = 1 And PT2 <> 1 And PT4 <> 1 And PT5 <> 1
And Home <> 1 And block = 0 And Timer4.Enabled = False Then
            cmdPT4.Enabled = False
            cmdPT5.Enabled = True
        End If

        If PT3 = 1 And PT2 <> 1 And PT4 <> 1 And PT5 <> 1
And Home <> 1 And block = 1 And Timer5.Enabled = False Then
            cmdPT4.Enabled = True
            cmdPT5.Enabled = False
        End If

        If cmdPT2.Enabled = True Then
            cmdAuto.Enabled = True
        End If

        If lblAuto.Value = True Then
            CommandData = 4
            WinsockClient.SendData CommandData
        End If
    End If
End Sub

'*****************************************************
'convert decimal to binary

Private Function DecimalToBinary(DecimalNum As Long) As
String
    Dim tmp As String
    Dim n As Long

    n = DecimalNum

    tmp = Trim(Str(n Mod 2))
    n = n \ 2

    Do While n <> 0
        tmp = Trim(Str(n Mod 2)) & tmp
        n = n \ 2
    Loop

    DecimalToBinary = tmp
End Function
'*******************************************************
******
```