

THE UNIVERSITY OF CALGARY

A FRAMEWORK OF A WEB-BASED DISTRIBUTED CONTROL SYSTEM

by

LIAN CHEN

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES
IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE
DEGREE OF MASTER OF SCIENCE

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

CALGARY, ALBERTA

MAY 2003

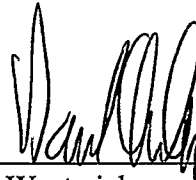
© Lian Chen 2003

**UNIVERSITY OF CALGARY
FACULTY OF GRADUATE STUDIES**

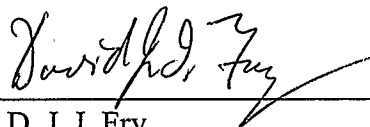
The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies for acceptance, a thesis entitled "A Framework of A Web-based Distributed Control System" submitted by Lian Chen in partial fulfilment of the requirements of the degree of Master of Science.



Supervisor, Dr. A. Eberlein
Department of Electrical and Computer Engineering



Dr. D. Westwick
Department of Electrical and Computer Engineering



Dr. D. J. I. Fry
Department of Physics and Astronomy

July 10, 2003
Date

ABSTRACT

This thesis describes a framework of a web-based distributed control system (WBDCS) and the design and implementation of its software using a multi-tier client/server architecture and distributed object technology.

The objectives of this research are (1) the exploration of a new framework for generic web-based distributed control systems, (2) the establishment of communication among distributed computing nodes, (3) the study of connectivity and controllability of on-line devices, and (4) the evaluation, discussion and implementation of time and event scheduling for real-time systems.

The prototype that was developed to validate our framework allows the communication of distributed computing nodes with each other through the Internet/intranet, it helps visualize real-time process data, and enables the authorized user to configure and monitor process status and digital devices remotely or locally using a web browser.

Testing of the prototype in a laboratory setting showed satisfactory performance as it was expected from the design.

ACKNOWLEDGMENTS

I wish to express my sincere thanks to my supervisor, professor Dr. Amin Eberlein, who gave me this excellent opportunity of being a part of this challenging work environment. His guidance and supervision made this thesis a reality.

I would also like to thank professor Dr. David Fry and professor Dr. David Westwick for kindly joining the defence committee and reviewing this thesis. Special thanks to Mr. Ed Evanik for making a lot of lab support for this research work.

Thanks also to the Department of Electrical and Computer Engineering at University of Calgary, NSERC, and AIF for providing financial support for my research.

I am grateful for my family's continuous support and encouragement throughout my study. I wish to dedicate my work and this thesis to my parents, husband and son.

TABLE OF CONTENTS

APPROVAL PAGE.....	ii
ABSTRACT.....	iii
ACKNOWLEDGMENTS	iv
TABLE OF CONTENTS.....	v
LIST OF FIGURES	x
LIST OF ABBREVIATIONS.....	xiii
1 INTRODUCTION.....	1
1.1 Real-time Distributed Systems	1
1.2 Background.....	3
1.3 Objectives	3
1.4 Approach of the Framework.....	4
1.5 Validation of the Design.....	5
1.6 Contributions of the Thesis.....	6
1.7 Organization of the Thesis.....	6
2 LITERATURE REVIEW.....	8
2.1 Introduction.....	8
2.2 Existing Distributed Control Systems (DCS).....	10
2.2.1 Large-scale DCS	10
2.2.2 Small-scale DCS	11
2.2.3 Internet-based DCS.....	14

2.3	Problems and Solutions	16
3	DISTRIBUTED OBJECT ARCHITECTURES	19
3.1	Introduction.....	19
3.2	Existing Distributed Object Technologies.....	19
3.2.1	CORBA.....	20
3.2.2	DCOM.....	20
3.2.3	RMI.....	21
3.2.4	Reasons for Choosing CORBA for WBDCS.....	21
3.3	CORBA Technology	22
3.3.1	Interface Definition Language	23
3.3.2	Object Request Broker.....	23
3.3.3	CORBA Model	24
4	FRAMEWORK OF WBDCS.....	28
4.1	Client/Server Architectures	28
4.1.1	Client and Server.....	28
4.1.2	Two-tier Architecture.....	29
4.1.3	Three-tier Architecture.....	29
4.2	Overview of WBDCS	30
4.2.1	Overview of Structure.....	31
4.2.2	Nodes	32
4.3	The Architecture of WBDCS.....	33
4.3.1	Web-based Multi-tier Approach	33

4.3.2	Distributed Object Approach	35
4.3.3	Analysis of the Approaches	35
4.3.4	Applet Security Issues.....	37
4.4	The Design of WBDCS	39
4.4.1	Functional Descriptions	39
4.4.2	Control Algorithms	46
5	REAL-TIME SCHEDULING IN WBDCS	49
5.1	Introduction.....	49
5.1.1	Task.....	49
5.1.2	Real-time Scheduling.....	50
5.2	Rate Monotonic Algorithm.....	52
5.3	Fixed Priority Scheduling Model.....	53
5.4	Real-time Scheduling in WBDCS	55
5.4.1	Priority Assignment	55
5.4.2	Interrupt.....	56
5.4.3	Time-driven Scheduling.....	57
5.4.4	Event-driven Scheduling.....	59
6	A PROTOTYPE OF WBDCS.....	60
6.1	Introduction.....	60
6.2	Client Tier.....	61
6.2.1	Control Applet	61
6.2.2	Database Applet.....	65

6.3	Middleware	67
6.3.1	CORBA.....	67
6.3.2	OMG IDL.....	67
6.4	Control Server Tier	70
6.4.1	Digital I/O Server Object.....	71
6.4.2	Database Management Server Object.....	71
6.4.3	Control Server Logger	72
6.5	Device Tier	72
6.5.1	PCI215 Card.....	72
6.5.2	JNI and DLL	74
6.5.3	Experiment Kit.....	76
6.6	Database Tier	77
6.6.1	JDBC.....	77
6.6.2	InterBase	78
6.7	Deployment.....	78
7	TESTING AND ANALYSIS.....	81
7.1	Testing	81
7.2	Analysis	82
7.3	Proposals of Test Plan	83
7.3.1	Rigid Testing on the Prototype	83
7.3.2	Virtual Plant Testing.....	85
8	CONCLUSIONS AND FUTURE WORK	87

8.1	Summary.....	87
8.2	Discussion and Future Work	89
8.2.1	Reliability and Speed	89
8.2.2	Scalability and Maintainability	90
8.2.3	Authority and Security.....	91
REFERENCES		93

LIST OF FIGURES

Figure 1.1	Real-time system.....	1
Figure 2.1	DeltaV architecture	11
Figure 2.2	Smart DCS	12
Figure 2.3	Fieldbus-based distributed architecture.....	13
Figure 2.4	New distributed architecture	13
Figure 2.5	System architecture	14
Figure 2.6	CORBA-based control system	15
Figure 2.7	Internet-based monitoring of DCS	16
Figure 3.1	CORBA model	24
Figure 4.1	Two-tier architecture.....	29
Figure 4.2	Three-tier architecture.....	30
Figure 4.3	System overview	31
Figure 4.4	Nodes in WBDCS	32
Figure 4.5	Architecture of WBDCS	34
Figure 4.6	Web-based multi-tier distributed object architecture.....	34
Figure 4.7	IIOP gateway.....	38
Figure 4.8	Use case diagram of WBDCS	40
Figure 4.9	Login use case diagram.....	41
Figure 4.10	Login activity diagram	41
Figure 4.11	Input use case diagram	42

Figure 4.12	Input activity diagram	42
Figure 4.13	Output use case diagram	43
Figure 4.14	Output activity diagram.....	43
Figure 4.15	Query use case diagram	44
Figure 4.16	Query activity diagram.....	44
Figure 4.17	Configuration use case diagram	45
Figure 4.18	Configuration activity diagram	45
Figure 4.19	Feedback control loop	47
Figure 4.20	PID control response	47
Figure 4.21	On-off control response.....	48
Figure 5.1	Event-driven scheduler.....	54
Figure 5.2	Timer-driven scheduler	55
Figure 5.3	Periodic and aperiodic interrupts	56
Figure 5.4	Time-driven scheduling	58
Figure 5.5	Event-driven scheduling.....	59
Figure 6.1	Architecture of the prototype	60
Figure 6.2	Configuration panel.....	63
Figure 6.3	Input panel.....	64
Figure 6.4	Output panel	65
Figure 6.5	Database applet	66
Figure 6.6	IDL file.....	69
Figure 6.7	Control server.....	70

Figure 6.8	JNI and DLL	75
Figure 6.9	Display circuit	76
Figure 6.10	Signal circuit	77
Figure 6.11	Homepage	79
Figure 6.12	Control applet page	79
Figure 6.13	Database applet page.....	80
Figure 7.1	Performance testing.....	84
Figure 7.2	Virtual plant	85

LIST OF ABBREVIATIONS

API	Application Programming Interface
COM	Component Object Model
CORBA	Common Object Request Broker Architecture
DBMS	Database Management System
DCOM	Distributed Component Object Model
DCS	Distributed Control System
DII	Dynamic Invocation Interface
DLL	Dynamic Link Library
DSI	Dynamic Skeleton Interface
GIOP	General Inter-ORB Protocol
GUI	Graphical User Interface
I/O	Input/Output
IDL	Interface Definition Language
IIOP	Internet Inter-ORB Protocol
IOR	Interoperable Object Reference
IP	Internet Protocol
JDBC	Java Database Connectivity
JDK	Java Development Kit
JNI	Java Native Interface
JRMP	Java Remote Method Protocol

JVM	Java Virtual Machine
LabVIEW	Laboratory Virtual Instrument Engineering Workbench
LAN	Local Area Network
MIS	Management Information System
OA	Object Adapter
OLE	Object Linking and Embedding
OMG	Object Management Group
OPC	OLE for Process Control
ORB	Object Request Broker
ORPC	Object Remote Procedure Call
PLC	Programmable Logic Controller
POA	Portable Object Adapter
PPI	Programmable Peripheral Interface
RMI	Remote Method Invocation
RPC	Remote Procedure Call
SCADA	Supervisory Control and Data Acquisition System
SQL	Structured Query Language
SSL	Secure Socket Layer
TCP	Transmission Control Protocol
VI	Virtual Instrument
WAN	Wide Area Network
WBDCS	Web-Based Distributed Control System

1 INTRODUCTION

1.1 *Real-time Distributed Systems*

A real-time system [Bennett94] is any system in which the time at which the output is produced is significant. This is usually because the input corresponds to some changes in the physical world, and the output has to relate to the same changes. The lag between input time and output time must be sufficiently small for acceptable timeliness. Typically real-time systems consist of controlling subsystems (computer controller) and controlled subsystems (physical environment). The interactions between the two subsystems are described by three operations: sampling, processing, and responding (see Figure 1.1). The computer subsystem continually samples data from the physical environment. Sampled data is processed immediately by the computer subsystem, and a proper response is sent to the physical environment. All three operations must be performed within specified times, which are the time constraints.

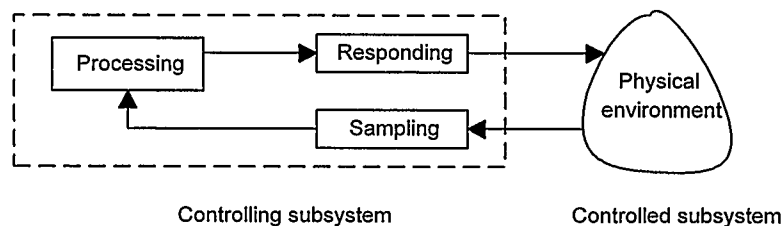


Figure 1.1 Real-time system

Real-time systems can be classified into hard real-time systems and soft real-time systems depending on the consequences of timing constraint violations. Hard real-time systems must meet their time constraints to avoid disastrous consequences, such as flight control systems, chemical process *control systems, robot control systems, and telephone

switching systems. Soft real-time systems are still considered functionally correct if time constraints are not seriously violated, such as remote data acquisition systems, airline ticket reservation systems, and automatic teller machines.

A distributed system [Tsai96] has many processes running on different processors working toward specific functional requirements. The distributed processes are coordinated by inter-process communication and synchronization. The processors may be dispersed geographically or located at one site or in one machine. Distributed systems can be classified as homogeneous or heterogeneous. The nodes in homogeneous distributed systems have the same hardware configurations and supporting software. In contrast, heterogeneous systems have different hardware architectures and/or supporting software. Distributed systems can also be classified as centralized or decentralized. Centralized distributed systems have master-slave or server-client relationships between their distinct computing nodes, and decentralized distributed systems have autonomous computing nodes.

A real-time distributed control system is a real-time distributed system, which plays an important role in many industries. Usually these systems are used to monitor and control industrial process plants, telecommunication systems, manufacturing production lines, robots, etc.

The development of real-time systems is much more complicated than that of non real-time ones. This is mainly because they have to meet strict timing constraints. The system must notice and respond to changes in the physical environment in a timely manner, usually on the order of milliseconds. The basic requirement for such systems is timeliness, which sets real-time systems apart from conventional computing systems.

1.2 Background

With the advent of the Internet, companies have realized that a whole new market has opened up to be exploited. Nowadays, industrial network applications are increasingly based on the Internet, TCP/IP (Transmission Control Protocol / Internet Protocol), and World Wide Web technologies. However, web-based distributed control systems have not yet been extensively exploited due to difficulties in coping with the large amount of required information, the lack of standardized approaches to communicate with on-line devices, and the unreliability of data transmission on the Internet.

Emerging and maturing infrastructures and technologies for communication and distributed object management have enabled the integration of heterogeneous applications that execute on distributed nodes throughout the Internet/intranet. The fundamental technologies are Microsoft's Distributed Component Object Model (DCOM), Object Management Group's (OMG) Common Object Request Broker Architecture (CORBA) and JavaSoft's Java Remote Method Invocation (Java RMI), all of which support distributed computing, concurrency access, and object communication across the Internet/intranet. Based on these technologies, a standard-based framework, Web-Based Distributed Control System (WBDCS) [Chen02, 03], is proposed in this research.

1.3 Objectives

This research intends to investigate technologies and standards that can be adopted to construct a framework of a web-based distributed control system, which takes advantage of Internet/intranet technologies and the World Wide Web to control and monitor

industrial processes. The major envisioned capabilities include distributed computing nodes communicating with each other via the Internet/intranet, visualizing real-time processes and data, and controlling and monitoring industrial process status. In other words, the supervision and operation of industrial processes can be conducted remotely or locally using a web browser. An Internet or intranet, as communication media, is integrated with the devices to control and supervise the operations of industrial processes. Overall, the objectives of this research are to explore the feasibility of a new framework and design an architecture for generic web-based distributed control systems, to establish communication and synchronization among distributed computing nodes, to study connectivity and controllability of on-line devices and their interactions with users, and to conduct a time and event scheduling studies for a real-time distributed control system.

1.4 Approach of the Framework

WBDCS is designed to adopt a web-based multi-tier client/server approach and distributed object computing approach. The web-based multi-tier client/server approach means that WBDCS is a web application with a multi-tier client/server software architecture. It is a thin client web application using only HTML pages and applets as graphical user interfaces (GUIs), which are downloadable from a web server in the system. As a result of this approach, there is no need to install any client applications on client machines. The distributed object computing approach used in WBDCS uses CORBA as middleware to facilitate the communication and interaction between distributed objects via the Internet/intranet.

WBDCS is designed as a decentralized distributed system because there is no node that plays a central role in the system even though it uses a client/server approach. A node can be a client if it requests services from other nodes, but it can also be a server when it provides services to others. CORBA [Vinoski97] enables WBDCS to be a heterogeneous distributed system because CORBA is a distributed framework designed to support heterogeneous systems. Nodes in the systems may differ not only by the hardware (CPU and memory) they use, but also by their operating system and programming language.

A real-time system is a system whose correctness depends not only on its logical behaviour but also on timing constraints [Tsai96]. Because of the importance of timeliness in distributed control systems, emphasis has been put on real-time scheduling for WBDCS. To avoid delays through the Internet, improve reliability of communication, and speed up real-time response, time-critical parts of the system can be localized within the nodes. Non time-critical information is accessible and manageable over the Internet/intranet. From this point of view, WBDCS can be considered as two parallel subsystems: one is a distributed control subsystem - a hard real-time system, the other is a centralized supervisory subsystem - a soft real-time system.

1.5 Validation of the Design

A prototype of WBDCS has been implemented and allows the remote control of digital devices. Resources and devices are dispersed across the Internet/intranet in multiple computing nodes. An authorized user can access local or remote nodes by using a web browser to perform control functions, such as real-time data acquisition and visualization,

and remote device monitoring and configuration. Testing showed satisfactory performance as it was expected from the design.

1.6 Contributions of the Thesis

Key contributions of this thesis are as follows:

- Proposed and designed a standard-based framework for a generic web-based distributed control system, which is a web application using a multi-tier client/server software architecture and distributed computing technology (CORBA). It provides a solution to the objectives of designing a portable, scalable, and maintainable web-based distributed application.
- Defined time-driven and event-driven scheduling behaviours in WBDCS based on the theory and models proposed by [Liu73] and [Katcher93]. A solution was proposed that can cope with potentially lower reliability and speed for data communication on the Internet by processing hard real-time tasks within a node to ensure timeliness, while soft real-time tasks can be processed and distributed across the Internet/intranet.
- Implemented a prototype to verify the architecture, performance, scalability and schedulability of the framework and to evaluate the connectivity and controllability of online devices.

1.7 Organization of the Thesis

This thesis is organized in 8 chapters. Chapter 1 is a brief introduction into the research pursued in this thesis. Chapter 2 is an overview of current industrial real-time distributed control systems and their characteristics. Chapter 3 describes different distributed object

architectures that enable distributed computing across the Internet. The architectural design and structure of WBDCS is detailed in Chapter 4. Chapter 5 discusses time and event scheduling and priority handling as introduced in WBDCS. Chapter 6 presents the implementation technologies used in a prototype of WBDCS. Chapter 7 analyzes the testing results on current prototype and proposes some further test plans for WBDCS. Finally, in Chapter 8, a summary of this research and its future direction are provided.

2 LITERATURE REVIEW

2.1 *Introduction*

Most real-time systems are inherently distributed and dynamic. In order to reflect these properties the computer-based control systems, which monitor, control, or simulate industrial processes must provide adequate means to cope with time, concurrency, and decentralization. In addition, control systems should still function and be available during maintenance, or provide fail-safe behaviour. The use of intelligent sensors and actuators is advisable to reduce data and control traffic and improve the robustness of the control system. Due to advances in technology, such sensors and actuators can now be built based on cheap and powerful micro-controllers. Traditional system structures based on static and hierarchical control do not suffice any longer; instead a new generation of computer-based control systems is needed based on a cooperation-oriented paradigm.

Industrial control systems can be divided into two groups: traditional Distributed Control Systems (DCS) and Programmable Logic Controllers (PLC). Current DCS systems use central Supervisory Control and Data Acquisition (SCADA) systems that communicate via local networks with numerous controllers, instruments, sensors and actuators. For many machine control and production processes, systems have generally been designed using PLCs. Traditionally, systems have been developed as large monolithic software packages that are generally difficult to reuse in new applications and are notably difficult to integrate with each other. Data and functionality of one application are not readily available to other applications, even if the applications are written in the same programming language and run on the same machine [Lewis97].

The emergence of standards in industrial communications such as Fieldbus allows different types of instruments and control devices to interoperate. In recent years, the differences between DCS and PLC based systems are starting to disappear, and DCS instruments and PLCs are beginning to offer similar functionality [Lewis97].

Control systems have been moved from centralized to distributed control, because the real-world problems are distributed, as in telecommunication, traffic control or building automation, and because the implementation has to be flexible and reliable. There is currently growing interest in new technologies and architectures for exploring the next generation technologies for distributed control systems. To achieve the higher level of integration, and to enable the creation of flexible systems will require a completely new approach to control software architecture design, which is based on the integration of distributed objects. There are already several well-advanced software technologies in this area. CORBA is a standard for designing distributed systems. OPC (OLE for Process Control) based on Microsoft's OLE (Object Linking and Embedding), COM (Component Object Model), and DCOM technologies allow software components to interoperate regardless of their location [Lewis97]. Internet and World Wide Web technologies are also considered for the development of software components in industrial process control systems.

In the following subsections, we will introduce the concepts, trends, and development of distributed control system technologies.

2.2 Existing Distributed Control Systems (DCS)

2.2.1 Large-scale DCS

Distributed control systems are computer-based control systems in which the main components are located in different places. These components interact with each other via a local area network (LAN).

There are off-the-shelf DCSs designed and manufactured by Honeywell [Honeywell], Siemens [Siemens], and Emerson Process Management [Emerson], etc. Each vendor has its own method and communication protocol for providing the information and control. Vendors also provide the required control for modern industrial systems and some can provide communication links to network systems such as Controlnet, Ethernet, Devicenet, Profibus, ASI bus, Foundation Fieldbus, Data Highway/Remote I/O, Hart Protocol, and Modbus RTU [Decker01]. These DCSs are designed for process controls in petrochemical, pulp, food, beverage, and mining industries.

A typical example is DeltaV Digital Automation System [Emerson] as illustrated in Figure 2.1. The field devices, such as sensors, valves, motors, and pumps, are connected with different digital communication buses like Fieldbus, HART, and DeviceNet. The linking devices are gateways or bridges between the field buses and high speed Ethernet. Operator stations are Windows workstations and server-based PCs. Overall, the DeltaV digital automation system is a DCS based on Windows workstations and server-based PCs, Ethernet technology, and bus standards.

The DeltaV suite of engineering tools handles configuration management both locally and remotely for all aspects of the DeltaV system and intelligent field devices. A global

and centralized configuration database coordinates control strategies, process graphics, history, events, and change management. All DeltaV hardware is automatically recognized as it is plugged in. The DeltaV operation software provides an easy-to-use environment for process operations and information access.

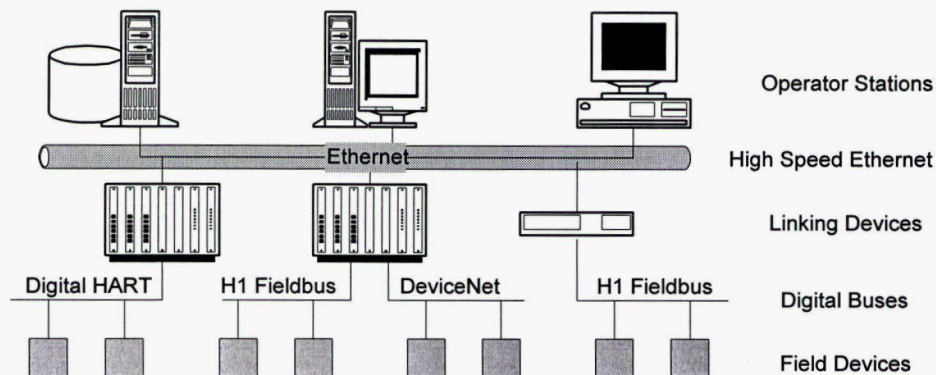


Figure 2.1 DeltaV architecture

All operations applications are fully remotely accessible anywhere on an Ethernet network or via modem. The process history view provides continuous trend, event views and batch views to intuitively present these different types of historical information. By using digital plant architecture (PantWeb) and plant-wide asset optimization software (AMS), the DeltaV provides easy access to vital device information for calibration, configuration, devices audit trail, and advanced diagnostics for predictive maintenance.

Large-scale DCSs, like DeltaV, are usually used to control almost all processes in a big plant.

2.2.2 Small-scale DCS

A small smart distributed control system [Yang02] is shown in Figure 2.2. The system includes an industrial PC, several controllers, and a communication interface using RS-

485. The PC is an operating station that is used to configure function blocks and monitor the process. The controller is used to sample data and execute control algorithms. Communication between the PC and the controllers is implemented in a master-slave type broadcasting on the communication network. The slave nodes are not allowed to transmit data without a request from the master, and do not directly communicate with each other. When a slave needs to send a message to another slave, the message has to be sent to the master and then the master forwards the message to the receiver.

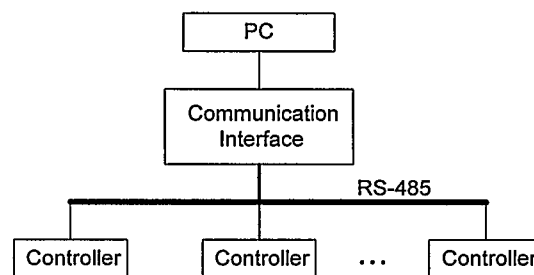


Figure 2.2 Smart DCS

Marti et al. proposed an integrated approach to real-time distributed control systems over Fieldbus [Marti01]. A control loop is implemented in a distributed architecture, with three nodes: a sensor node, a controller node and an actuator node that communicate with each other across a fieldbus communication network as depicted in Figure 2.3. The sensor node periodically samples the process and sends the data to the controller node. The controller node executes a control algorithm and sends the output to the actuator node.

Marti et al. found that the design shown in Figure 2.3 may decrease the performance of the control system and even might cause instability of the control system because of time delays on the network. In order to meet the timing constraints of the real-time system,

they redesigned the distributed architecture by combining the controller and actuator into one node (see Figure 2.4) to reduce time delays between them. The latter design has been successfully implemented in a real-time fieldbus-based distributed control system. They concluded that the successful design and implementation of real-time distributed control systems requires an appropriate integration of the control systems, real-time systems, and communication systems.

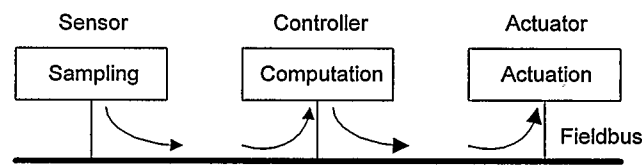


Figure 2.3 Fieldbus-based distributed architecture

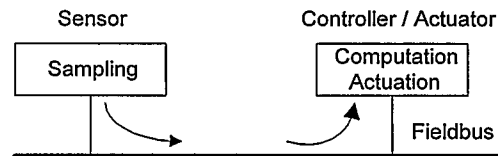


Figure 2.4 New distributed architecture

A distributed control system is introduced in [Stolen99] for supervising a power supply system of a telecommunication system as depicted in Figure 2.5. The DCS consists of a main controller, several distributed units, and a communication network using multi-drop RS-485 bus with a proprietary communication protocol. The core of the supervision system is the main controller, which is a micro-controller based unit with communication ports (RS-232, RS-485) and a user interface. The RS-232 port is intended for higher-level supervision, typically for remote connection via modem or computer network. The RS-485 ports are used for communication between the main controller and distributed units.

Each distributed unit contains a small micro-controller for data acquisition and control. Each unit has a unique address. They are connected to the bus and receive both serial communication and power from the bus. The main controller collects information from distributed units using the master-slave network and sends the messages to all distributed units, but only the right unit with the correct address will answer.

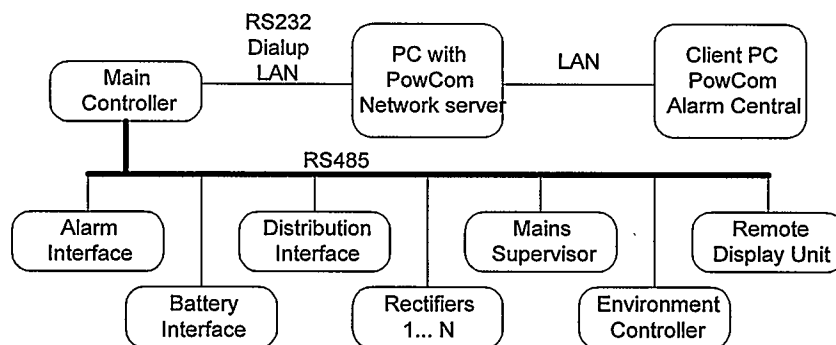


Figure 2.5 System architecture

2.2.3 Internet-based DCS

DCOM, CORBA, and Java RMI are well known middleware architectures based on object-oriented methodologies and have been extensively used in commercial and military distributed information systems. Since they support communication protocols that are transparent to operating systems, users can design and manage the whole system without paying too much attention to the data transmission between remote devices [Kang02]. In Kang's paper, CORBA-, DCOM-, and Tspace-based distributed control systems are designed and implemented for performance analysis using I/O object, control object, broadcasting service and event service. When using middleware as a communication bus for objects, the delay may be greater than the one introduced by the Fieldbus, since DCOM, CORBA, and Java RMI have not been designed for the physical

device level but for integrating and linking of software components at higher levels. Therefore, middleware-based systems are likely to be applied to soft real-time areas like data monitoring systems [Kang02]. According to the results of performance analysis, the CORBA-based DCS had a shorter response time than DCOM- and Tspace-based systems, thus CORBA is a more suitable middleware for distributed control systems as illustrated in Figure 2.6.

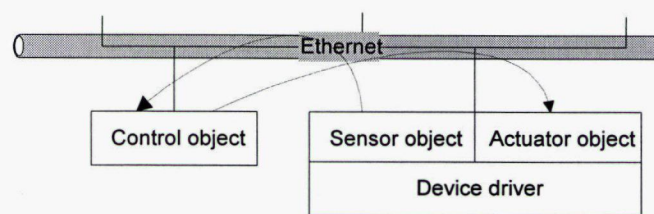


Figure 2.6 CORBA-based control system

An Internet-based system is developed in [Tan01, Tan02] to allow monitoring of process variables with a distributed control system. The core of the web-based DCS is a desktop computer monitoring the status and performance of processes and equipment that are wired to the computer. The DCS is based on the ADAM series [Tan02] of intelligent sensor-to-computer interface modules with built-in microprocessors. The necessary hardware used in the development of a virtual laboratory at the server end consists of ADAM network I/O modules and sensors connected to a PC with a RS-232 link as shown in Figure 2.7.

The overall system involves two subsystems, a client-server application and a backend LabVIEW VI (Virtual Instrument). The front end involves a client side applet that presents the user with a GUI displayed in a web browser. Through the browser, users are able to access the status of the sensors over the Internet. The backend control software, or

VI, is written in Laboratory Virtual Instrument Engineering Workbench LabVIEW. The VI supplies information about sensors to the client applet via a server program running on the host computer. In this system, sockets are used for communication between the server and client based on the TCP/IP protocol.

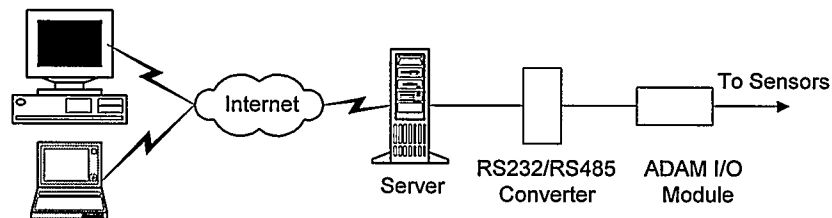


Figure 2.7 Internet-based monitoring of DCS

2.3 Problems and Solutions

The use of microprocessors has proliferated in distributed control systems, but they often do not work together. Application software should readily communicate with field devices as well as other applications, but this is not often the case. Making these systems work together is one of the most pressing needs of process industries.

In the absence of any standard, vendors have developed proprietary hardware and software solutions. The process control and information systems that are on the market today have proprietary techniques, interfaces, and APIs in order to exchange information. The cost of integrating the different systems and the long-term maintenance and support of such a heterogeneous environment can be significant.

The solution is to adopt a standard that provides real plug-and-play software technology for process control and factory automation where every system, every device and every

driver can freely connect and communicate. Having such a standard would allow open enterprise-wide communication between systems and devices, from plant floor to Management Information System (MIS).

OPC [OPC] and CORBA [OMG] are two industrial standards for integrating software that have been written on different systems and in different languages. OPC is an industry standard created with the collaboration of a number of leading worldwide automation and hardware/software suppliers working in cooperation with Microsoft. The organization that manages this standard is the OPC Foundation. The Foundation has over 290 members from around the world, including nearly all of the world's major providers of instrumentation and process control systems.

CORBA is another industrial standard defined by OMG for distributed object systems. The OMG founded in 1989 is the world's largest computer industry consortium, with over 800 members, including almost all the major vendors and developers of distributed object technology, including platform, database, and application vendors as well as software tool and corporate developers. The goal of the OMG is to develop, adopt, and promote a standard architectural framework for the development and deployment of distributed object-oriented applications in heterogeneous environments.

OPC [Janke99] defined a standard set of objects, interfaces and methods for use in process control and manufacturing automation applications to facilitate interoperability based on Microsoft's OLE, COM and DCOM technologies.

Although OPC is widely used in automation industries, the core technologies used in OPC is COM and DCOM. According to the performance analysis in [Kang02], DCOM is not suitable for distributed control systems due to longer delay time between objects as

compared to CORBA. Although Microsoft claims to have improved the object model to increase performance, benchmark tests indicate otherwise. Benchmarks incorporating network communication and simple method invocation show DCOM to be almost 20% slower than CORBA [Lewandowski98].

For these reasons, CORBA was chosen as the distributed object architecture for the synchronization and communication between client side and server side applications in the design of WBDCS in this thesis.

3 DISTRIBUTED OBJECT ARCHITECTURES

3.1 Introduction

Business and industrial applications have evolved over a period of time from a relatively rigid monolithic architecture to an extremely flexible, highly decentralized distributed one. Application architectures have offered increasing robustness and reliability because of the development of distributed object computing, which extends an object-oriented programming system by allowing objects to be distributed across a heterogeneous network. These objects may be dispersed on different computers throughout a network, but they appear as though they were local to the application.

There are various reasons for developing applications in a distributed manner. Some applications must execute on multiple computers because the data that the application accesses exist on multiple computers. Some applications execute on multiple computers in order to take advantage of multiple processors computing in parallel to solve special problems. Some applications execute on multiple computers because users of the application are dispersed and interact with each other via the network. A distributed application can take advantage of the scalability and heterogeneity of the distributed object technologies.

3.2 Existing Distributed Object Technologies

Three of the most popular distributed object paradigms are Microsoft's DCOM, OMG's CORBA and JavaSoft's Java RMI. We will briefly introduce each of them in the following subsections.

3.2.1 *CORBA*

CORBA relies on an Internet Inter-ORB Protocol (IIOP) for remote objects invocation. The core of CORBA is the Object Request Broker (ORB) that acts as an object bus over which objects transparently interact with other objects located locally or remotely [Vinoski97]. Each CORBA server object has an interface and exposes a set of methods. To request a service, a CORBA client acquires an object reference to a CORBA server object. The client can make method calls to the object reference as if the CORBA server object resided in the client's address space. The ORB is responsible for all mechanisms to find the object's implementation, prepare it to receive the requests, communicate the requests to it and carry the reply back to the client. A CORBA object interacts with the ORB through either an Object Adapter (OA) or an ORB interface.

3.2.2 *DCOM*

DCOM is a distributed extension to COM. DCOM relies on a protocol called Object Remote Procedure Call (ORPC) to support remote objects. A DCOM server object can support multiple interfaces each representing a different view or behaviour of the object. A DCOM client interacts with a DCOM server object by acquiring a pointer to one of the server object's interfaces and invoking the methods through that pointer, as if the server object resided in the client's address space. Since the COM specification is at the binary level, it allows DCOM components to be written in different programming languages such as C++, Java, Delphi, and Visual Basic. DCOM can be used on a platform as long as it supports COM services. DCOM is heavily used on the Windows platform.

3.2.3 *RMI*

Java RMI is based on the Java Remote Method Protocol (JRMP). Java relies heavily on Java object serialization, which allows objects to be marshaled as a stream. Since Java object serialization is specific to Java, both the RMI server and the client objects have to be written in Java. Each RMI server object defines an interface that can be used to access the server object outside of the current Java Virtual Machine (JVM) and on another machine's JVM. The interface exposes a set of methods that are indicative of the services offered by the server object. For a client to locate a server object for the first time, RMI uses a naming mechanism called an RMIRegistry that runs on the server machine and holds information about available server objects. An RMI client acquires an object reference to an RMI server object by looking for a server object reference and invoking methods on the server object as if the RMI server object resided in the client's address space. RMI can be used on diverse operating system platforms as long as these platforms are JVM enabled.

3.2.4 *Reasons for Choosing CORBA for WBDCS*

CORBA has several primary benefits, among them language-, vendor-, and operating-system independence. CORBA ORBs are available on every major operating system in use today. CORBA ORBs also exist to bind to a wide variety of languages including C++, Ada, COBOL, Smalltalk, and Java. Using IIOP, a CORBA ORB developed by one vendor can retrieve and manipulate objects obtained from a remote ORB developed by another vendor.

DCOM, like CORBA, is a language-independent distributed object technology. Since DCOM runs mainly on Microsoft operating systems, DCOM is less portable than CORBA. Several barriers to porting DCOM to other platforms exist. For example, DCOM relies on the Windows NT security model to provide system security; it is unclear what will provide security when DCOM is used on other platforms. In contrast, CORBA uses a universal security mechanism that will work on all platforms, regardless of operating-system-level security [Lewandowski98].

Unlike CORBA and DCOM, Java RMI is a language-dependent distributed object technology. It is most suitable when a system purely consists of Java objects that need to communicate on a distributed system. In our research, we propose a web-based distributed control system framework that is independent of programming languages and operating system platforms. Therefore, both Java RMI and DCOM are not suitable solutions.

The primary goal of CORBA is to provide software developers with a means for developing systems that are not reliant on a single operating system and/or programming language. Although a number of different technologies exist for building distributed client/server applications on the web, CORBA has made itself the most generally applicable choice for our WBDCS with its features and broad industry support. In the following section, we will introduce CORBA in more detail.

3.3 *CORBA Technology*

CORBA is a standard defined by OMG for distributed object systems. The goal of the OMG is to develop, adopt, and promote a standard architectural framework for the

development and deployment of distributed object-oriented applications in heterogeneous environments.

3.3.1 Interface Definition Language

The basic CORBA paradigm is to support requests for services provided by a distributed object. Services that an object provides are given by its interfaces defined by OMG's Interface Definition Language (IDL).

IDL is a declarative language for defining interfaces of CORBA objects. Its syntax is similar to C and C++, but cannot directly be compiled into a binary program. Instead, IDL is intended to be an intermediary language to define the interfaces that a client will use and a server will implement. IDL is best thought of as a “contract” between the system that makes use of an object and a system that implements an object.

IDL declarations are compiled with an IDL compiler and converted to their associated representations in the target programming languages according to the standard language mapping. For example, *idlj* within J2SE 1.3 is an IDL-to-Java compiler that automatically maps an IDL file to the Java programming language and generates all Java language stub and skeleton source files, along with the infrastructure code for connecting to the ORB.

3.3.2 Object Request Broker

A core in CORBA is the Object Request Broker (ORB), which is an object bus (software bus) for connecting objects between remote address spaces. The ORB's responsibilities are establishing a communication channel between a client and a remote server object, marshaling the parameters sent by the client or server object to a network, unmarshaling the parameters received by the client or server object from the network, and managing

concurrency of simultaneous requests from multiple clients. In short, ORB locates the remote object on the network, passes the request to the object, waits for the results and sends those results back to the client.

3.3.3 CORBA Model

Figure 3.1 illustrates the components in CORBA [Brose01]. All of them collaborate to provide the portability, interoperability, and transparency of CORBA. Each component in the model is described briefly in the following.

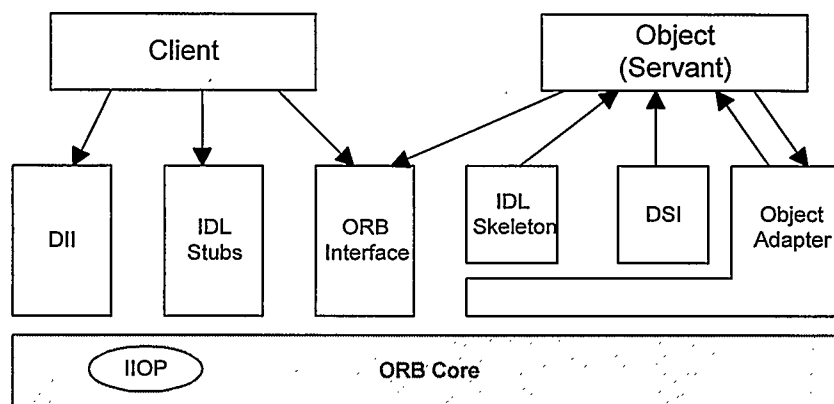


Figure 3.1 CORBA model

- **Client**

A client is a program that uses CORBA object references to invoke the operations on the objects. Objects can be remote or local to the client. Accessing a remote object is as simple as calling a local object because a client has a local reference to the remote object.

- **Object**

An object is an instance of an IDL interface. The object is identified by an object reference, which uniquely names the instance across servers. An object ID associates an object with its servant implementation, and is unique within the scope of an object

adapter. An object has one or more servants associated with it for implementing the interface.

- Servant

A servant implements the operations defined in an IDL interface. In an object-oriented programming language, servants are implemented using one or more objects. A client never interacts with a servant directly, but always through an object.

- ORB Core

The ORB core provides a mechanism for transparently communicating client requests to target object implementations. The ORB core simplifies distributed programming by decoupling the client from the details of the method invocations. This makes client requests appear like local procedure calls.

When a client invokes an operation on an object, the ORB core is responsible for delivering the request to the object and returning a response to the client. For a remote object, the ORB core communicates through IIOP, which runs on top of TCP/IP. Usually the ORB core is implemented as a run-time library linked into both client and server applications.

- ORB Interface

An ORB is a logical entity that may be implemented in various ways (such as one or more processes or a set of libraries). To decouple applications from implementation details, the CORBA specification defines an abstract interface for an ORB. This interface provides standard operations, which initialize and shutdown the ORB, convert object references to strings and vice versa, and create argument lists for requests made through the Dynamic Invocation Interface (DII).

- IDL Stub and IDL Skeleton

An IDL stub, generated by an IDL compiler, is a client-side object that makes a particular CORBA server interface available to a client. An IDL skeleton, which is also generated by an IDL compiler, is a server-side object that provides the framework on which the server implementation code for a particular interface is built. Client stub and server skeleton serve as a sort of glue that connects language independent IDL interface specifications to language specific implementation code.

- Dynamic Invocation Interface (DII)

This interface allows a client to access the underlying request mechanisms provided by an ORB. Applications use the DII to issue requests to objects without requiring IDL interface-specific stubs to be linked in. Unlike IDL stubs, which only allow Remote Procedure Call (RPC) requests, the DII also allows clients to make non-blocking deferred synchronous (separate send and receive operations) and one-way (send only) calls.

- Dynamic Skeleton Interface (DSI)

This is the server side's interface similar to the client side's DII. The DSI allows an ORB to deliver requests to an object implementation that does not have compile-time knowledge of the type of the object. The client making the request has no idea whether the implementation is using the type-specific IDL skeletons or is using the dynamic skeletons.

- Portable Object Adapter (POA)

The CORBA specification defines the concept of an object adapter. It is a framework for implementing CORBA objects and provides a comprehensive set of interfaces for managing object references and servants. The code written using the POA [OMG97]

interfaces is now portable across ORB implementations and has the same semantics in every ORB that is compliant to CORBA 2.2 or above [Brose01]. According to the CORBA specification, the purpose of a POA is to dispatch incoming invocation requests to the correct servant. It associates a servant with objects, de-multiplexes incoming requests to the servant, and collaborates with the IDL skeleton to dispatch the appropriate operation call on that servant.

- Internet Inter-ORB Protocol (IIOP)

The Internet Inter-ORB Protocol is a TCP/IP implementation of the General Inter-ORB Protocol (GIOP). The GIOP specification defines how ORBs communicate, including how messages are sent, how byte ordering is done for integers and floating point numbers, and how parameters are marshaled for remote object invocations.

With CORBA it is possible to build a client application using one vendor's ORB and IDL compiler, build a server or object implementation with a second vendor's ORB and IDL compiler, and create a set of common services for both client and server with yet a third vendor's ORB and IDL compiler. IIOP allows each of the three different vendor's products to communicate with each other using a standard set of protocol semantics [Zukowski98].

4 FRAMEWORK OF WBDCS

4.1 *Client/Server Architectures*

In general, web applications have either two-tier or three-tier client/server architectures. The two-tier architecture was developed in the 1980s from the file server software architecture design. Its intention is to improve usability by supporting a form-based user interface. It also improves flexibility and scalability by allocating the two tiers over the computer network. The three-tier (multi-tier) architecture emerged in the 1990s, with a middle tier in-between the user interface and the data management server. This middle tier provides process management and is the place where the business logic and rules are executed. Compared with the two-tier architecture, the multi-tier architecture increases the scalability and flexibility of web applications.

4.1.1 *Client and Server*

Generally, computers on a network can be categorized into two types: clients and servers. Typically, a client is an application that runs on a personal computer or workstation and relies on a server to perform some operations. A server is a computer or device on a network that manages network resources and provides services. For example, a file server is a computer dedicated to storing files. Any user on the network can store files onto the server. A print server is a computer that manages one or more printers, and a network server is a computer that manages network traffic. This means, machines that provide services to other machines are servers. The machines that connect to those services are

clients. For instance, a database server accepts requests for data from clients and returns the results to the clients. The clients manipulate the data and present results to the user.

4.1.2 Two-tier Architecture

In a two-tier architecture of software systems (see Figure 4.1) server software runs on a large server machine. Client machines connect to the server via a network and make requests of the server as necessary. In this approach, each client machine needs client software installed locally. It works well in relatively homogeneous environments, where application logics (business rules) do not change very often.

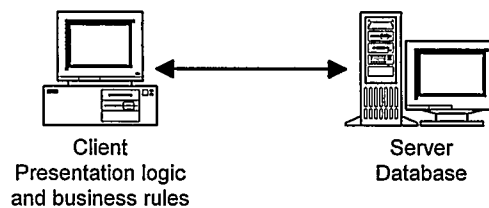


Figure 4.1 Two-tier architecture

The two-tier architecture improves flexibility and scalability by distributing the two tiers over the network. However, there are obvious limitations with the two-tier software architecture. For example, the client requires a custom application to be written that then needs to be deployed on every client machine. Since the presentation logic and business rules are usually located in the client application, even the smallest change to an application might require a complete rollout to the entire system.

4.1.3 Three-tier Architecture

The three-tier architecture consists of three well-defined and separate processes, each running on a different platform as shown in Figure 4.2. The first tier is referred to as the

user interface, which runs on the user's computer (the client). The middle tier consists of the application or business logic, which runs on a server and is often called application server. The other tier contains the data that is needed for the application, which is usually a database management system (DBMS) that stores the data required by the middle tier. This tier runs on another server called the database server.

The three-tier design has many advantages over the traditional two-tier system. For example, separating presentation logic from business rules makes it easier to modify or replace one tier without affecting the others.

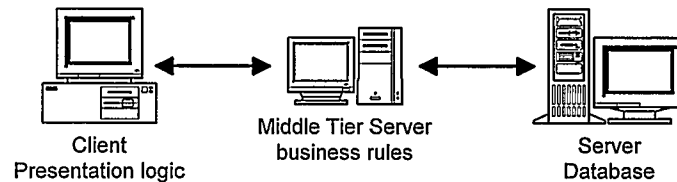


Figure 4.2 Three-tier architecture

The three-tier architecture has been used successfully since the early 1990s in commercial and military distributed client/server environments, where distributed information computing is required in a heterogeneous environment.

4.2 Overview of WBDCS

WBDCS is a multi-tier web application using CORBA as a distributed object framework. This means that it is designed and implemented using a multi-tier client/server architecture with distributed object technology. Unlike many other web applications, WBDCS is a specific web application for industrial process control and production monitoring. Therefore, the architecture of WBDCS is different from those of generic web

applications even though it adopts some of the existing technologies, such as multi-tier client/server architectures, and distributed object technologies.

4.2.1 Overview of Structure

An overview of the application architecture is shown in Figure 4.3. WBDCS is viewed as a collection of computer nodes that are communicating and cooperating to reach a common goal. The nodes in WBDCS are geographically dispersed across the Internet/intranet. Nodes can be homogeneous or heterogeneous. We adopted the heterogeneous architecture, because nodes in WBDCS may have different hardware architectures and software configurations, such as PCs running Windows and Sun workstation running Solaris. The devices that are connected to nodes may be different such as controllers, data acquisition systems, radio remote control devices, DCS, and DBMS. A homogeneous system can be considered a special case of the heterogeneous systems.

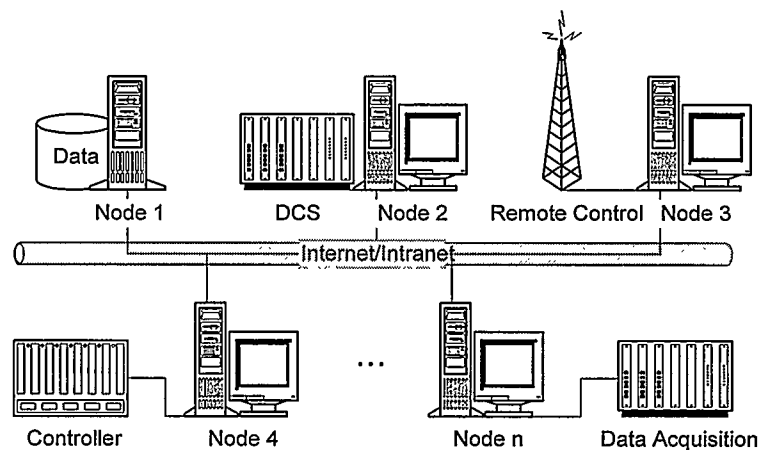


Figure 4.3 System overview

The system also allows users to integrate their existing applications, such as PLC, DCS, into a web-enabled distributed system by wrapping them with CORBA objects. These objects can then make calls to legacy systems and expose them to the Internet/intranet.

4.2.2 Nodes

The nodes in WBDCS are PCs and/or workstations connected to one or more devices. Each node may have different hardware and software configurations with different devices connected. This allows WBDCS to be a flexible heterogeneous system. All nodes in WBDCS can communicate with each other across the Internet/intranet. A node can be a client when it is sending a request to another node; on the other hand, it can also be a server when it provides a service to other nodes in the system. A client node does not need to install any client application in order to be able to access any other nodes because the system is designed using applets as GUIs that are automatically downloaded from a web server in WBDCS.

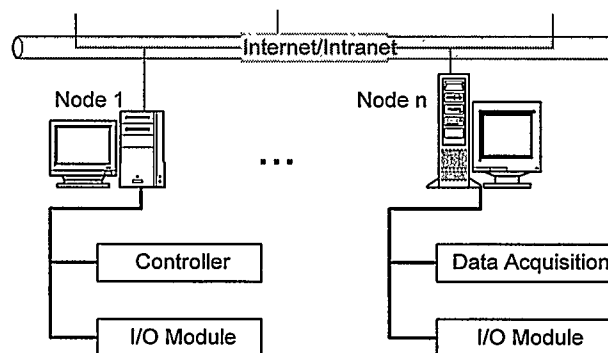


Figure 4.4 Nodes in WBDCS

For example, there are n nodes as depicted in Figure 4.4. Node 1 connects to the controllers via their I/O module and node n connects to a data acquisition system via its I/O module. Node 1 and node n are different control and monitoring systems. Node 1 can

be a client of node n , when a user opens a web browser on node 1 and types the URL of node n in the browser, the user will be able to download the applets as GUIs from a web server of the system and can make any requests for services provided by node n . At this moment, node 1 is a client and node n is a server.

4.3 The Architecture of WBDCS

WBDCS is designed using a web-based multi-tier client/server software architecture and CORBA technology. As mentioned in section 4.1, two-tier architectures have problems with maintainability. The need to install the client application on every client machine can be costly depending on how many clients there are and how often updates will be made. The web-based multi-tier distributed object architecture attempts to address this issue.

4.3.1 Web-based Multi-tier Approach

WBDCS is designed using a multi-tier client/server software architecture with a web application approach. It consists of a web server and an IIOP gateway, control servers, devices, and a database. Its architecture is depicted in Figures 4.5 and 4.6 using different points of view. It is a multi-tier web application, which includes a client tier, a middle tier, and a data tier as illustrated in Figure 4.6.

The web server (Apache HTTP Server or VisiBroker Gatekeeper) is where the applets are located. The IIOP gateway (VisiBroker Gatekeeper) is an OMG-CORBA compliant GIOP proxy server, which enables CORBA clients and servers to communicate across networks, while still conforming to security restrictions imposed by Internet browsers, firewalls and Java sandbox security. It will be detailed in subsection 4.3.4.

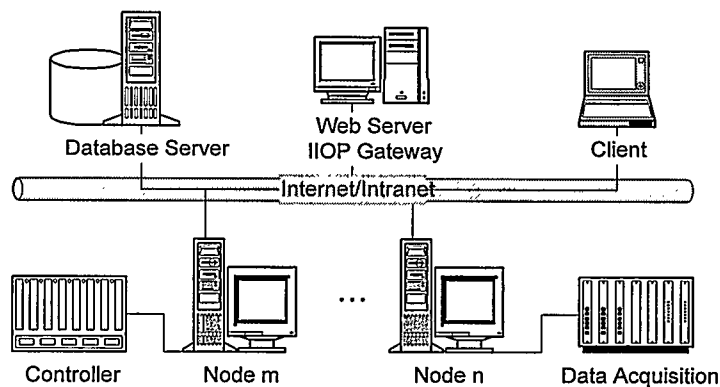


Figure 4.5 Architecture of WBDCS

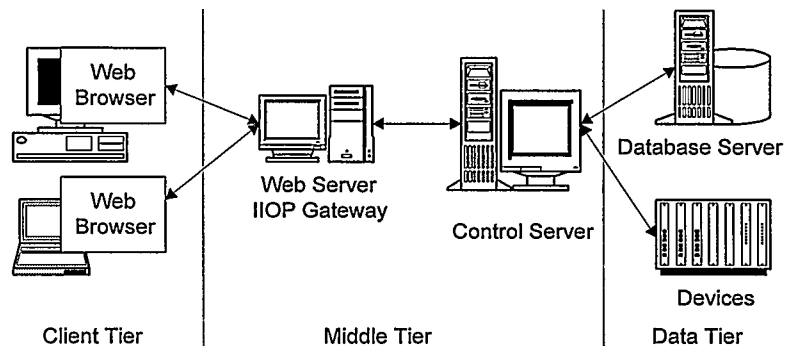


Figure 4.6 Web-based multi-tier distributed object architecture

The control servers are usually located in nodes. They are the middle tier between client and data tiers. They provide two parts of services. One is control service; the other is information management service. The control service includes sampling, processing and responding services. The information management service includes storing real-time data into the database and updating real-time data on client GUIs in a certain period of time.

A user may access the application by navigating to the node's URL using a web browser on a client machine. Applets are downloaded from the web server and run in a user's browser. The prime advantage of this web-based approach is that all code associated with the client tier is downloaded dynamically from the web server. There is usually no need

to install any application-specific software on the client machine. This greatly reduces long-term maintenance costs.

4.3.2 Distributed Object Approach

A distributed object system is a system in which all entities are modeled as objects. It is a popular paradigm for object-oriented distributed applications. CORBA is a standard framework for distributed object systems. It allows a distributed, heterogeneous collection of objects to interoperate across a network.

Distributed object technology allows large server programs to be broken down into several smaller server objects. Each object can potentially reside on a different machine on the network. Server objects can even be run on small desktop computers rather than on a large server machine. Since distributed objects allow applications to be split into lightweight pieces that can be executed on separate machines, less powerful machines can run heavily demanding applications [Lewandowski98].

WBDCS uses CORBA as its distributed object framework. This approach offers an improvement in the scalability of the system. Many distributed object frameworks that are discussed in previous chapters support one or more forms of object pooling, which is a critical feature for implementing scalable distributed object systems.

4.3.3 Analysis of the Approaches

The growing popularity of distributed object technologies has been driven by several problems with the two-tier client/server approach, which include the following:

- Scalability - This term refers to how easily a particular solution can be extended from a small-scale application to a large-scale one. Many applications work well

with just a few functions and users, but fall apart when having to support large numbers of functions and users. Also, some applications perform well on a Local Area Network (LAN), but may not work well on a Wide Area Network (WAN).

- Maintainability - This term refers to how costly it is to administer a particular solution, which includes costs associated with updating the software and distributing updated versions to client machines.

By using web-based distributed object technologies, the above problems can be minimized and sometimes eliminated. There is usually no need to design complex infrastructure software that improves scalability and maintainability: the necessary infrastructure already exists, and web-based distributed object technologies take advantage of that fact.

As described above, the advantages of using a web-based distributed object solution are obvious. However, there are also disadvantages compared to a two-tier client/server solution. The two-tier solution is the simplest approach: client applications talk directly to servers. The web-based distributed object approach requires the use of web servers, web browsers, and intermediate server objects. The extra layers of software improve the scalability and maintainability of the system, but at the cost of simplicity.

There are also disadvantages compared to a non web-based distributed object solution. The primary means of dynamically downloading code into a browser used in this system are Java applets. Using Java applets introduces the following disadvantages:

- Security - Running an applet using a browser has certain security restrictions. Therefore, most dynamically downloaded applet code cannot do all of the things

that a standalone client application can do. This issue will be discussed in the following subsection.

- Performance during initialization - Because client application code must be downloaded from a web server, initialization time is much longer than that of a standalone application.
- Increased network traffic - Downloading an applet (thin client) from a web server usually increases network traffic compared to a standalone version of the same client application installed on a client machine. But it might not increase the traffic if a client application contains both presentation logic and business rules like the two-tier client/server solution in section 4.1.
- Run-time performance - Java “bytecode” was designed to work in any browser on any platform. It must be interpreted by the JVM in a browser, and converted to native machine instructions. This conversion process could potentially decrease performance of applications. Standalone applications are usually in the form of compiled code, which requires no interpreter.

In most cases, these disadvantages are acceptable given the great number of advantages.

4.3.4 Applet Security Issues

Applets are used as client GUIs by dynamically downloading from the web server in WBDCS. In addition to the disadvantages mentioned above, there are some primary security restrictions imposed on Java applets such as file I/O, printing and network access. It is referred to as the Java “Sandboxing” security model, which limits the

applets' effectiveness in distributed object applications. Network access restrictions are as follows:

- An applet can only establish network connections with the host that served the applet.
- An applet can only accept network connections from the host that served the applet.

In other words, Java “Sandboxing” security prevents Java applets from communicating with server objects located on servers other than the ones running on the host from which the applets were downloaded.

These network restrictions create a big problem for CORBA application. CORBA provides location transparency, that is, as long as a client holds an Interoperable Object Reference (IOR), it can invoke operations on a server object, regardless of the location of the server object. Applet sandboxing breaks CORBA location transparency [Brose01].

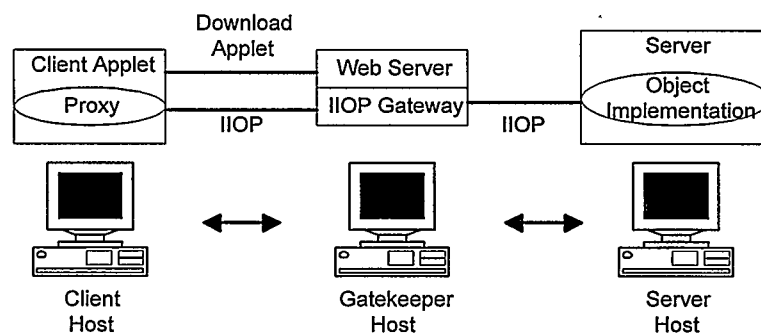


Figure 4.7 IIOP gateway

These restrictions can be solved by using an IIOP gateway on a web server as illustrated in Figure 4.7, such as VisiBroker Gatekeeper. The IIOP gateway acts as a proxy for the CORBA object by sending requests to the object and passing responses back to the

applets. Without the IIOP gateway, Java applets will only be able to use references to objects that reside on the web server host.

4.4 The Design of WBDCS

The software architecture of WBDCS is not only concerned with structure and behaviour, but also with usability and functionality. By using UML, the system architecture can be modeled from different perspectives in order to visualize, specify, construct, and document the system.

4.4.1 Functional Descriptions

The use cases of the system describe aspects of behaviour of the system as seen by its end users or testers. UML allows the static aspects of the system to be captured in use case diagrams and the dynamic aspects of the system to be captured in activity diagrams.

A use case diagram [Booch99] is a description of a set of sequences of actions. An actor represents a coherent set of roles that users of use cases play when interacting with these use cases. Typically, an actor represents a role that a human, a hardware device, or even another system plays.

An activity diagram [Booch99] is one kind of UML diagram used for modeling the dynamic aspects of a system. It emphasizes the flow of control from activity to activity.

An activity is an ongoing non-atomic execution within a state machine. Activities ultimately result in some action, which is made up of excitable atomic computations that result in a change in state of the system or the return of a value.

As illustrated in Figure 4.8, there are four actors in WBDCS. The user represents a role that interacts with the system. The database is an information management system, which

stores the configuration and operation information of the system. The sensor is a device that measures process variables from a physical environment. The actuator is a device that performs an action towards the physical environment.

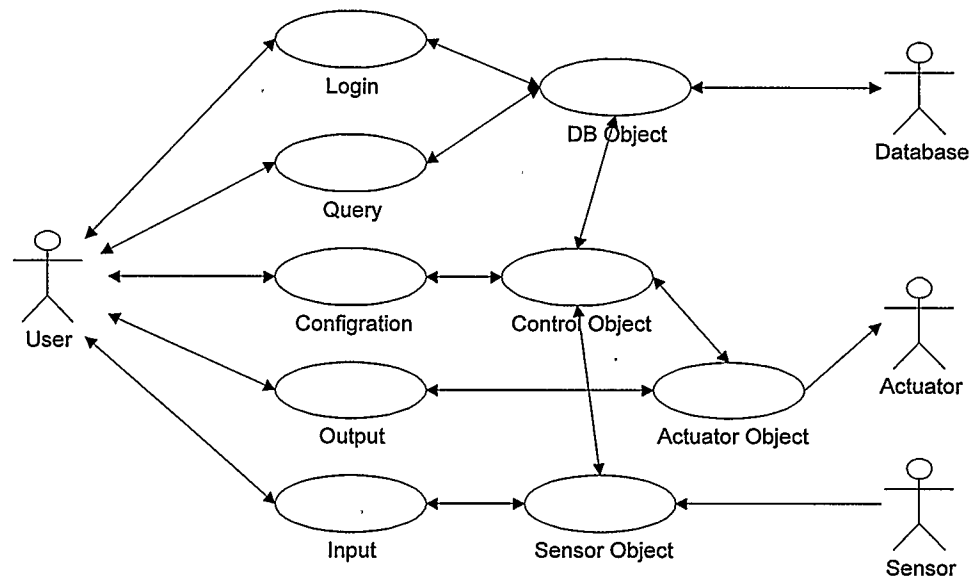


Figure 4.8 Use case diagram of WBDCS

Sensor, control, actuator, and DB objects are made up of CORBA server objects called the control server tier in WBDCS. The sensor object is responsible for acquiring data from the sensor, passing them to the control object and user. The actuator object is responsible for writing data from the user or control object to the actuator. The control object is a controller that processes sampled data and produces a response to the actuator object. The DB object is responsible for collecting data from the control object and for storing them in a database.

Login, query, configuration, output, and input are the client tier in WBDCS. Five use cases have been identified in Figure 4.8. These use cases will be described in detail in the following subsections.

4.4.1.1 Login

In this use case, a user has to login the system before using it. The user's information will be verified to make sure that the user has the authority to use the system.

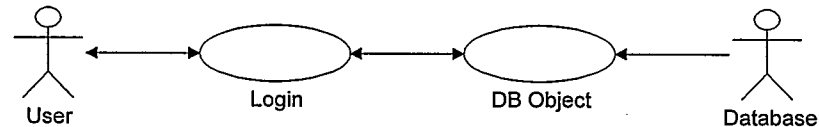


Figure 4.9 Login use case diagram

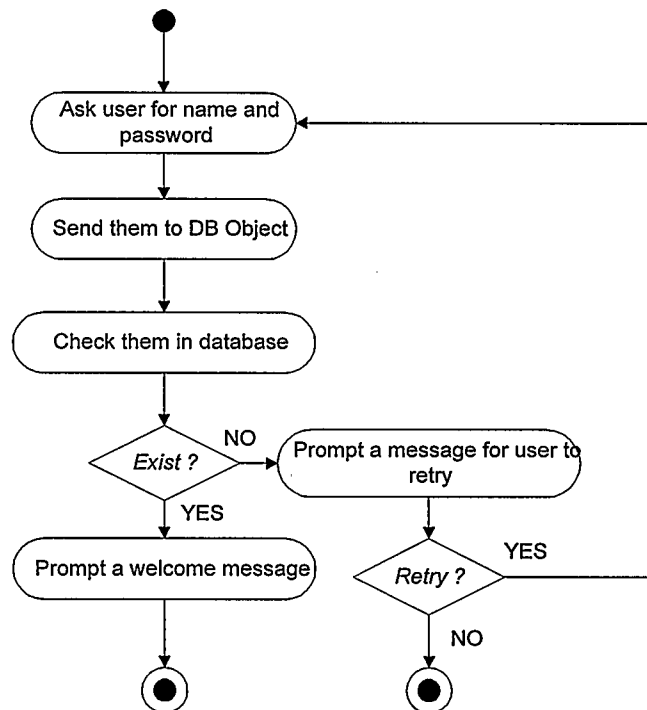


Figure 4.10 Login activity diagram

In Figures 4.9 and 4.10, a user is the actor who initializes the use case. The user name and password will be sent to the DB server object via the network, and the DB object will check if they are the same as the ones stored in the database. The DB object will notify the user about the result of the check in one of two possible ways: one is the welcome

information, which means the user information is valid; the other is a warning message, which means that user name and/or password were wrong.

4.4.1.2 Input

The input process, as depicted in Figures 4.11 and 4.12, is defined as reading data from the sensor. In this use case, the user will first select the input channels and then send an input request to the sensor object. The sensor object will get the data from sensor and send them back to the user.

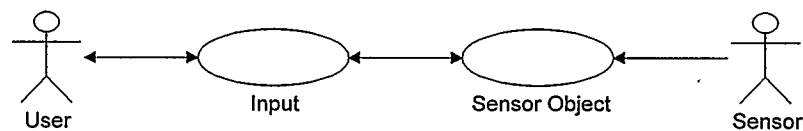


Figure 4.11 Input use case diagram

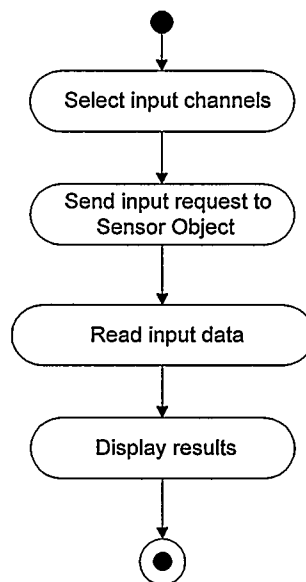


Figure 4.12 Input activity diagram

4.4.1.3 Output

The output process, as illustrated in Figures 4.13 and 4.14, is defined as writing data to an actuator. In this use case, the user first selects the output channels and sets the output values, and then sends an output request to the actuator object, which will write the output data to the actuator.

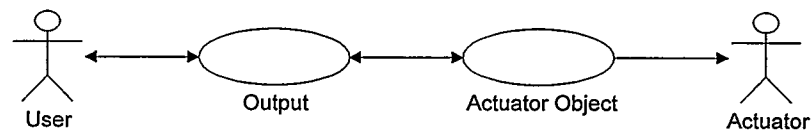


Figure 4.13 Output use case diagram

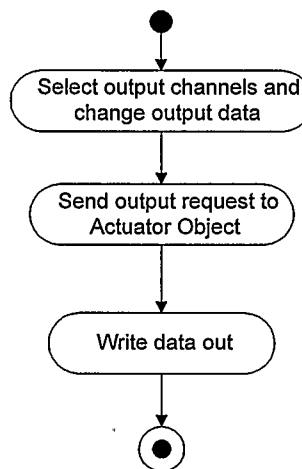


Figure 4.14 Output activity diagram

4.4.1.4 Query

The querying process, as shown in Figures 4.15 and 4.16, is defined as querying historical data from the database. In this use case, the user will first select information to be searched, and then the request is sent to the DB server object. The DB object will access the database, find the requested data, and send them back to the user.

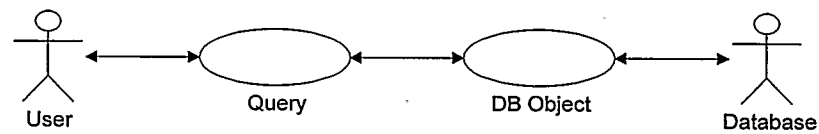


Figure 4.15 Query use case diagram

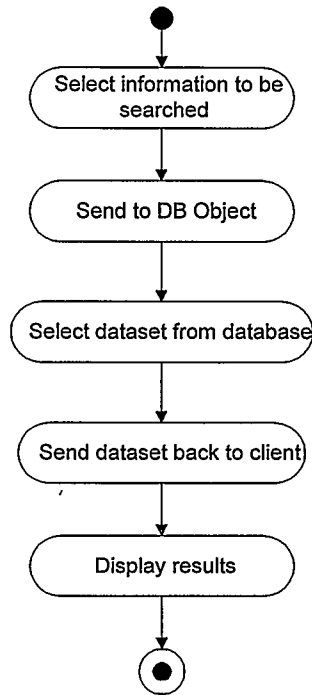


Figure 4.16 Query activity diagram

4.4.1.5 Configuration

The configuration process is defined as configuring the working modes of the controller (control object) and making the control loop work properly as depicted in Figures 4.17 and 4.18. In this use case, the user can select control algorithms, change set point (a desired value for a controlled variable), and switch the controller from manual to automatic mode, and vice versa.

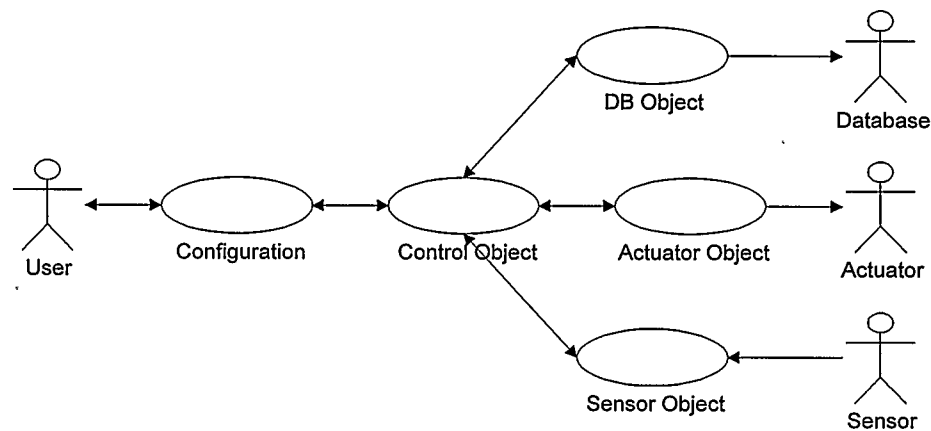


Figure 4.17 Configuration use case diagram

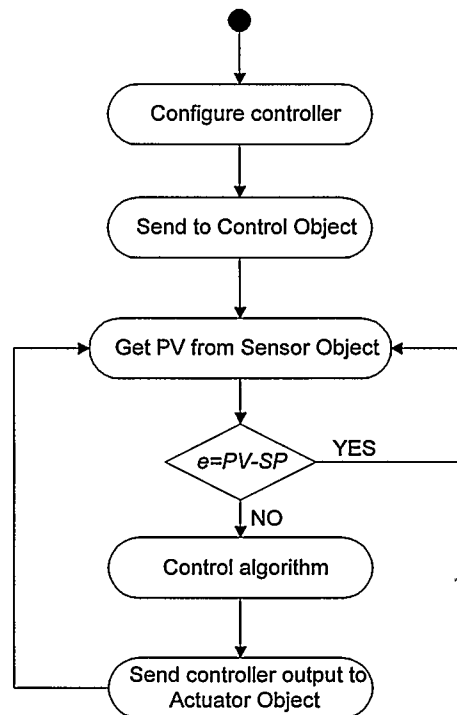


Figure 4.18 Configuration activity diagram

In automatic mode, the controller will cooperate with the sensor, actuator, and DB objects to ensure that a physical process is controlled successfully. The sensor object samples the process variable (PV) measured by the sensor and passes it to the control object. The

control object compares the PV with a desired value or set point (SP) and produces an output using a certain control algorithm, and sends the output to the actuator object. The actuator object manipulates the actuator according to the output of the controller. These control procedures execute continuously to maintain the physical process under control. The DB object will store the configuration and operation information in the database.

4.4.2 Control Algorithms

There are many existing control algorithms that can be adopted and implemented in the controller (control object) of WBDCS. In the following subsections, we will introduce two of the most popular control algorithms.

4.4.2.1 PID Control

PID (Proportional + Integral + Derivative) is a well-established control algorithm, which is commonly used in process industry. A single-input and single output feedback control system consists of a sensor, a controller, an actuator and a process as illustrated in Figure 4.19. The goal of this loop is to maintain the level in the tank at certain value. SP is a predetermined value for the level. PV is the actual level measured by the sensor. Again, the control loop is to maintain PV at a predetermined SP . The PID control algorithm is described by Equation (4.1):

$$e(t) = PV - SP$$

$$m(t) = m_0 + K_c \left[e(t) + K_d \frac{de(t)}{dt} + K_i \int_0^t e(t) dt \right] \quad (4.1)$$

Where:

PV = Process variable, a measured value for controlled variable

SP = Set point, a predetermined (desired) value for controlled variable

$m(t)$ = output from the controller in %

m_0 = manual reset

K_c = controller proportional gain

$e(t)$ = error signal to the controller in %

K_d = controller derivative gain in minute

K_i = controller integral gain in 1/minute

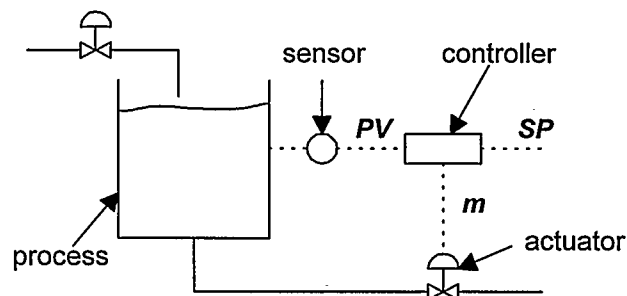


Figure 4.19 Feedback control loop

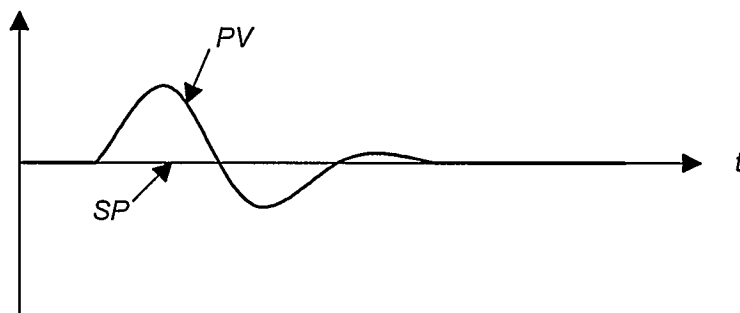


Figure 4.20 PID control response

The principle of this control algorithm is: whenever $PV > SP$, the controller will open the control valve a little bit; whenever $PV < SP$, the controller will close the control valve a little bit in order to maintain $PV = SP$. For example, if the level in the tank changes due

to some disturbances, the control loop will respond immediately to ensure PV approaches SP . Figure 4.20 illustrates the dynamic response of the PID control loop.

4.4.2.2 On-Off Control

The most rudimentary form of regulatory control is on-off control [Svrcek00]. An example of on-off control is a home heating system. Whenever the temperature goes above the set point, the heating system shuts off, and the temperature drops below the set point, the heat system turns on. This control algorithm is shown by Equation (4.2).

$$\begin{aligned} m(t) &= 0 \% && \text{if } PV > SP \\ m(t) &= 100 \% && \text{if } PV < SP \end{aligned} \quad (4.2)$$

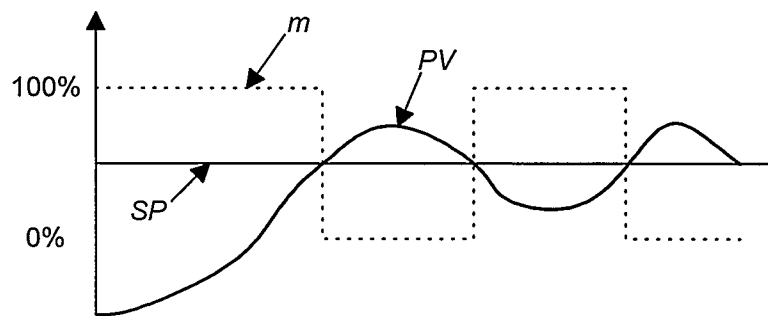


Figure 4.21 On-off control response

The controller output is equal to 0% whenever PV exceeds SP . The controller output is 100% whenever PV is below SP . The ideal dynamic response of the on-off control loop is depicted in Figure 4.21. For simplicity, hysteresis is not considered in the example.

There are different control algorithms for different physical environments. WBDCS allows the implementation of different control algorithms in the control object, i.e., the system allows different controller to be implemented.

5 REAL-TIME SCHEDULING IN WBDCS

5.1 *Introduction*

Distributed control systems are widely used for controlling and monitoring industrial processes. Usually, these systems are real-time systems that are characterized by the need for timing constraints. In other words, a real-time system is a system the correctness of which depends on meeting logical requirements as well as timing constraints. This kind of system can be characterized by the following features [Tsai96]: continuous operation, stringent timing constraints, asynchronous process interaction, unpredictable race conditions, non-deterministic execution, and multiple threads of process interaction. In summary, real-time systems are different from non real-time systems by their timing constraints, which means that timeliness is a main feature of real-time systems.

5.1.1 *Task*

A task in a real-time system is a thread of execution performing a specific function. For example, a task could be a simple thread polling a serial port to check if any data have arrived. This means that a computer periodically asks (polls) the various sensors to see if action is required. A real-time task can be classified as periodic or aperiodic depending on its arrival pattern, or as soft or hard based on its deadline.

In a real-time system, each of the tasks must complete execution before some fixed time has elapsed since its request. This fixed time is known as the deadline of the task. If meeting a given tasks' deadline is critical to the system's operation, the task is called a

hard real-time task. If occasionally missing deadlines of a particular task does not adversely affect the system's performance it is a soft real-time task.

A hard real-time task must satisfy the deadlines on each and every occasion. An example would be a system that checks the temperature in a chemical plant process. The temperature is sampled every 10 ms, the control calculation is carried out, and the output value is sent to the temperature regulator (actuator). This task must be performed within a certain time.

For a soft real-time task, an occasional failure to meet a deadline does not compromise program correctness. An example of a soft real-time task is an automatic bank teller machine. An event is initiated when a customer inserts a card into the machine. The machine response will be specified in terms of an average response time of say, 15 seconds. The actual response time may vary from time to time.

Tasks with regular arrival times are called periodic. A common use of periodic tasks is to process sensor data. For example, a temperature monitor of a reactor should be read periodically to detect any changes promptly. Tasks with irregular arrival times are aperiodic tasks. They are used to handle the processing requirements of random events such as operator requests.

5.1.2 Real-time Scheduling

Scheduling techniques hold great promise as a means to maintain timing correctness of a real-time system. A scheduling algorithm is a set of rules that determines a task to be executed at a particular time slot. A scheduler in a real-time system provides predictability to the system and coordinates resources to meet the timing constraints of

the system. A scheduling algorithm is said to be static if the priorities of the individual tasks are fixed. These priorities could be assigned based on the worst-case execution time of tasks, the period of tasks, or the criticality of the task. A scheduling algorithm is said to be dynamic if the priorities of individual tasks might change from request to request [Liu73].

The major real-time scheduling techniques that have been proposed are priority-driven preemptive and non-preemptive scheduling algorithms, such as the Rate Monotonic (RM) algorithm [Liu73, Lehoczky89] and the Earliest Deadline First (EDF) algorithm [Liu73]. Preemptive refers to the way a currently executed task is interrupted if a request for any higher priority task occurs. This means that whenever there is a request from a task of higher priority, the running task is immediately interrupted and the newly requested task is started.

RM is a static (fixed) priority-driven preemptive scheduling algorithm, in which priorities are assigned according to the request rate (frequency) of tasks. That is, tasks with higher frequency will have higher priority. EDF is a dynamic priority and both a preemptive as well as a non-preemptive scheduling algorithm, in which the task with the nearest deadline for its current request has the highest priority. This means that the EDF algorithm has to calculate the priorities of tasks dynamically. In contrast, the priorities of tasks do not change with time in the RM algorithm.

Most real-time applications in industry have used a fixed priority preemptive scheduling (RM) algorithm. Therefore, RM is also adopted as the scheduling algorithm in WBDCS. The next section describes one of the most common static priority real-time scheduling algorithms, the Rate Monotonic algorithm.

5.2 Rate Monotonic Algorithm

Liu and Layland [Liu73] presented an optimum static priority scheduling algorithm known as the Rate Monotonic (RM) algorithm. In this algorithm, priorities are assigned according to the request rate (frequency) of tasks. Specifically, tasks with higher request rates will have higher priorities.

A task set is said to be schedulable if all its deadlines are met, that is, if every periodic task finishes its execution before the end of its period. Tasks are said to be independent if requests for a certain task do not depend on the initiation or the completion of requests for other task. A necessary condition for schedulability of a given task set is that the sum of the processor utilization of all the tasks is less than or equal to 1. Using [Liu73, Sha90],

A set of n independent periodic tasks, $\tau_1 \dots \tau_i \dots \tau_n$, can be characterized by the following equations using RM algorithm:

$$U(n) = \frac{C_1}{T_1} + \dots + \frac{C_n}{T_n} \leq n \left(2^{\frac{1}{n}} - 1 \right) \quad (5.1)$$

$$i = 1, 2, 3, \dots, n$$

$$\min_{(k,l) \in R_i} \sum_{j=1}^i C_j \cdot \frac{1}{lT_k} \left\lceil \frac{lT_k}{T_j} \right\rceil \leq 1 \quad (5.2)$$

Where,

$$R_i = \left\{ (k, l) \mid (1 \leq k \leq i), l = 1, \dots, \left\lfloor \frac{T_i}{T_k} \right\rfloor \right\}$$

$U(n)$ = the total processor utilization

C_i = the execution time of task τ_i

T_i = the period of task τ_i

RM guarantees that any task set to be schedulable if equation (5.1) holds, and each task is schedulable if equation (5.2) holds.

Most commercial real-time operating systems use the above fixed priority preemptive schedulers.

5.3 *Fixed Priority Scheduling Model*

Katcher, Arakawa and Strosnider [Katcher93] presented a model for fixed priority schedulers. Their model is an extension of the mathematical scheduling framework of Lehoczky, Sha and Ding [Lehoczky89], which expands on the work of Liu and Layland [Liu73].

In [Katcher93], Katcher et al. derived a set of extended schedulability analysis equations for different scheduling implementations in modern real-time operating systems. Their work classified real-time operating systems into two generic implementations: event-driven and timer-driven. Event-driven implementations rely on an external hardware device to generate interrupts. Timer-driven implementations use periodic timer interrupts from a programmable timer. Figure 5.1 and Figure 5.2 depict the differences between timer-driven and event-driven scheduling models.

They further categorized event-driven implementations into: integrated and non-integrated interrupt event-driven scheduling. In an integrated interrupt event-driven scheduling implementation, all tasks are initiated by external interrupts. An interrupt is posted to the processor at the beginning of a task period. The hardware interrupt priority

is matched with software task priority and the interrupt will only be served if the currently executing task is of lower priority. Otherwise, it will remain pending.

In a non-integrated interrupt event-driven scheduling implementation, all tasks are initiated by external interrupts. Every time a new task arrives the current task is interrupted. In other words, the priority of the interrupt associated with a task's arrival has no correspondence to the software priority of that task, and is thus "non-integrated".

If the arriving task has a priority higher than the executing task, preemption will occur.

On the other hand, if the arriving task has a lower or equal priority it will not preempt the current task, but will still cause an interrupt service routine and scheduler execution.

Once the scheduler completes execution, processing of the current task will continue.

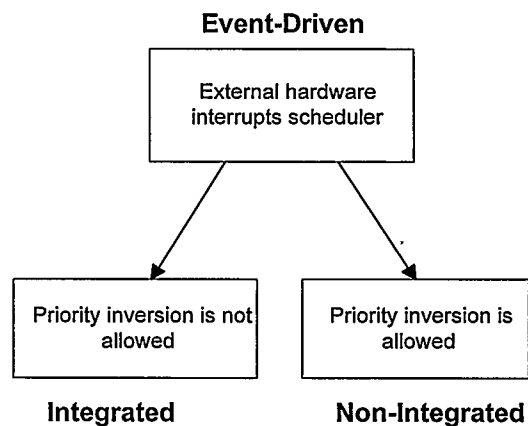


Figure 5.1 Event-driven scheduler

Timer-driven implementations are classified as: timer-driven scheduling and timer-driven scheduling with counter. Timer-driven scheduling uses the interrupts from a periodic timer to interrupt execution and to invoke the scheduler. In timer-driven scheduling with counter, a counter is used to limit the number of scheduling points. At every scheduling point, a counter is initialized with the number of clock ticks to the next task deadline. On

every timer interrupt, the counter is decremented and the scheduler is invoked as the counter expires.

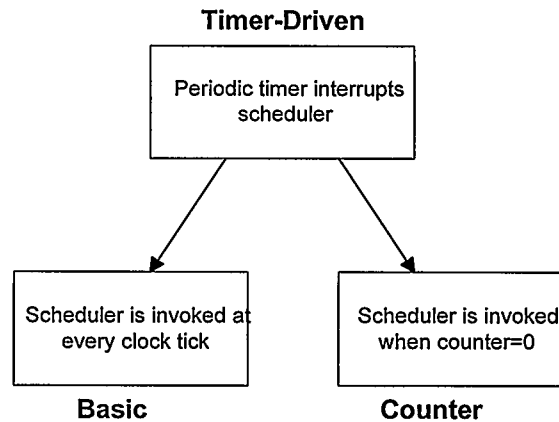


Figure 5.2 Timer-driven scheduler

The above work provides a methodology for incorporating the costs of scheduler implementations for fixed priority scheduling algorithms in a uni-processor environment.

5.4 Real-time Scheduling in WBDCS

In WBDCS, two kinds of real-time scheduling have been defined: time-driven and event-driven scheduling, which is based on the theories and models of Katcher, Arakawa and Strosnider [Katcher93] and Rate Monotonic algorithm and rate monotonic priority assignment [Liu73].

5.4.1 Priority Assignment

In WBDCS, the priorities of the tasks are assigned based on relative importance of both periodic and aperiodic tasks. Higher priority is assigned to strictly timing dependent variables, such as alarms, emergency processing requests from operator stations, emergency processing routines. Lower priority is given to loose timing dependent

variables, such as routine inputting, outputting, configuring, and querying requests from operator stations.

For periodic tasks, rate-monotonic priority assignment is adopted. The tasks with higher request frequency are assigned higher priorities. While for aperiodic tasks, priority assignment is only based on how critical a task is. For example, an alarm event is an aperiodic and emergency task, and the system and operators have to respond and process it immediately. Usually emergency tasks have highest priorities in the task set. Routine requests from operators have lowest priorities in the task set.

5.4.2 Interrupt

The real-time scheduling activity of WBDCS is modeled in Figure 5.3. This part of the system is designed to handle interrupts from both clients and devices. Time-driven periodic interrupts and event-driven aperiodic interrupts are introduced on both client side and device side.

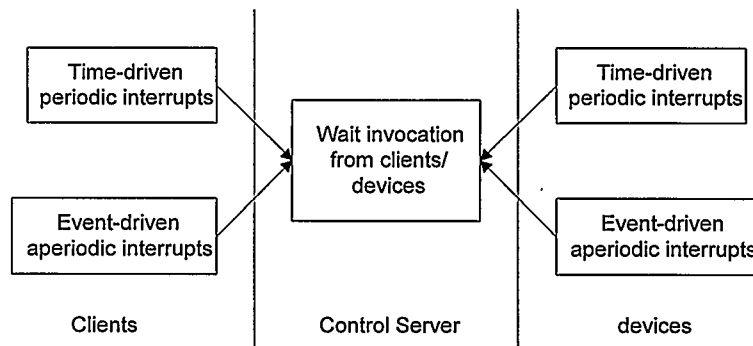


Figure 5.3 Periodic and aperiodic interrupts

On the client side, time-driven periodic interrupts are produced by the programmable timers (software timers) for updating process data and displaying them on GUIs. Event-

driven aperiodic interrupts are initiated by the users for submitting their requests of input, output, configuration, and query.

On the device side, time-driven periodic interrupts are produced by the programmable timers (software timers) in the drivers of devices or hardware timers on the devices. Those timers are responsible for sampling process data at certain points in time. Event-driven aperiodic interrupts are initiated by the interrupt requests from hardware, such as I/O devices. For example, alarm signals will wake up the control server when the level control system in Figure 4.19 is out of control (level is too high or too low). These signals need to be processed immediately and have highest priorities.

5.4.3 *Time-driven Scheduling*

Time-driven scheduling is based on continuous and periodic activities, such as data acquisition (sampling) and real-time information updating. In the time-driven approach, processing is initiated by periodical interrupts of a system clock. When a timer timeout occurs, an internal transition associated with the operational state is triggered [Selic99].

In the prototype of WBDCS, three programmable timers (software timers) are designed and implemented in *control.dll* (see section 6.5 for details) on the device side, which are running in different threads with different priority levels. Time-driven scheduling is triggered by interrupts of the periodic timers. Different timers are responsible for scanning and sampling the different groups of sensors. Each timer has been assigned a priority level on the basis of its request frequency.

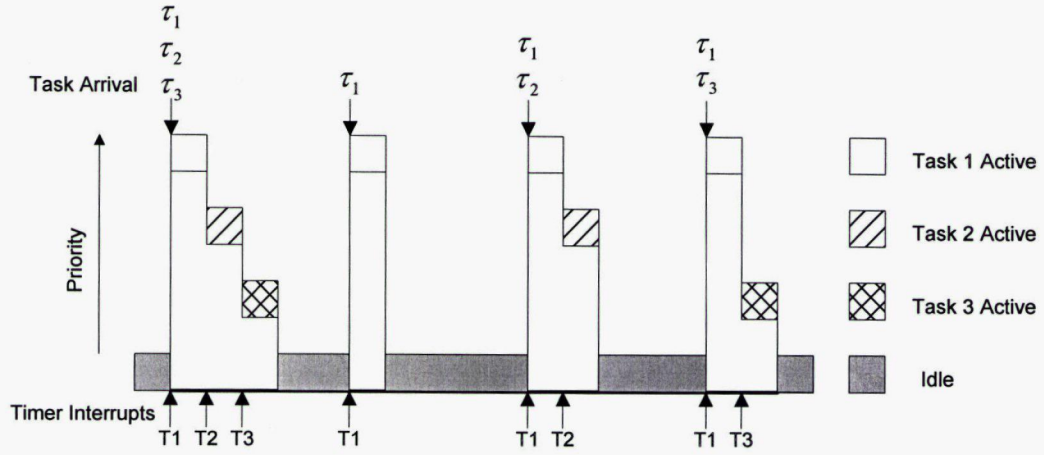


Figure 5.4 Time-driven scheduling

Figure 5.4 shows a test result of fixed priority scheduling for the time-driven interrupts in the prototype of WBDCS. Let τ_1 , τ_2 , and τ_3 , denote a priority-ordered task set, which are the tasks to be executed during interrupts of *Timer1*, *Timer2*, and *Timer3*, respectively. During the interrupt of *Timer1*, τ_1 will be executed and during the interrupt of *Timer2*, τ_2 will be executed, and so on. Let T_1 , T_2 , and T_3 , denote the actual start points of *Timer1*, *Timer2*, and *Timer3* interrupts. τ_1 has a highest request rate i.e., *Timer1* has the highest priority level, while *Timer3* has the lowest priority level. If *Timer1*, *Timer2*, and *Timer3* interrupt requests occur at the same time, τ_1 will be executed immediately because it has the highest priority. After τ_1 is completed τ_2 can start to be executed, and then τ_3 . T_1 , T_2 , and T_3 show the actual start points of each interrupt. In time-driven scheduling the request for any task will occur periodically.

5.4.4 Event-driven Scheduling

Event-driven scheduling is based on discrete or aperiodic activities, such as alarms, emergency processing routines, and operator routine requests. In the event-driven approach, processing occurs in discrete steps, triggered by the arrival of the events.

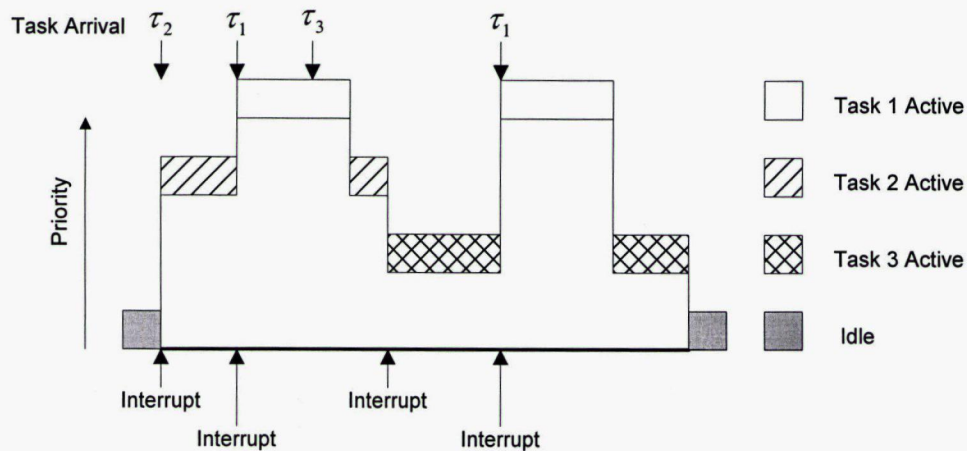


Figure 5.5 Event-driven scheduling

For example, let τ_1 , τ_2 , and τ_3 , denote a set of priority-ordered tasks with τ_3 being the task with the lowest priority. Consider a particular request for τ_2 that occurs at the beginning of the period as illustrated in Figure 5.5. τ_2 will be executed immediately because there is no other task executing at this moment. Requests for τ_1 and τ_3 subsequently occur. Clearly, the preemption of τ_2 by τ_1 will cause a certain amount of delay in the completion of the request for τ_2 . After the highest priority task τ_1 is completed τ_2 can be continued. Although the request for τ_3 has already occurred, it will not be executed until the requests for τ_1 and τ_2 have been executed because τ_3 has the lowest priority in this task set. In event-driven scheduling the request for any task arrives randomly, as a result of operator actions or aperiodic events.

6 A PROTOTYPE OF WBDCS

6.1 Introduction

A prototype of WBDCS has been implemented based on the architectural design in Chapter 4. It consists of control and database applets that act as client GUIs, a control server that manages data input/output (I/O) and information archives, a digital I/O card (PCI215) that performs data I/O, an experiment kit that displays output signals and generates input signals, and a relational database that stores configuration information and I/O data. The implementation architecture is depicted in Figure 6.1.

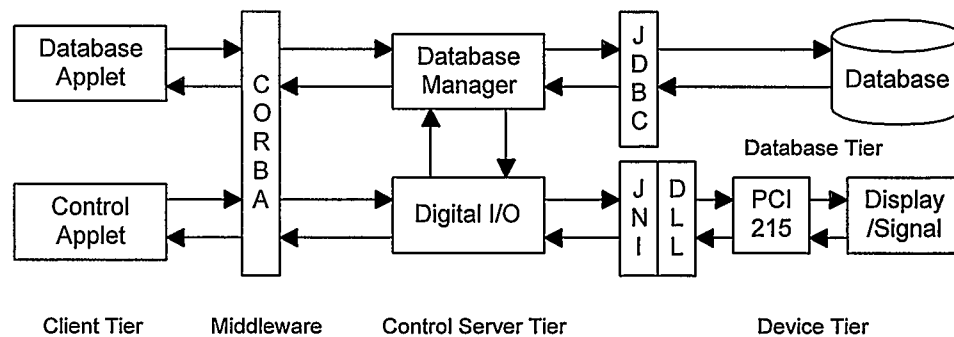


Figure 6.1 Architecture of the prototype

The prototype of WBDCS is a multi-tier web application. The control and database applets represent the client tier; digital I/O (I/O server object) and database manager (DB server Object) are located in the control server tier; the database and controlled devices are the data tier of this application.

In addition, the prototype contains a set of standard interfaces such as OMG IDL, JNI (Java Native Interface), and JDBC API (Java Database Connectivity Application Programming Interface). The IDL defines the interfaces between the client and server parts of the application, and specifies what operations and attributes on the server objects

are available for clients to access. The JNI defines the native methods that are implemented as a dynamic link library (DLL) written in C/C++. The JDBC API is used to connect the control server tier to the DBMS. A detailed description of each subsystem of the prototype is presented in the flowing sections.

6.2 *Client Tier*

The graphic user interfaces of the prototype are implemented using Java applets. An applet is a program written in the Java programming language that can be included in an HTML page. When a user uses a Java-enabled browser to view a web page that contains an applet, the applet's code is downloaded to the user's system and executed by the browser's JVM. As mentioned in Chapter 4, a client machine does not need to install any application specific software to access WBDCS as long as it has a commercial browser installed, such as Netscape Navigator or Microsoft Internet Explorer. There are two applets in the client tier of the prototype. One is called the control applet, and the other is referred to as the database applet.

6.2.1 *Control Applet*

The control applet is a GUI that mainly performs the control and monitoring functions of WBDCS. Three toggled control panels namely configuration, reading, and writing are within this applet.

6.2.1.1 Configuration Panel

The configuration panel, shown in Figure 6.2, is a panel that configures the working modes of digital I/O devices (hardware) before these devices can perform any control

activities. This means that the configuration is a start point for any control action. For example, inputting data from hardware or outputting data to hardware has to be performed after sending configuration information to the hardware and initializing it.

The configuration panel allows the user to assign the number of I/O on the hardware. All channels are bi-directional, which means that each port of the I/O card can be input as well as output channel.

All input channels have to be assigned a sampling rate and priority level. Input channels can be categorized in three different groups. Each group has an assigned sampling rate and priority level. The simplest case is that all input channels are set at the same sampling rate and priority level. In this mode, all input ports are periodically scanned from Port 0 to Port 48. A complicated case is that all ports are assigned in three different groups with different sampling rates and priority levels. In this situation, input ports will be scanned group-by-group depending not only on the priority levels, but also on the sampling rates.

For simplicity, we assume all input channels have the same sampling interval of 1,000ms, Port 0 and Port 1 in Group 1 have a priority level “high”, Port 2 and Port 3 in Group 2 have priority level “low”, and the remaining ports in Group 3 have a priority level “medium”. In this case, all ports will be scanned group-by-group depending only on their priority level. The scanning sequence will be Port 0, Port 1, Port 4 to Port 47, and then Port 2, Port 3. That is, the group with highest priority will be scanned first, then the group with medium priority, and last the group with low priority.

Applet Viewer: distributedcontrolsystem.ControlApplet.class

Applet

Configuration Reading Writing

Group 1

Priority Level: High

Sample Rate: 1000

Group 2

Priority Level: High

Sample Rate: 1000

Group 3

Priority Level: High

Sample Rate: 1000

InputNumber: 48

OutputNumber: 48

Port Name	Port ID	Group ID
Port XA0	0	2
Port XA1	1	2
Port XA2	2	1
Port XA3	3	1
Port XA4	4	3
Port XA5	5	3
Port XA6	6	3
Port XA7	7	3
Port XB0	8	1
Port XB1	9	1
Port XB2	10	2
Port XB3	11	2
Port XB4	12	1
Port XB5	13	1
Port XB6	14	1

Apply Clear

Applet started.

Figure 6.2 Configuration panel

6.2.1.2 Reading Panel

The reading panel, as illustrated in Figure 6.3, is a display panel for all inputs or individual input. Each input channel, port name, port ID, sampled time, value, and status are shown in the input table. If the I/O status is OK, the input value is valid. Otherwise, the input value is invalid. On the reading panel, a user is able to read from any input channel. Furthermore, the user can change the update rate on a client machine. The update rate is the rate that displays the real-time data in the reading table, but it is limited to the highest sampling rate.

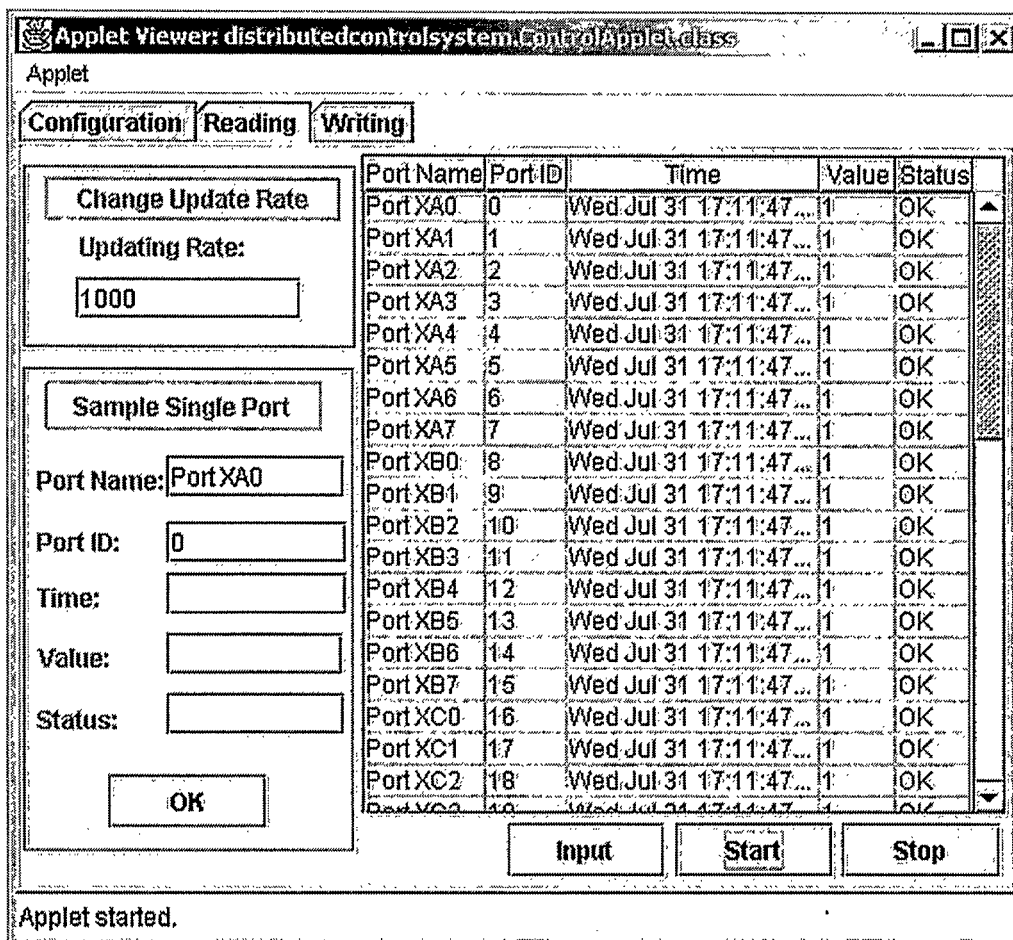


Figure 6.3 Input panel

6.2.1.3 Writing Panel

The writing panel, as shown in Figure 6.4, is a control panel that is used to output data to hardware. The table on the panel can be used to write all output values at once, while an individual output action can be performed as well using the left pane on this panel.

Applet Viewer: distributedcontrolsystem.ControlApplet.class

Applet

Configuration Reading Writing

Write Single Port

Port Name:

Port XA0

Port ID:

0

Set Value:

1

Submit

Port Name	Port ID	Value
Port XA0	0	1
Port XA1	1	1
Port XA2	2	0
Port XA3	3	1
Port XA4	4	1
Port XA5	5	0
Port XA6	6	1
Port XA7	7	1
Port XB0	8	0
Port XB1	9	0
Port XB2	10	0
Port XB3	11	1
Port XB4	12	0
Port XB5	13	1
Port XB6	14	1
Port XB7	15	0
Port XC0	16	0
Port XC1	17	1
Port XC2	18	1

Send Set Reset

Applet started.

Figure 6.4 Output panel

6.2.2 Database Applet

The database applet is a GUI that mainly performs the database management functions of WBDCS. Using this GUI the user can retrieve configuration information and I/O data from the database. A user can also remove information that has been stored in the database as shown in Figure 6.5.

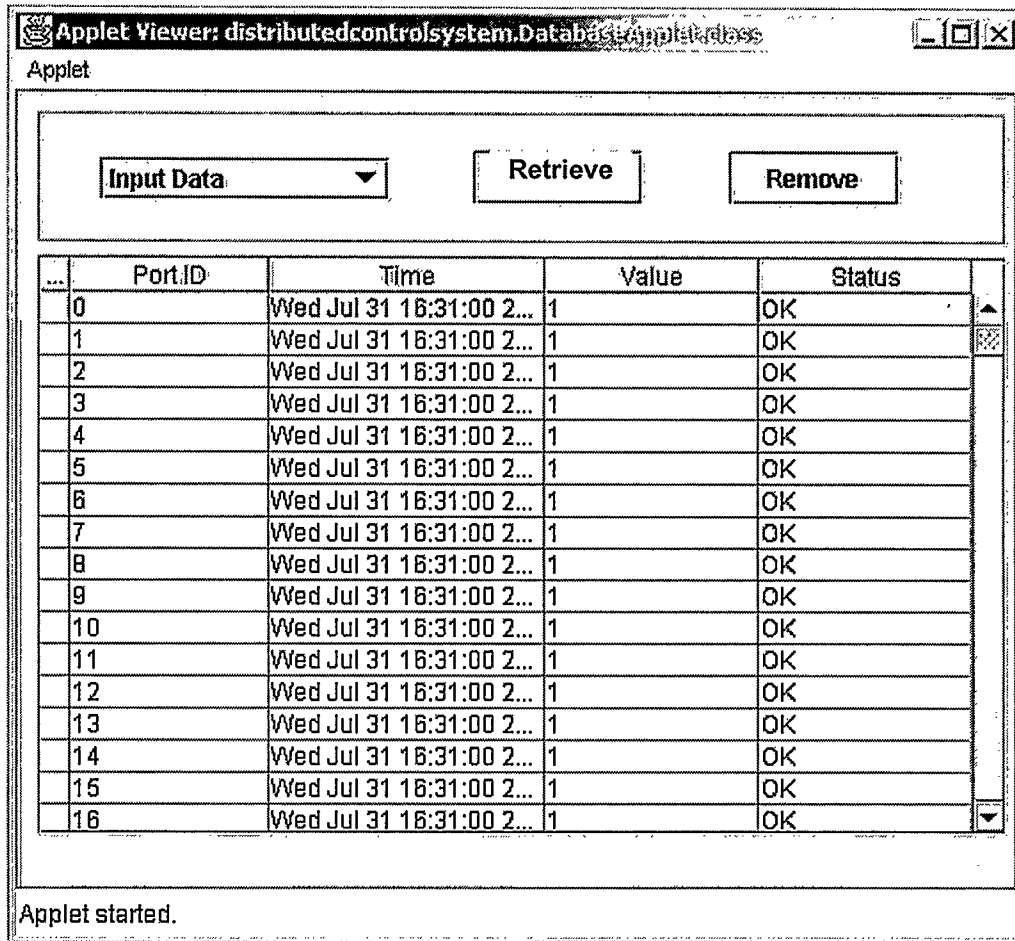


Figure 6.5 Database applet

The database is an essential part of an industrial DCS for recording real-time data and status information for industrial processes and any configuration information submitted from control and monitoring stations by operators or engineers, because of the need for real-time trend displays for a variety of process variables and process graphics and reports. In WBDCS, the purpose of the database management subsystem is to ensure all configuration, reading, and writing information is recorded for future use. A relational database management system, InterBase, is used for this purpose.

6.3 *Middleware*

Transparent access to non-local services and resources distributed across a network is usually provided through middleware, which serves as a framework for communication between the client and server portions of a system. Middleware can be thought of as the networking between the components of a client/server system [Lewandowski98].

The fundamental purpose of middleware components is to enable an application at a client side to access a variety of services on various servers without being concerned about differences between the servers. The ultimate task of middleware components is to route client requests to the appropriate services of servers.

6.3.1 *CORBA*

As discussed in Chapter 4, CORBA has been chosen as the middleware in the implementation of WBDCS. The CORBA specification has been developed as a standard to improve interoperability in a heterogeneous computer environment. Developing client/server systems using technologies that support distributed objects holds great promise, as these technologies support interoperability across languages and platforms, as well as enhancing maintainability and adaptability of the system.

6.3.2 *OMG IDL*

OMG IDL provides a means of separating interfaces from implementations for distributed object applications [Brose01]. IDL [OMG98a] is a strongly typed declarative language that defines the interfaces of objects independently from programming languages. Figure 6.6 shows the IDL file designed for the prototype. This file defines what kinds of services the server objects will provide.

```

module DigitalIO
{
    //define data structure for input
    struct InputDataStruct
    {
        string portName;
        unsigned long portID;
        string sampledTime;
        unsigned long value;
        string status;
    };

    //define data structure for output
    struct OutputDataStruct
    {
        string portName;
        unsigned long portID;
        unsigned long value;
    };

    //define data structure for configuring port
    struct PortInfoStruct
    {
        string portName;
        unsigned long portID;
        unsigned long groupID;
    };

    //define data structure for configuring group
    struct GroupInfoStruct
    {
        unsigned long groupID;
        unsigned long priorityLevel;
        unsigned long sampleRate;
    };

    //define input, output, port, group data list
    typedef sequence < InputDataStruct > InputDataSequence;
    typedef sequence < OutputDataStruct > OutputDataSequence;
    typedef sequence < PortInfoStruct > PortInfoSequence;
    typedef sequence < GroupInfoStruct > GroupInfoSequence;

    interface DigitalIO
    {
        attribute InputDataSequence inputDataList;
        attribute OutputDataSequence outputDataList;
        attribute PortInfoSequence portInfoList;
        attribute GroupInfoSequence groupInfoList;
        readonly attribute long inputListCount;
        readonly attribute long outputListCount;
    }
}

```

```

//configure port
void configureDIOPort( in PortInfoSequence portInfoList);
void configureDIOGroup( in GroupInfoSequence groupInfoList);

//configure data input output channels
void configureInputDimension( in unsigned long inputDimension);
void configureOutputDimension( in unsigned long outputDimension);

//set update rate in input table
void setUpDateRate( in unsigned long updateRate);

//read data from digital I/O chips
void readingAllDigitalPort(out InputDataSequence inputDataList);
InputDataStruct readingOneDigitalPort(in PortInfoStruct onePortInfo);

//write data to digital I/O chips
void writingAllDigitalPort(in OutputDataSequence outputDataList );
void writingOneDigitalPort(in OutputDataStruct outputOneData );
};

interface DatabaseManager
{

    attribute InputDataSequence inputDataRecord;
    attribute OutputDataSequence outputDataRecord;
    attribute PortInfoSequence portInfoRecord;
    attribute GroupInfoSequence groupInfoRecord;
    readonly attribute long inputRecordNum;
    readonly attribute long outputRecordNum;
    readonly attribute long portInfoRecordNum;
    readonly attribute long groupInfoRecordNum;

    void retrieveInputDataRecord(out InputDataSequence inputDataRecord,
                                in string startTime, in string endTime);
    void retrieveAllOutputDataRecord(out OutputDataSequence outputDataRecord);
    void retrieveAllPortInfoRecord(out PortInfoSequence portInfoRecord);
    void retrieveAllGroupInfoRecord(out GroupInfoSequence groupInfoRecord);

    void removeAllInputDataRecord();
    void removeAllOutputDataRecord();
    void removeAllPortInfoRecord();
    void removeAllGroupInfoRecord();

};
};

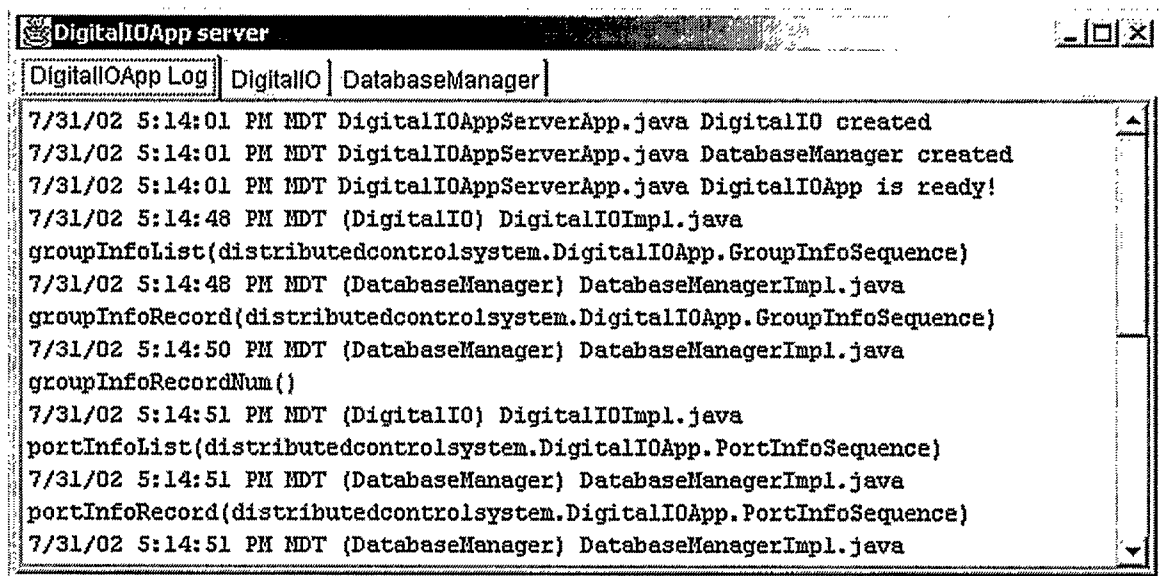
```

Figure 6.6 IDL file

6.4 Control Server Tier

In a multi-tier architecture, the middle tier is between the client environment and the data server environment. There are a variety of ways to implement the middle tier, such as transaction processing monitors, message servers, and application servers. For example, if the middle tier provides queuing, the client can deliver its request to the middle tier and then disengage because the middle tier will access the data and return the answer to the client. In addition, the middle tier can add scheduling and prioritization for work in progress.

The control server tier in the prototype of WBDCS allocates the main body of the application logic to run on a shared host. It shares a digital I/O service engine, a data retrieval engine, and a logging message console for all transactions between the clients, the devices and the database.



```

DigitalIOApp server
DigitalIOApp Log | DigitalIO | DatabaseManager |
7/31/02 5:14:01 PM MDT DigitalIOAppServerApp.java DigitalIO created
7/31/02 5:14:01 PM MDT DigitalIOAppServerApp.java DatabaseManager created
7/31/02 5:14:01 PM MDT DigitalIOAppServerApp.java DigitalIOApp is ready!
7/31/02 5:14:48 PM MDT (DigitalIO) DigitalIOImpl.java
groupInfoList(distributedcontrolsystem.DigitalIOApp.GroupInfoSequence)
7/31/02 5:14:48 PM MDT (DatabaseManager) DatabaseManagerImpl.java
groupInfoRecord(distributedcontrolsystem.DigitalIOApp.GroupInfoSequence)
7/31/02 5:14:50 PM MDT (DatabaseManager) DatabaseManagerImpl.java
groupInfoRecordNum()
7/31/02 5:14:51 PM MDT (DigitalIO) DigitalIOImpl.java
portInfoList(distributedcontrolsystem.DigitalIOApp.PortInfoSequence)
7/31/02 5:14:51 PM MDT (DatabaseManager) DatabaseManagerImpl.java
portInfoRecord(distributedcontrolsystem.DigitalIOApp.PortInfoSequence)
7/31/02 5:14:51 PM MDT (DatabaseManager) DatabaseManagerImpl.java

```

Figure 6.7 Control server

In the control server, each server object generally has a single, well-defined purpose, and makes it available to clients via an interface. There are two server objects implemented in the control server of the prototype, the first object is responsible for performing the configuration of hardware and digital I/O, and the second object is responsible for database management.

6.4.1 Digital I/O Server Object

DigitalIO as shown in Figure 6.7 is a CORBA server object that provides the controlling and monitoring functionality in the prototype of WBDCS. Unlike the design in Section 4.4, the prototype of WBDCS implements only one server object, DigitalIO, which contains all functions of the sensor object, control object, and actuator object due to the simplicity and capability of hardware (PCI215) since only logic control systems can be implemented using this hardware. For example, Port X is an input channel and Port Y is an output channel in a logic control loop. The control algorithm is to turn on a light connected to Port Y whenever Port X is “ON” (Logic 1). In this example, the input process simulates the functions of the sensor object, the control algorithm does the functions of the control object, and the output process does the functions of the actuator object.

6.4.2 Database Management Server Object

The DatabaseManager as shown in Figure 6.7 is another CORBA server object that mainly performs database management functionality in the prototype of WBDCS. This object executes all commands sent from the database applet by users such as retrieving

and removing historical input data, output data, port information, and group information from a relational database.

6.4.3 Control Server Logger

The DigitalIOApp log as shown in Figure 6.7 is a time-stamped, real-time event recorder. When the control server is started, it first creates an I/O object and a DB object, and then tells the user that it is ready to serve user requests. After that, every request from user will be logged in this monitor. Users are able to check all events that happened in the past and their time of occurrence.

6.5 Device Tier

This tier refers to a PCI215 card that is plugged into a PCI slot of the host PC and an external experiment kit connected to the PCI215 card through a 78-pin connector.

6.5.1 PCI215 Card

The PCI215 [Amplicon] is a plug-in, multi-functional digital I/O board, which provides 48 bits of parallel digital input/output and six 16-bit counter/timers [Cooper98]. The card can be used on any PC that supports the PCI bus version 2.1. The host computer must run under one of the operating systems: Windows NT, 2000, 95, 98, and Me.

6.5.1.1 Features of PCI215

- 48-bit flexible, programmable digital I/O
- Six 16-bit, 10 MHz counter/timers with an on board 10 MHz crystal oscillator timing source, each with six programmable counter modes.

- Crystal clock/divider with 5 rates, independently software selectable for each counter/timer clock input.
- Interrupt controlled operations, with the facility for interrupts to be generated from on board timers or one of six external signals.
- PCI bus version 2.1 plug and play interfaces.

6.5.1.2 Digital I/O

The PCI215 card has two 82C55 Programmable Peripheral Interface (PPI) chips with all functions of ports A and B, and optionally port C. The operational mode for each port is established by writing to the control register of the 82C55. The 24 I/O pins of each PPI are brought out to a D-type connector, and can be used to control external devices.

The digital I/O facility of PCI215 provides 48 lines in two clusters of three 8-bit ports as shown in Table 6.1.

Table 6.1 I/O ports on PCI215

	Group A	Group B	Group C
PPIX	Port XA0 – XA7	Port XB0 – XB7	Port XC0 – XC7
PPIY	Port YA0 – YA7	Port YB0 – YB7	Port YC0 – YC7

Each cluster is divided into two groups of 12 bits each. Group A comprises the 8 bits of port A and the 4 most significant bits of port C. Group B comprises the 8 bits of port B and the 4 least significant bits of port C.

A control word sent to the 82C55 control register configures the ports to operate as input, output or bi-directional. The 82C55 PPI can be programmed in various working modes.

The three basic operating modes are summarized in Table 6.2.

Table 6.2 Working modes of 82C55

Modes	Features	Descriptions
Mode 0	Basic I/O	This mode is the power-up default with all ports set as inputs. In mode 0, the PPI provides simple I/O operations. No control signals are required and the ports defined as input reflect the current state of the digital signals on the lines (no latching). The lines of output ports are set to zero on the mode change, and when a port is loaded, the outputs are latched to that value. All 24 bits can be used for input or output by any combination of two 8-bit ports and two 4-bit ports.
Mode 1	Strobed I/O	This mode provides I/O operations on group A and/or Group B each with a simple handshake protocol. In either group, the 4-bit port is used for status and control of the associated 8-bit port. An IRQ facility in this mode is available on PCI215. Each 8-bit port can be used uni-directionally for either input or output operations, both being latched.
Mode 2	Strobed Bi-directional I/O	This mode can be applied to group A only, and provides one 8-bit bi-directional data port and one 5-bit control and status port with IRQ facility. Both input and output operations are latched. Port B can be used in the mode 0 or 1 while port A is in mode 2.

6.5.2 JNI and DLL

Java Native Interface (JNI) comes with the standard Java Development Kit (JDK) from Sun Microsystems. It allows Java code that runs within a JVM to operate with applications and libraries written in other languages, such as C, C++, and Assembly. Programming through the JNI framework enables native methods to function. Native

methods may represent legacy applications or may be written explicitly to solve a problem that is best handled outside of the Java programming environment.

In the prototype of WBDCS, the native methods are functions written in C/C++ in order to perform control functions through PCI215. The implementations of these functions are compiled into a dynamic link library (*control.dll*), which an operating system dynamically loads and links into the processes of the control server. Figure 6.8 shows a part of the code in the I/O server object (Java code), which first declares JNI methods, and then loads the DLL in run time when the digital I/O server object is executing.

```
//declare native methods written in c++ (JNI)
public native void initialization(int group1[ ], int group2[ ], int group3[ ], int configInfo[ ]);
public native void setInputDimension(int inputDim);
public native void setOutputDimension(int outputDim);
public native void writingAllPorts(int values[ ]);
public native void writingOnePort(int portID, int value);
public native void readingAllPorts(int values[ ], String time[ ], String status[ ]);
public native void readingOnePort(int portID, int values[ ], String time[ ], String status[ ]);

//load library control.dll
static {
    System.loadLibrary("control");
}
```

Figure 6.8 JNI and DLL

Software that directly configures and initializes the PCI215 card and performs I/O actions is programmed as a Dynamic Link Library using Microsoft Visual C/C++. The *control.dll* performs all operations on PCI215, such as initializing the card, inputting data from and outputting data to PCI215.

Using the JNI framework, native methods can directly access data buffers on the server object. The JNI also enables us to use the advantages of the Java programming language to catch and handle exceptions raised from the native methods in the Java server object.

6.5.3 Experiment Kit

An experiment kit of the WBDCS prototype consists of two parts: the display circuit and the signal circuit. The display circuit is designed for displaying the output signals from I/O ports, and the signal circuit is designed for simulating input signals to the I/O ports.

6.5.3.1 Display Circuit

The PCI215 card is a programmable digital I/O board that provides 48 bits of parallel digital I/O. According to the specification of PCI215, the current is not strong enough to drive the LEDs. Figure 6.9 shows a circuit designed to drive a LED. The LED will be 'ON' when port PPIXA0 is logic 1, and it will be 'OFF' when port PPIXA0 is logic 0.

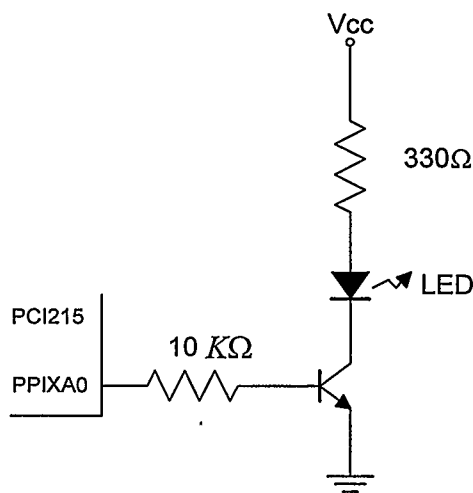


Figure 6.9 Display circuit

6.5.3.2 Signal Circuit

In order to simulate inputs to the ports of PCI215, an input signal circuit is designed on the experiment kit as shown in Figure 6.10. If the switch is up, PPIXA0 will be supplied with a V_{cc} (logic 1) signal. If it is down, the PPIXA0 will receive 0 V (logic 0).

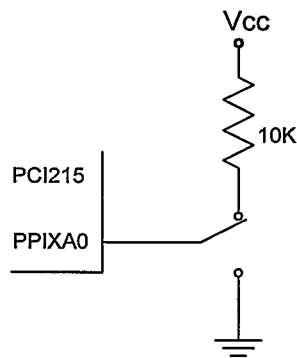


Figure 6.10 Signal circuit

6.6 Database Tier

The database maintains the system data and control status in a distributed control system. In the prototype of WBDCS, the JDBC API facilitates access of the DB server object to InterBase.

6.6.1 JDBC

JDBC API [Allamaraju00] is an industrial standard for database independent connectivity between the Java programming language and a wide range of relational databases. This means that the JDBC API provides cross-vendor connectivity and data access across relational databases from different vendors. It also provides a set of generic database access methods for Structured Query Language (SQL)-compliant relational databases.

Using the JDBC API, the DB server object in the prototype makes the connections with InterBase, executes SQL statements, processes the results extracted, and sends the results back to the client.

6.6.2 *InterBase*

InterBase [Borland] is an open source relational database that runs on Linux, Windows, and UNIX platforms. In the prototype of WBDCS, four data tables are established to store real-time process data in the database management system, such as input data, output data, port information, and group information tables.

6.7 *Deployment*

The deployment process occurs when web applications have been fully tested and are ready for production. At this point we will deploy the client programs on end-users' desktops or server applications on server machines. For the prototype, the client programs were deployed on a web server because they are HTML pages and applets.

Figure 6.11 is the homepage of the prototype of WBDCS. It is downloaded from a web server of the system and running in Microsoft Internet Explorer on a client computer. Figure 6.12 and Figure 6.13 are also HTML pages. But what makes these pages different from the homepage is that they contain applets. The applets are able to visualize the I/O data on an actual I/O card and to communicate with the control server via CORBA and to perform controlling and database management functions remotely.

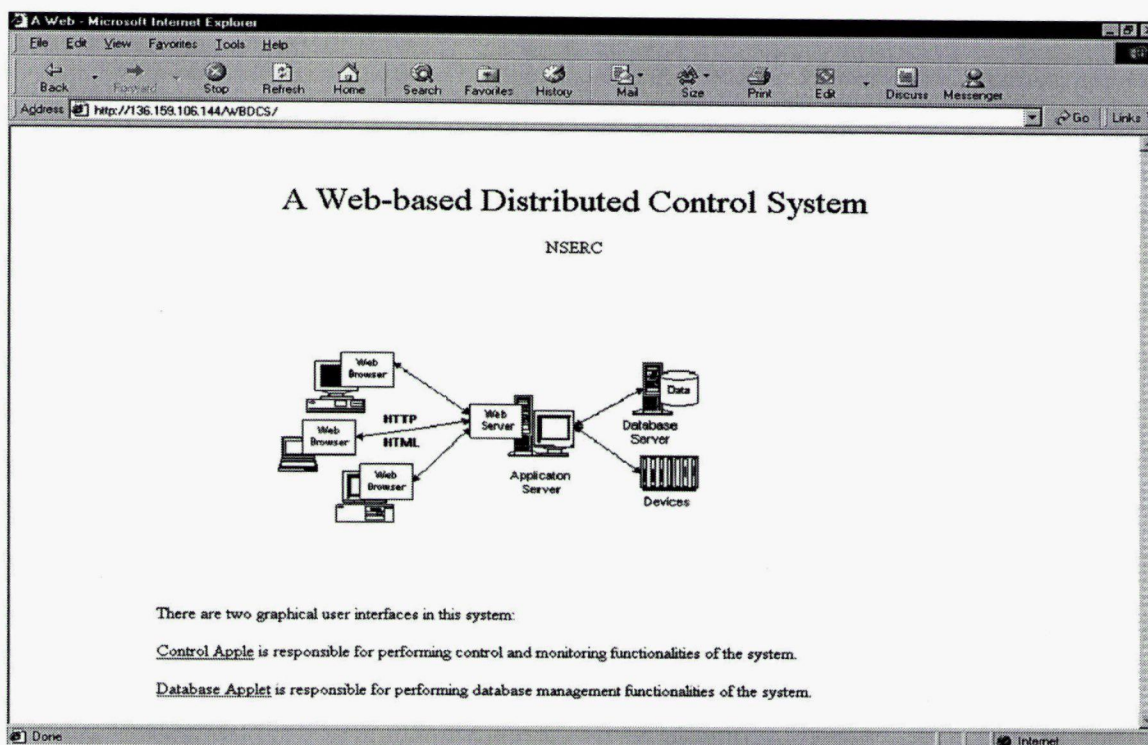


Figure 6.11 Homepage

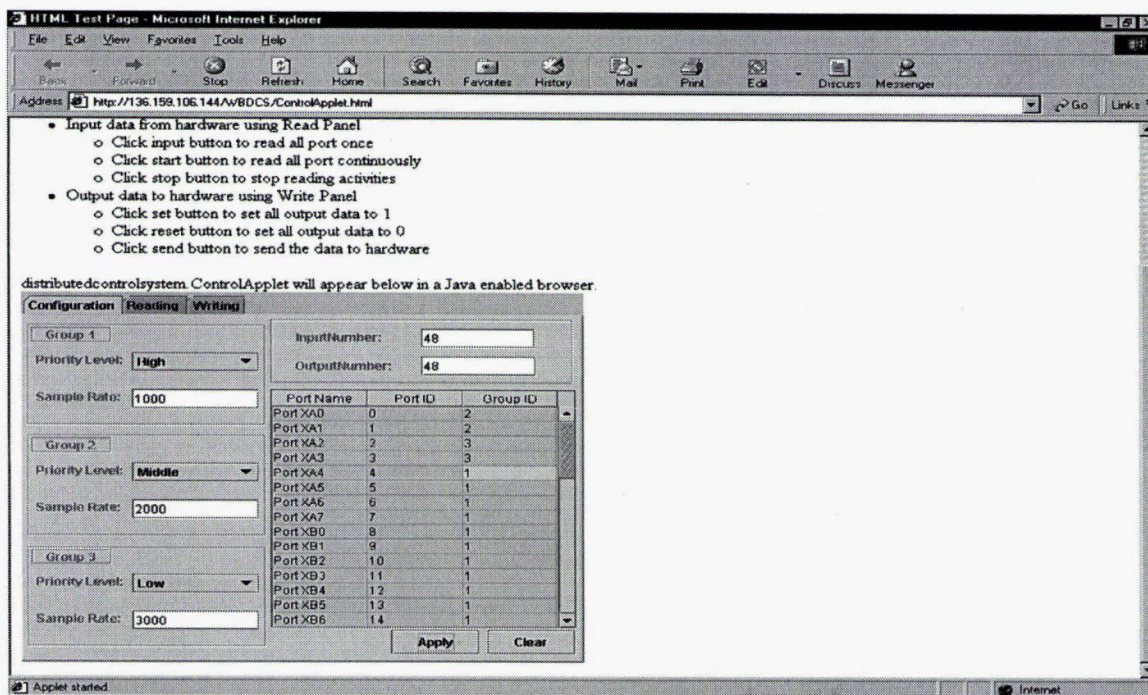


Figure 6.12 Control applet page

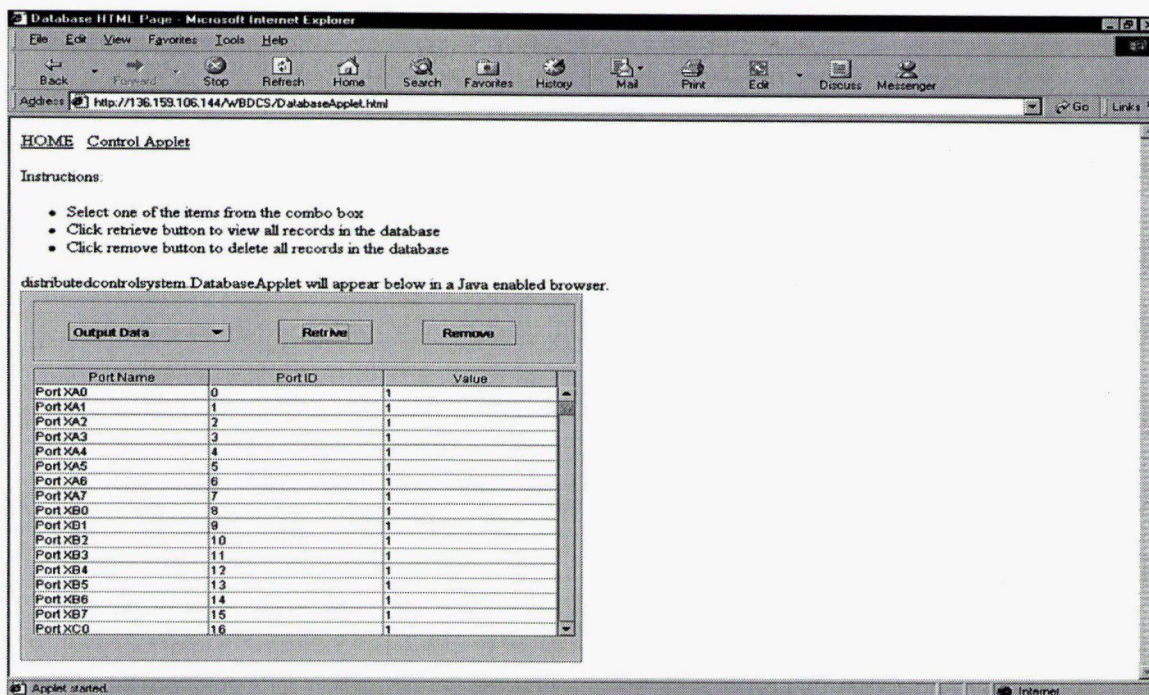


Figure 6.13 Database applet page

7 TESTING AND ANALYSIS

In order to evaluate whether the proposed architecture can achieve the expected results, performance, scalability and schedulability testing of the prototype of the framework is performed. Performance represents the end-to-end response time to the requests of a single user at a time. Scalability shows the ability of a system to maintain the same response times as the number of simultaneous users increases [Allamaraju00]. Schedulability is achieved if all the tasks in a task set meet their deadlines.

Performance, scalability and schedulability of a system depend mainly on the speed of the hardware and network. These tests are conducted on a 10MHz intranet and Internet, a Pentium-I 200MHz and AMD 800MHz personal computers. The hardware of the testing environment may limit the performance of the prototype but it gives us an understanding of how the system performs in our pilot testing environment. Higher performance can be expected when faster computers and networks are used.

7.1 Testing

Performance and scalability testing is conducted in two different environments: one is on a local intranet; the other is on the public Internet. The testing results are shown in Table 7.1 under a normal network traffic load.

Table 7.1 Testing result

	On Intranet		On Internet	
Performance	< 1 second		1-2 seconds	
Scalability	1 users	3 users	1 users	3 users
	< 1 second	< 1 second	1-2 seconds	1-2 seconds

Performance is tested by measuring the time from a user sending out a request to the user receiving the corresponding response. In this test, the request is to set the 48 LED channels on or off and display the current status back on the user screen.

Scalability testing measures the same response time. But it evaluates if the system can maintain the same response time for each of the users when the number of simultaneous request increases. The test results show nearly no time differences when the number of simultaneous user increases.

Schedulability testing is performed within a node. Testing code is implemented in a dynamic link library (control.dll). We have found that a task of 48 input/output channels in three groups can always meet the required schedulability in both 500ms and 1000ms when rate monotonic priority is adopted.

Furthermore, time-driven and event-driven scheduling behaviours are also tested in the prototype. The tests are similar to the schedulability testing, but in these test cases we focus on how time-driven and event-driven interrupts are initiated and processed in terms of assigned priority levels and sampling intervals in three groups of 48 input requests. We have found that a task with a higher priority will always interrupt the execution of the ones with a lower priority as required. Detailed testing results are described in section 5.4.

7.2 Analysis

The architecture of WBDCS is aimed at small to medium sized systems, 2 to 20 distributed nodes in low frequency sampling process such as petrochemical processes. The architecture itself may not impose too many limitations. However, the speed of

hardware and network may limit the applicable scope of the framework. The design of the framework delegates the highly time-dependent tasks to the local nodes. A local computer executes those tasks without interfering with the network and can achieve a very high speed. For example, in a closed loop control system, a remote user sends a setpoint via the network to a local node computer and the local computer can conduct the closed loop control on the order of milliseconds. But sending and changing setpoints, acquiring control status, and displaying them to the end user are not very time critical, and usually take more than one second in most DCSs.

Even though the prototype is a very simple LED system, it can simulate the processes of sending setpoints to and acquiring control status from a local node on a network. The testing results positively show the feasibility of the framework of WBDCS.

Unfortunately, more sophisticated and large-scale tests have not been done in this thesis due to time and resources constraints. More tests are required to further evaluate the performance, scalability and schedulability of the system. Searching for bottlenecks of the system and improving them remain as future work. The following methodologies are proposed for further testing and de-bottle-necking.

7.3 Proposals of Test Plan

7.3.1 Rigid Testing on the Prototype

More rigid tests need to be conducted with the current prototype. In this case, some codes (time stamps) are needed to be added to both client and server side. For example, in Figure 7.1 a client sends a request to a server object at time t_1 , the server object receives it at t_2 , then the server object processes the request and sends a response back to the client

at t_3 , and the client gets the response and displays it to the user at t_4 . The response time is $(t_4 - t_1)$. Table 7.2 illustrates the times to be measured in the prototype.

Table 7.2 Performance testing

	Outgoing time	Incoming time	Processing time
Performance	$t_2 - t_1$	$t_4 - t_3$	$t_3 - t_2$

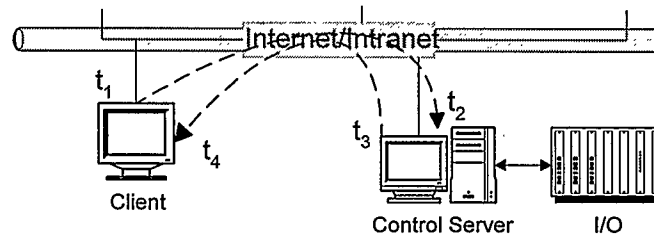


Figure 7.1 Performance testing

The outgoing time is the time spent on a network when a user sends a request to a server object. The incoming time is the time spent on the network when the server object sends a response back to the user. The processing time is the time consumed in a node for the server object to process the user's request, such as acquiring data from an input channel. By measuring these times we can identify the components, which consume the most time during the cycle of a request and response.

In the real world, a scalable system means that the response time increases only very little when the number of simultaneous users increases [Allamaraju00]. Scalability testing can be automated by using simulated simultaneous requests because it is not feasible to have more than twenty or even hundreds of real users to test a system simultaneously. Furthermore, performance and scalability testing should be conducted on a variety of traffic loads and bandwidths of a network, as well as different computer hardware and software configurations.

7.3.2 Virtual Plant Testing

Simulation technology is fairly popular in recent years. It is very common in petrochemical industry to use simulation tools to simulate a real plant for problem solving and process analysis. HYSYS [HYSYS] is one such simulation tool used in petrochemical industries and educational institutes.

Using HYSYS to simulate a real process is another testing plan for further evaluation of the feasibility and applicability of the framework of WBDCS. The simulated process is known as a virtual plant, which can be a signal-input and signal-output (SI/SO) system like the level control system depicted in Figure 4.19, or can be a very sophisticated multi-input and multi-output (MI/MO) system as illustrated in Figure 7.2. This testing scenario is to control a simulated distillation column.

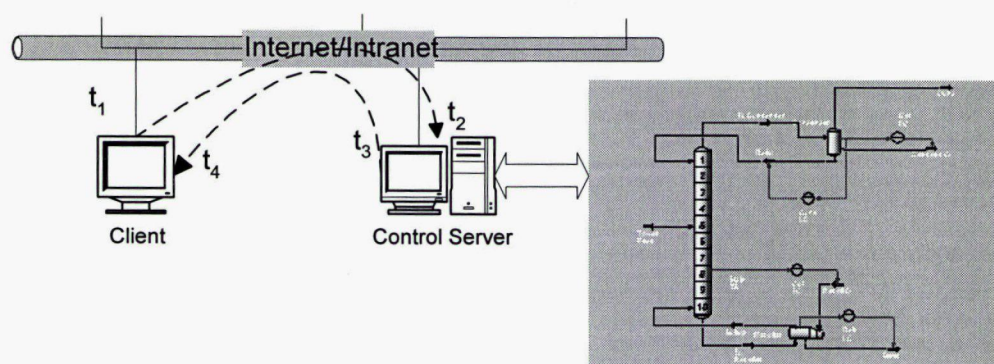


Figure 7.2 Virtual plant

Nodes in WBDCS can control the temperature, pressure, level and flowrate in the virtual plant via I/O interfaces of HYSYS. The main development procedure and test plan include:

- Development procedure:
 - Developing HYSYS dynamic cases as a virtual plant.

- Designing I/O interfaces that manage I/O between the control servers and the virtual plant.
- Implementing the control server consoles and control, sensor, and actuator objects in the local nodes, which includes I/O modules and control algorithms for SI/SO and MI/MO systems.
- Designing and developing corresponding graphic user interfaces (applets) on client side.
- Test plan:
 - Performance and Scalability testing for both SI/SO and MI/MO systems.
 - Schedulability testing for both SI/SO and MI/MO systems.
 - Controller performance analysis for different control algorithms.

One of the goals of this testing project is to fully implement the designed framework shown in Figure 4.8. In order to further evaluate real-time scheduling behaviours in WBDCS, a real-time CORBA [OMG98b] (see 8.2.1 for details) should be adopted in the testing project because we need to be able to schedule the time-driven and event-driven activities not only between the control server and virtual plant, but also between the control server and client. More complete schedulability, performance and scalability research and testing on real-time web application are anticipated in this proposed testing project.

8 CONCLUSIONS AND FUTURE WORK

Technologies and standards that can be adopted to construct a framework for a web-based distributed control system are investigated in this thesis. The integrated architecture takes advantage of Internet/intranet technologies and the World Wide Web to control and monitor industrial processes. Its major capabilities include distributed computing nodes communicating with each other via the Internet/intranet, visualization of real-time processes and data, and supervision and control of industrial process status. The operation of industrial processes can be controlled remotely or locally using a web browser.

8.1 *Summary*

A framework of WBDCS has been described in this thesis. WBDCS is designed by using a web-based multi-tier client/server architecture and distributed object technology approaches. It provides a new way to implement a distributed control system on the basis of the WWW and Internet/intranet. It also provides a framework to integrate existing control systems together and make them accessible and manageable via the Internet/intranet. The multi-tier client/server architecture and CORBA middleware used in this framework provide a solution to the objectives of designing a portable, scalable, and maintainable web-based distributed application.

There are two parallel subsystems in WBDCS: one is a distributed control subsystem, which is usually a hard real-time system; the other is a centralized supervisory subsystem, which is usually a soft real-time system. The distributed control subsystem consists of a number of nodes, which are geographically dispersed across the Internet/intranet. Each node may have different software and hardware configurations and control functions.

Devices in a control loop can be distributed in different nodes across the Internet/intranet, and they can also be localized within a node depending on how strict timing requirements are.

A centralized supervisory subsystem (similar to SCADA) is used to bring the data from all sources together, organize it, store it, and present it to the client over the Internet/intranet, which provides engineers and operators with a comprehensive picture of the status of plant operations. On the one hand, a centralized database management system is designed to manage and store historical data and operational status for all distributed nodes. On the other hand, there is a control server in each distributed node that manages and maintains real-time data and operational status of controlled processes and makes them available and accessible to the clients. Briefly, the centralized supervisory subsystem allows a user of the system access to real-time and historical data and to the operational status of all distributed nodes from one single machine on the network. A web server, a database, and a number of CORBA server objects are involved in managing information transactions between nodes, as well as between clients and nodes. For example, process variables, such as alarm signals, status, and real-time data, can be transferred to CORBA server objects from sources, and then the objects can make them accessible via the Internet/intranet.

WBDCS is a web-based distributed real-time system combining on-line visualization devices and remote service capabilities. Several important issues are discussed in the design of WBDCS. First of all, a portable, scalable, maintainable web-based distributed architecture is introduced. Secondly, the scheduling and processing of time-driven and event-driven interrupts are discussed. Thirdly, to avoid delays on the Internet, improve

reliability of communication, and speed up real-time response, time-critical parts of the system can be localized. By localizing real-time control and centralizing information management, WBDCS presents a way to introduce the Internet and World Wide Web into real-time control systems.

A prototype of WBDCS has been successfully implemented to control a set of remote digital devices. Resources and devices are distributed across the Internet/intranet in multiple computing nodes. An authorized user can access local or remote nodes to perform control functions, such as real-time data collection and visualization, and controlling and monitoring of remote devices. Tests showed satisfactory performance as it was expected from the design.

8.2 Discussion and Future Work

8.2.1 Reliability and Speed

Reliability and speed are key criteria for successful real-time applications. Usually, a high-speed local area network is used in most industrial real-time applications as communication media, including DCS, SCADA, and PLC systems.

Unlike a LAN, the Internet potentially presents lower reliability and speed. In order to avoid delays in a network, improve the reliability of communication and transmission, and speed up the real-time response, the time-critical part of WBDCS can be localized. The localized applications can be implemented using a low level programming language, such as C or assembly. For example, a closed loop control system can be locally implemented in a node. The local control system can perform control activities in milliseconds. Non-time-critical information, such as process status and variables, are

further transferred to the control server, and then made accessible and manageable via the Internet/intranet.

One of the future work on improving the reliability and timely features of WBDCS will be to adopt the real-time CORBA (RT-CORBA) products as the middleware. The real-time CORBA [OMG98b] specification defines standard middleware features that allow applications to allocate, schedule, and control CPU, memory, and networking resources necessary to ensure end-to-end quality of service support.

8.2.2 Scalability and Maintainability

A critical factor for a distributed system is its ability to grow with the number of users, the amount of data, and the required functionality. The application should be small and fast, and able to handle future demands without sacrificing performance or reliability. Scalability refers to how easily a particular solution could be extended from a small-scale application to a large-scale one.

The multi-tier client /server architecture of WBDCS makes it much more scalable than a two-tier architecture would do. One of the design considerations in WBDCS is to be scalable when the number of users of the system is increased and when additional functions of the system need to be added. Scalability both on the number of users and functions is a significant advantage of WBDCS.

Maintainability refers to how costly it is to administer a particular application, including the costs associated with updating the software and distributing updated versions to client nodes. There are also some considerations in the design of WBDCS. First, using middleware products is a way to reduce the maintenance cost because these products are

well developed and tested. Second, using a multi-tier client/server architecture enables separate functionalities in the different tiers of the application: when one tier needs to be modified other tiers can remain the same. Third, using Java applets as the client GUIs makes a great deal of savings in long-term maintenance cost because no matter how many users access one of nodes via the Internet/intranet, the GUIs are automatically downloaded from the web server of WBDCS. There is no need to install any client application on other nodes.

8.2.3 Authority and Security

Access authorization to the system is an essential requirement for WBDCS and needs further work. If a user wants access to the system by using its URL in a browser, it first needs to provide a login page with a form to accept the login name and password of the user. Once the login is successful the user is able to download applets and carryout any further tasks, otherwise the user is refused. Therefore, only authorized users may gain access to a local or remote node to perform the distributed control functions.

Security is also an important issue for a web-based distributed system especially when the application is deployed on the Internet/intranet. Two ways could be used to secure the messages that have to be transmitted over the networks. One way would be to implement security features when the application is developed using Secure Socket Layer (SSL) [Thomas00]. This means that some code would have to be added to WBDCS for securing HTTP and IIOP connections. The other way would be to use VisiBroker Gatekeeper when the application is deployed on the Internet/intranet. In this case, the application developer would not need to worry about the security features of the system. In the

deployment phase, SSL features provided by the Gatekeeper can be configured to secure the communications across the Internet/intranet. However, these issues are part of the future work.

REFERENCES

- [Allamaraju00] Subrahmanyam Allamaraju, et al. "Professional Java Server Programming J2EE Edition", *Wrox Press Ltd.*, ISBN: 1-861004-65-6, 2000.
- [Amplicon] <http://www.mev.co.uk/dio.htm>
- [Bennett94] Stuart Bennett, "Real-time Computer Control", *Prentice Hall Europe*, ISBN: 0-13-764176-1, 1994.
- [Booch99] Grady Booch, James Rumbaugh, Ivar Jacobson, "The Unified Modeling Language User Guide", *Addison Wesley Longman Inc.*, ISBN: 0-201-57168-4, 1999.
- [Borland] "Interbase Documentation".
- [Brose01] Gerald Brose, Andreas Vogel, and Keith Duddy, "Java programming with CORBA", third edition, *John Wiley & Sons, Inc.*, ISBN: 0-471-37681-7, 2001.
- [Chen02] L. Chen, Y. Wang, "Design and Implementation of a Web-Based Distributed Control System", *Proceedings of the 2002 IEEE Canadian Conference on Electrical and Computer Engineering (CCECE'02)*, Canada, May 2002.
- [Chen03] L. Chen, A. Eberlein, "A Framework of A Web-based Distributed Control System", *Proceedings of the 2003 IEEE Canadian Conference on Electrical and Computer Engineering (CCECE'03)*, Canada, May 2003.
- [Cooper] D.N. Cooper, "PCI215 Instruction Manual Part N^o 860 035 14 Issue A3" Amplicon Liveline Limited.
- [Decker01] D.L. Decker, "What is in store for DCS systems? Where are they headed?", *2001 Conference Record of Pulp and Paper Industry Technical Conference*, 2001.

- [Emerson] <http://www.emersonprocess.com/systems/products/index.html>
- [Honeywell] <http://www.honeywell.com/acs/index.jsp>
- [HYSYS] <http://www.hyprotech.com/hysys/>
- [Janke99] M. Janke, "OPC-simple software integration for legacy systems", *IEEE Industry Applications Society Advanced Process Control Applications for Industry Workshop*, 1999.
- [Kang02] Weonjoon Kang; Hyoungyuk Kim; Hong Seong Park, "Design and performance analysis of middleware-based distributed control systems", *Proceedings of 2001 8th IEEE International Conference on Emerging Technologies and Factory Automation*, Volume: 2, 2001.
- [Katcher93] D.I. Katcher, H. Arakawa and J.K. Strosnider, "Engineering and Analysis of Fixed Priority Schedulers," *IEEE Transactions on Software Engineering*, 19(9), September, 1993.
- [Lehoczky89] J. Lehoczky, L. Sha, and Y. Ding, "The rate monotonic scheduling algorithm: exact characterization and average case behaviour," *Proceedings of 10th IEEE Real-Time Systems Symposium*, Santa Monica, CA, December 1989.
- [Lewandowski98] Scott M. Lewandowski, "Frameworks for Component-Based Client/Server Computing", *ACM Computing Surveys*, Vol.30, No.1, March 1998.
- [Lewis97] R. Lewis, "Design of distributed control systems in the next millennium", *Computing & Control Engineering Journal*, Volume: 8 Issue: 4, August 1997.
- [Liu73] C.L. Liu, and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard real time environment," *Journal of the Association for Computing Machinery*, v.20, n.1, pp. 44-61, January 1973.

- [Luh96] Yih Ping Luh, Shean-Shyong Chiou, Jau-Woie Chang, "Design of distributed control system software using client-server architecture", *Proceedings of The IEEE International Conference on Industrial Technology*, 1996.
- [Marti01] P. Marti, J.M. Fuertes, G. Fohler, "An integrated approach to real-time distributed control systems over fieldbuses", *Proceedings of 2001 8th IEEE International Conference on Emerging Technologies and Factory Automation*, Volume: 1, 2001.
- [Nogiec98] J.M. Nogiec, E. Desavouret, D. Orris, J. Pachnik, S. Sharonov, J.C. Tompkins, K. Trombly-Freytag, "A distributed monitoring and control system", *Proceedings of the 1997 Particle Accelerator Conference*, Volume: 3, 1998.
- [OMG] <http://www.omg.org/>
- [OMG97] Object Management Group, Specification of the Portable Object Adapter (POA), OMG Document orbos/97-05-15 ed., June 1997.
- [OMG98a] Object Management Group, The Common Object Request Broker: Architecture and Specification, 2.2 ed., Feb. 1998.
- [OMG98b] Object Management Group, Realtime CORBA, OMG TC Document orbos/98-10-05.
- [OPC] <http://www.opcfoundation.org/default.asp>
- [Selic99] B. Selic, "Turning Clockwise: Using UML in the Real-Time Domain", *communication of the ACM*, Vol.42, No.10, October 1999.
- [Sha90] L. Sha and J. B. Goodenough, "Real-Time Scheduling Theory and Ada," *IEEE Computer*, April 1990.
- [Siemens] <http://www.sea.siemens.com/process/default.html>

- [Stolen99] L.H. Stolen, "Distributed Control System", *INTELEC '99., The 21st International Telecommunication Energy Conference*, 1999.
- [Svrcek00] William Y. Svrcek, Donald P. Mahoney, Brent R. Young, "A Real-Time Approach to Process Control", *John Wiley & Sons Ltd.*, ISBN: 0-471-80363-5, 2000.
- [Tan01] Kok Kiong Tan, Tong Heng Lee, Chai Yee Soh, "Remotely operated experiment for mechatronics: monitoring of DCS on the Internet", *Proceedings. 2001 IEEE/ASME International Conference on Advanced Intelligent Mechatronics*, Volume: 2, 2001.
- [Tan02] Kok Kiong Tan, Tong Heng Lee, Chai Yee Soh, "Internet-based monitoring of distributed control systems-An undergraduate experiment", *IEEE Transactions on Education*, Volume: 45 Issue: 2, May 2002.
- [Thomas00] Stephen A. Thomas, "SSL and TLS Essentials", *John Wiley & Sons Inc.*, ISBN: 0-471-38354-6, 2000.
- [Tsai96] Jeffrey J.P. Tsai, Yaodong Bi, Steve J.H. Yang, Ross A.W. Smith, "Distributed real-time systems: monitoring, visualization, debugging, and analysis", *John Wiley & Sons Inc.*, ISBN: 0471160075, 1996.
- [Vinoski97] Steve Vinoski "CORBA: Integrating Diverse Applications within Distributed Heterogeneous Environments", *IEEE Communications* 14, 2, February 1997.
- [Yang02] Yirong Yang, Shanan Zhu, "Small smart distributed control system", *Proceedings of the 4th World Congress on Intelligent Control and Automation*, Volume: 3, 2002.

[Zukowski98] John Zukowski, “Mastering Java 2”, *SYBEX Inc.*, ISBN: 0-7821-2180-2, 1998.