DISTRIBUTED SYSTEMS, HARDWARE-IN-THE-LOOP SIMULATION, AND APPLICATIONS IN CONTROL SYSTEMS

PAUL HANDRIGAN

Distributed Systems, Hardware-in-the-Loop Simulation, and Applications in Control Systems

by

©Paul Handrigan
St. John's, Newfoundland, Canada

B.Eng., Memorial University of Newfoundland (2001)

A thesis submitted to the
School of Graduate Studies
in partial fulfillment of the
requirements for the degree of
Master of Engineering

Faculty of Engineering and Applied Science
Memorial University of Newfoundland

December 2004

Distributed Systems, Hardware-in-the-Loop Simulation, and Applications in Control Systems

by

Paul Handrigan

St. John's, Newfoundland, Canada

Abstract

This thesis entitled "Distributed Systems and Hardware-in-the-Loop Simulation and Applications in Control Systems" presents a design and implementation of a Hardware-in-the-Loop Simulation with a software model of a process and a real hardware controller. For this thesis, the process simulation tool that has been selected is HYSYS. This software has the ability to model processes from the most simplest examples to complicated industrial processes. The hardware, or controller, that will be used in this thesis is a Modicon Programmable Logic Controller that is often used in many industrial processes.

There are two main goals for this thesis. The first goal is to design and implement a Hardware-in-the-Loop Simulation environment using HYSYS and a PLC that can be used as a verification tool to verify the correctness of the controller and the process. As well, the results of the PLC controller must also be compared with results produced from a third party software controller to show the effectiveness of using a real controller. The simulation environment must also be done using multiple sys-

tems that must communicate over a TCP/IP network to enable remote simulation of the PLC. The second goal is to analyze the effect of time latency on the distributed system.

The work completed in this thesis will further the understanding of the development of Hardware-in-the-Loop Simulation environments and understand the effects of time latency in a real time distributed system environment.

Acknowledgement 1 *First, I would like the PanAtlantic Petroleum Consortium for funding this research.*

Acknowledgement 2 *Secondly, I would like to thank Dr. Siu O'Young for his guidence and support.*

Contents

List of Figures

Chapter 1

Introduction

The complexity of industrial process systems is high and has been increasing at a fast rate in the past number of years. As the complexity of these systems increase, so does the complexity of the hardware and software that are responsible for the control of that particular process. This issue evolves when the complexity of an entire system increases, so does the chance of a failure in the software or hardware of the controller. The need to model a process and a controller together is necessary in order to ensure the correctness of an entire system.

Traditionally, the designer of a control system would take the specification of a process, create the signals that the process would provide to the controller, and debug the controller software and hardware accordingly. This method was sufficient when the engineer was dealing with small and simplistic processes and controls, but this method is not practical for large and complex systems for several reasons.

One of the reasons is that it is too costly to model signals in a lab environment for every different case that a controller has to deal with. The actual cost of lab tests and the amount of time that it takes to implement and perform one of these tests is not feasible for a large, complex system. Another reason is that it is almost impossible to meet the specification of such a process in a general lab environment. Specification can be easily misinterpreted and will cause errors in the system if the specifications are not properly met in the lab. In such a large and complex system the move to computer simulation is the natural choice.

Today, computer systems are becoming much more complex then they were twenty-five years ago and this change in technology is due to two major advances. The first advance is with the actual individual computers themselves. Development of more powerful microprocessors and larger, faster memory enabled the computers to run at speeds thousands of times faster than before while at only a fraction of the cost. A second advancement is the development of computer networks. This allowed the computers to send data from one computer to another. Given this ability, simulation over networks is a possibility.

Computer simulation of the process allows the designer to simulate the entire process at low cost, meet the proper specification of the design, and have the ability to make design changes and with little to no down time. There are many different process simulators on the market, but the one that is being used for the White Rose offshore oil development project is called HYSYS. This tool gives the process

engineers the ability to model any process no matter how simple or complex the actual process. HYSYS is presently being used to model the entire topside of the White Rose FPSO and can help the designers verify if the entire design is correct before any implementation. Getting the design correct before implementation is necessary for completing the project on time and keeping the cost to a minimum.

Along with the process simulation, controller simulation is necessary for simulating a real system. The process and controller can be simulated on the same machine, but each simulator will take a percentage of the CPU, slowing the simulation down making it unable to run at the desired time. Using two computers, one for the process simulation and one for the controller simulation can solve this issue. Each of the simulators will only pass the necessary information across the network making the overall simulation much more efficient.

1.1 Distributed Systems and Controls

1.1.1 Distributed Systems

As computing tasks became more complex and the need for many computer processes needed to complete simultaneously, the idea of using more than one system to complete a single task (or a number of tasks) had evolved. These processes would not be able to act totally independently. They would have to communicate over a medium. This evolved into the idea of multiple computers, communicating over a given net-

work to accomplish a task or multiple tasks as efficiently as possible. This idea of multi-computing over a network is called *distributed systems.*

Definition 3 (Distributed System) *[6]: A distributed system is a collection of independent computers that appears to its users as a single coherent system.*

Distributed systems can be useful in many computation tasks where there is a desire to break down a task to smaller components and run each component on different computer systems or when there are multiple tasks that must be performed at the same time.

1.1.2 Control Systems

Control systems are everywhere in the modern world. Every form of automation will have a control system as its base. For example, the speed control system in an automobile will have a control system to monitor the speed and change the accelerator according to a given set speed.

Definition 4 (Control System) *[7]: A control system is any group of components that maintains a desired result or value.*

The goal of every control system is to obtain and maintain a desired result. The speed of a car, the level of a tank, or any other form of dynamic system can be controlled with a given controller.

1.2 INCA and PPSC Project

1.2.1 The INCA Centre

INCA stands for the Centre for Instrumentation, Control and Automation. As a part on Memorial University of Newfoundland, INCA provides the resources for both research and education. INCA is involved in several major research projects such as the PPSC project and the RAVEN project.

1.2.2 PPSC Project

This component of the PPSC project that the INCA centre is involved with is the Instrumentation, Control and Stimulated Simulation (ICSS). The goal of this component of the project is to incrementally build an interface system module that enables a cluster of DCS (Distributed Control System) over different platforms to communicate with other control systems over a network. The objectives of the of this project are [8]:

1. Increased innovation capacity to undertake innovation in process control systems for offshore oil and gas applications.

2. Design a Self-Contained Stimulated Simulator (SCSS) system to study connectivity over two DCS on a common platform.

3. Design a Universal Simulation Interface (USI) module to study connectivity

over a heterogenous platform of DCS and process control simulators over a LAN (Local Area Network) .

4. Extend the USI connectivity over a WAN (Wide Area Network) to study dynamics of control loops which are closed over Ethernet or similar networking protocols.

5. Establish a closed-loop performance benchmark of the USI module by artificially degrading networking throughout between agents distributed within a cluster.

6. Build and test a pilot USI based system involving the closed-loop control of an offshore operation from an onshore system.

This thesis will concentrate on objectives 1, 3, 4, 5, and 6.

1.3 Problem Statement

1.3.1 Industrial Problem

Within the process control industry, the need to verify the actual process and controller before commissioning is necessary for the design of the overall system. Specifications of the controller and the process must work together properly in order to avoid major issues that can cause major problems when the controller is commissioned into the field.

Computer simulation of both process and controller is one way of verifying that they work with one another. Tools like Matlab, Labview, and other simulation packages can be used to verify the process and the controller. In industry, there are also other simulation tools used for modeling processes and controllers.

HYSYS does represent an accurate model of the actual process, but PID controllers provided by HYSYS do not represent the controllers that will be used in the actual system. HYSYS does provide simple PID controllers for the purpose of dynamic simulation, but the controllers are different compared to a real PLC (Programmable Logic Controller).

Real PLC's must be tested and software programmed onto the PLC must be verified. PLC hardware and software is often responsible for the control of critical systems. If the controller for these critical systems is incorrect, it could cause severe process down time or even cause danger to human life. For this reason we need to test the hardware and software of the PLC.

One way to tune a PLC is to create possible signals that model the process, connect the signals to the PLC under test, and debug the PLC according to the given signals. This method is often called hardware-to-hardware simulation. As mentioned before, this is very time consuming, costly, and inaccurate. Ideally, we want the actual process simulation acting on the PLC. One way that this can be done is to have an accurate software model of the PLC and interact it with the process model and then observe the reaction. This method is often called co-simulation.

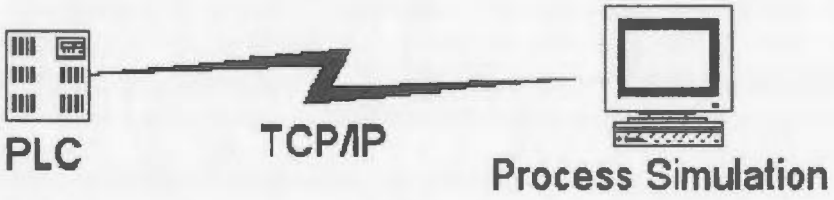

Figure 1-1: Simulation of PLC with HYSYS over a TCP/IP Network

Co-simulation is a software-to-software simulation solution where the process model is concurrently interacting with a PLC model. Usually, each simulation is conducted on either one or many computers with network communication. This type of simulation will show that the model of the PLC will interact correctly with a model of the process. This will enable the control software designer to debug the software before even adding it to a real PLC. Most modern PLC companies such as Modicon do have models and simulators available. Along with the simulation of the PLC and the process most of the issues with a PLC can be solved prior to commissioning. However, there are other issues when you are dealing with computer simulation compared with an actual system.

A computer simulation is dependent upon computer hardware (microprocessor, RAM, etc.), computer software (the operating system), and communication protocols between the computers. These issues can cause problems with the simulation. For example, if an event on the computer simulation takes 100 ms, but in reality it takes 1 ms to complete, you will have latency issues that co-simulation will not model properly. The operating system is event driven and cannot send information from

one simulation to another until a task is completed. The time that a task takes to complete is dependent on how fast the hardware is on each of the systems and how efficient the operating system performs. This is often not consistent on each workstation depending on the scheduling of the processes on each machine. This can cause inconsistency problems when dealing two different simulations on two different workstations. The latency on both machines will not model the reaction time for each the process and more importantly, the PLC. This inability to simulate proper latency between the process and the controller (PLC) may not model the real system causing major issues within the simulation.

This issue is an important one because it prevents the system to run at real-time. Latency between the controller (PLC) and the process simulator may result in an inaccurate simulation result.

Industry Scenarios

Within industry, having the ability to commission a controller to the field without any issues can be a very valuable asset. A scenario where this can happen is during the development of the FPSO for the White Rose project in Marystown. Before PLC's are commissioned to the FPSO, they must be verified in a simulation environment. The best way to verify the PLC's before sending controllers to Marystown to be commissioned is using a hardware-in-the-loop simulation method using PLC's as the real hardware and the HYSYS process model of the topside of the FPSO as the

process model. A platform that is relatively easy to use that could perform this task would be very useful in the commissioning of the controllers. It would significantly decrease the chance of controllers failing in the field and the controllers will work the first time instead of making too many changes in the field that, at times, can be very costly to the project.

A second scenario that is likely to happen within the development of the White Rose FPSO project is that a controller may need to be verified in Marystown, but all of the HYSYS simulation tools are in St. John's. In this case, hardware-in-the-loop simulation may have to be performed over a TCP/IP network to save time and money. For this case, the HILS platform must have the ability to connect to the process simulation in St. John's and perform HILS as if the controller was in St. John's. Such a simulation platform could be very useful and valuable to the FPSO topside project due to the remoteness of the construction site.

A third and final industry scenario could be that the FPSO is at sea and a controller needs to be tuned. It may be very dangerous and costly to tune a controller in the field since changes in the field due to tuning could affect the process and have unwanted results. A way the controller could be tuned is by HILS, but again the HYSYS process simulation and tools are in St. John's. A simulation platform, similar to the second scenario, could be used to simulate a controller over a TCP/IP network with the HYSYS process simulation in St. John's without any unwanted effects that may occur by tuning the controller with the actual process.

1.3.2 Research Problem

In this project we have the challenge to take a PLC (Modicon) and simulate it with a process simulation in HYSYS over a communication network. The initial issues are:

1. How to interface correct data from the process simulator to the PLC and from the PLC to the process simulator?
2. How do you do this over a communication network such as TCP/IP?
3. How can a PLC be configured so that it cannot tell that is it is connected a simulator instead of an actual process?
4. What is the difference between a software controller to software process to a hardware controller to software process?
5. What effects will time latency have on the controller to process simulation when a random delay is added by a Wide Area Network?

This research project should produce client/server protocol to access necessary objects within HYSYS and communicate with a remote PC over a TCP/IP network. Once this connection is established we must interface the remote PC with the PLC over Modbus serial communications. Modbus communications will allow the remote protocol to access the internal memory locations (or registers) that are responsible for the input and outputs on the controller.

This simulator must be compared to a similar system. This will be done with a Labview control algorithm communicating with the HYSYS process simulator over a

TCP/IP network. This can then be compared to the performance of the real PLC communicating with the HYSYS simulation.

When using HILS you are often using more than one system. These separate systems may be communicating over a type of network or some type of physical I/O device. When performing a common task on two or more systems is called distributed systems.

1.4 Overview of Thesis

Chapter 2 gives a discussion on the related literature in several related areas. Chapter 3 illustrates the design and implementation of the HILS over a distributed system using a WAN. In Chapter 4, a presentation of the design, analysis, and results of the experiment that shows the effect that time latency has on HILS over a WAN, as well as the comparison between real hardware controllers and third party software controllers. Finally, Chapter 5 gives conclusions, suggestions, and future work that can be conducted from this thesis.

Chapter 2

Background

The use of simulation in order to validate a system is not a new idea. The idea of modelling a design before implementation has been commonly used for decades. One means of simulation that has been used in the past is by the use of mathematical models. System components could be first modelled as a mathematical equation and then solved to see if the desired result has been achieved. For simple systems these equations could be solved by hand, but using software tools such as Matlab should solve for real and more complex systems the systems.

As techniques of simulation advanced, the use of a part of the real system along with the simulation started. The designer of the system would take a component of the system, integrate it with the simulation environment, and simulate the real component of the system with the software component of the system. One example of this is in control systems where the designer integrates a real controller. This

method can be called Hardware in the Loop Simulation (HILS).

When using HILS you are often using more then one system. These separate systems may be communicating over a type of network or some type of physical I/O device. When performing a common task on two or more systems is called *distributed systems.* HILS is naturally using distributed systems since the hardware is a separate system then the software simulation. When using the "real" hardware, the computations are done at real-time and since time is an issue, the latency between the hardware and the software simulation must be considered.

This chapter will give some literature and background toward HILS, distributed systems, and how this can be done with control systems applications. It will also review background on OPC which is the universal interface that will is used for accessing data within the hardware. The literature that will be presented in this chapter shows what has been previously done in HILS and distributed systems. The rest of this thesis will illustrate the effects of using distributed systems along with HILS and study effect of time latency using HILS over a distributed system. This thesis will also study the use of different controllers (software and hardware) over the distributed system.

2.1 Hardware In The Loop Simulation

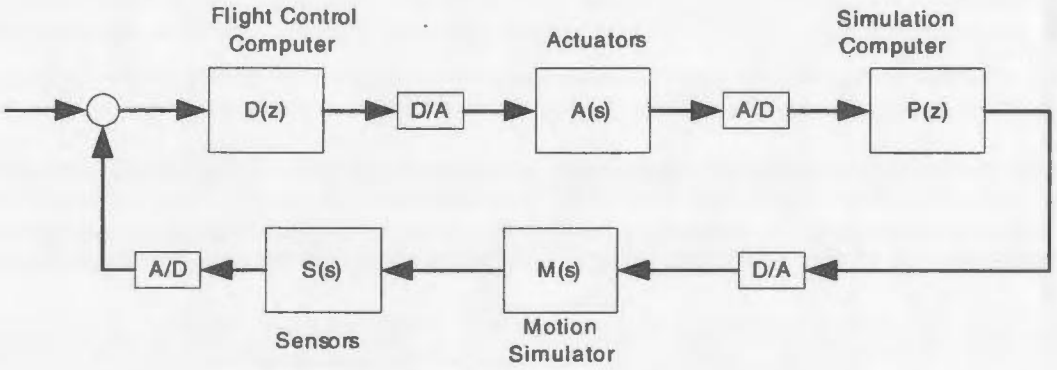

Figure 2-1: HILS of a Flight Control System [1].

This section will outline some related work that has been done with HILS. The article "Control System analysis of Hardware-In-The-Loop Simulation"[1] showed how HILS played a role in developing a missile guidance system. Figure 2-1 shows a simple HILS for flight control.

D(z) represents the real controller for the aircraft and the simulation computer and P(z) represents the simulator for the flight. Information from the simulation computer is then sent to the analog motion simulation M(s) that models the motions of the airframe (pitch, roll, and yaw). The "hardware" used in this simulation are the controller, actuators, and sensors.

This paper then discusses the effectiveness of HILS by analyzing the system using a multivariable z-domain linear technique showing why HILS is effective compared to other simulation methods.

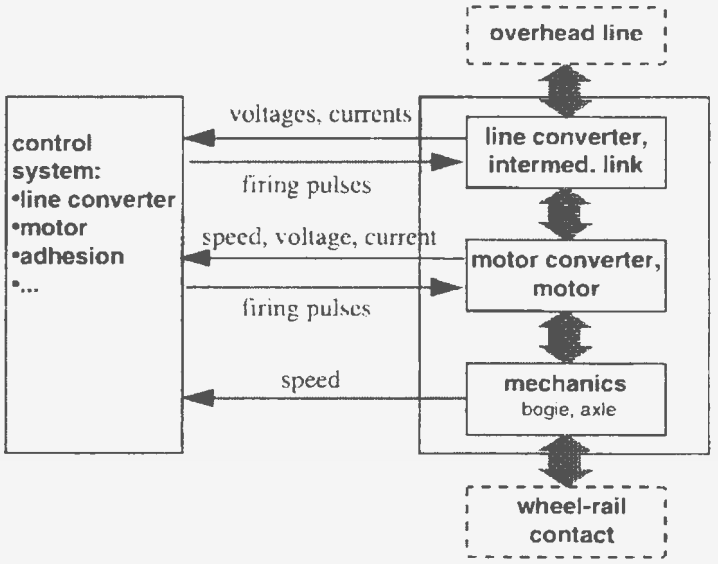

Figure 2-2: HILS Structure of the Loco 2000 [2]

2.1.1 Rail Vehicle Control System Integration Testing Using Digital Hardware-in-the-Loop Simulation

The article "Rail Vehicle Control System Integration Testing Using Digital Hardware-in-the-Loop Simulation"[2] discusses how HILS can help in the design of complex processes. Using the example of the Swiss Federal Railways *Loco 2000*, an electric train that can travel up to 230 km/h and weighs 81 tons. Figure 2-2 (from[2]) shows the design of the HILS environment.

The process component of the HILS structure will output continuous signal such as voltages, currents, and speeds where as the controller will read the continuous signals and output discrete firing pulses, a method called Pulse Width Modulation (PWM).

The HILS structure illustrated in figure 2-2 is has strict real-time requirements on the simulator. The controller has a cycle time between 40-60 μs and the computation of currents from firing pulses is very sensitive, for example, a 1 μs jitter can affect simulation results. In order to meet accurate processing for the overall system, it was found that a simulation frame time of about 30 μs was best suited for this system. Frame time is the time taken to calculate one simulation step with the entire model. At the end of each frame time the simulator communicates with the controller. This shows real-time synchronization between the simulator and the controller.

This system combines both continuous and discrete systems. For this system the vehicle simulation model is the continuous system (or analog system) and is comprised of a set of first order ordinary differential equations (ODE's). The discrete system (or digital system) is the controller and is comprised of a set of combinatorial and sequential equations. Combining discrete and continuous systems is one of the main reasons that real-time is taking so long to replace hybrid simulation. Hybrid Simulation consists of comparators (converts analog to digital) and switches (converts digital to analog) that is a natural technique to establish this link[9], [10].

According to this study, linking discrete and continuous systems have three issues that cause problems[2]:

- External events. (E.g. firing pulses from the controller to the converters.)
- State (internal) events. (E.g. idealized diodes that conduct or block depending on voltage and current conditions)

- Time events. (E.g. waiting time conditions)

In real-time simulation, handling state and external events are the major issues because the limited frame time corresponding to a fixed external step size does not permit adaptive step sizing around the event. Time events are not as significant of an issue because they are usually known in advance and can be dealt with accordingly.

In HILS, there are ways to deal with external and state events. External events can be handled by[2]:

- Small frame time.
- Interrupt-driven integration.

State events can be handled by:

- Small frame time.
- Event time registration and correction.

According to the above solutions, the easiest and best way to reduce the effects of state and external events is to have small frame time, however, depending on the simulation. other methods may be used to optimize the simulation.

Before building the system, they created a model of the system using a simulation package called dSPACE, from a German company that offers fast, modular real-time hardware together with a software interface to Matlab/Simulink[11]. dSPACE systems are widely used for rapid development and HILS in many applications, especially in the automotive industry.

The simulation environment shown in figure 2-2 uses a model of the controller for off-line simulation and sensitivity studies. For HILS, an actual Adtranz controller replaces a model controller. The process model is created using Simulink, but some of the lower level parts are created in C and embedded into the Simulink simulation.

The digital real-time simulator used here replaces previous hybrid simulators of greater cost. This simulator is less costly, requires less maintenance, and only requires two PCs. Comparing the fidelity of both simulators against actual process measurements showed a very high fidelity for both, with no clear accuracy advantage for either the digital real-time simulator or the hybrid simulator[2].

Closed-loop real-time HILS of the vehicle using a real control system simplifies tests and further investigate possible issues during vehicle operation. The digital real-time simulation technique is a definite alternate approach to real-time HILS.

2.1.2 Hardware-in-the-loop Simulation and its application in Control Education

A simulator with an actual PLC can be valuable for verifying that actual control system hardware and software is valid for the given process or plant. The article "Hardware-in-the-loop simulation and its Application in Control Education[3]" demonstrates a method on how to integrate an actual PLC with a computer simulation of a process for the purposes of control system education. This method has a computer simulation of a process that is modelled and analyzed in Matlab and

Simulink . The simulation and analysis component is then interfaced with a real-time kernel. The reason for a real-time kernel is that Matlab/Simulink models are not run at real-time. In order to have a sense of real-time with the I/O of the PLC, a real-time interface is needed. Along with the kernel, there is a real-time model running on the kernel. This model ensures all of the delays and software events are done in real-time.

This simulation system also consists of an interface between the software and the PLC hardware. The PLC is connected to the simulation via an appropriate I/O board. On this I/O board there are digital I/O for the PLC, A/D converter for the actuators and a D/A converter for the sensors. This board is connected to the PC that contains the simulation and a real-time kernel. Within the kernel there are I/O drivers that interface the I/O signals from the signal board to the process simulation. The architecture of this simulation environment is illustrated in figure 2-3(from [3]).

This simulation is a good demonstration for an educational tool. It demonstrates how a simulated plant reacts with the I/O of a PLC in a lab environment. This method could be used in industry for some PLC applications, but if you want to use multiple PLC's over a network, this method would be very costly to implement do to the fact that the ADC and DAC converters on the board must meet the standard of the PLC and you would also need hardware for every PLC that is simulated.

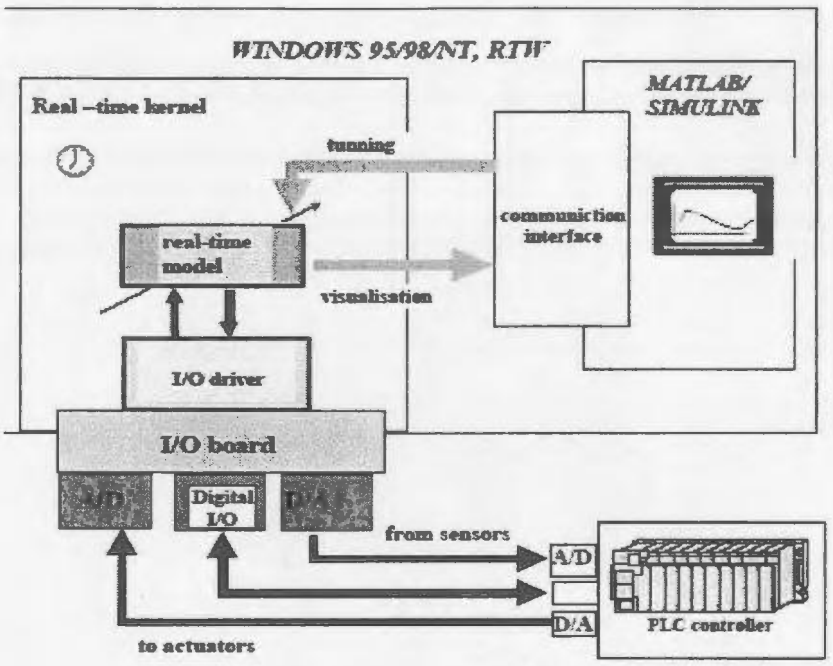

Figure 2-3: HILS using a PLC, a real-time OS and a Matlab/Simulink Process Model [3]

2.1.3 Hardware-In-The-Loop-Based Verification of Controller Software

Another successful attempt has been made with hardware-in-the-loop simulation using a PLC. In the article "Soft-Commissioning: Hardware-in-the-Loop Based Verification of Controller Software"[4] shows another method on how do interface a PLC with a soft Discrete Event Simulator (DES) that represents a process.

This method consists of several different components:

- A simulator.
- A Software to World interface.

- I/O hardware.
- I/O device Drivers (IODD).
- A PLC.

The basic layout of the environment is illustrated in figure 2-4(from [4]).

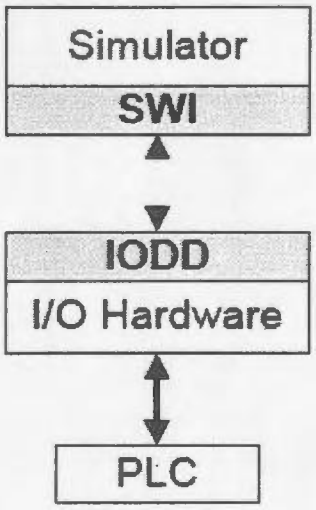

Figure 2-4: The Basic SoftCom Platform [4]

The IODD is the internal link to the I/O cards in the simulation environment. The interface for the IODD is defined by a library interface that is often referred to a Dynamically Linked Library (DLL). Implementation of the DLL depends on the I/O hardware that is being used. Each DLL is written in C/C++ and must be coded accordingly to the specifications of the given I/O hardware. Also, the IODD must be able to support more then one library at the same time to be able to establish links to different I/O cards if needed.

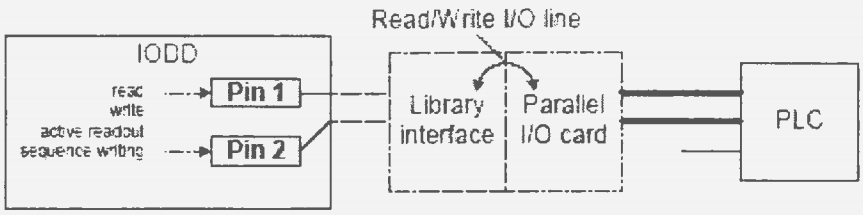

Figure 2-5: Function of the IODD [4]

The IODD internal representation of a PLC input or output is called a "Pin". A "Pin Object" permits the definition of complex access functions such as triggered reading and writing. Figure 2-5 (from [4]) illustrates the basic function of the IODD.

The purpose of the software interface (SWI) is to provide a communication interface between the simulator and the SoftCom [4] system. The SWI's implementation depends on the simulator's approach of providing access to its variables and objects.

For this SWI, the designer used a DLL interface in order to link the SWI with the simulator. The reason DLL is used is because DLL is created by C++ software and gives the programmer a great deal of flexibility when creating the interface, rather than using a Visual Basic Application (VBA) which is easier to use, but does not have the flexibility required. The DLL interface defines the routines to interact with the simulation.

Other components of SoftCom include the Virtual I/O System (VIOS) and the SoftCom manager (SCM). The VIOS was developed to provide a module that has signal processing. This relieves the SoftCom system of doing low-level tasks such as logical or mathematical evaluation of signals. The SCM is the manager of the entire

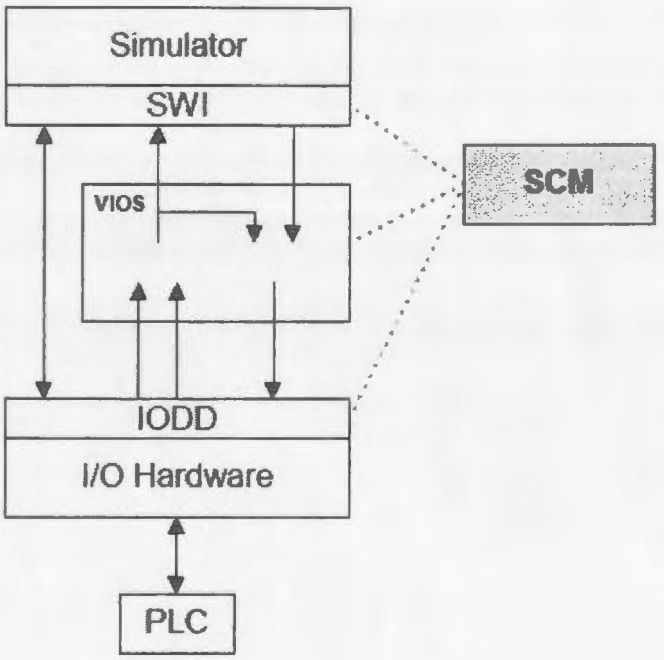

Figure 2-6: Detialed Architecture of SoftCom System [4]

system. It maintains a link to all elements of the SoftCom system for configuration and runtime control. A more detailed version of the architecture of the SoftCom simulation environment is illustrated in figure 2-6 (from [4]). It contains both the VIOS and the SCM along with other previously defined components.

The communication protocol is responsible for data exchange between the different members of SoftCom. The communication protocol that is used is based on a protocol set on top of TCP/IP. Using TCP/IP allows the system to run on different computers with different operating systems. This system is demonstrated in figure 2-7(from [4]).

One major difference with the between the SoftCom design and other hardware in the loop solutions is the use of real-time operating systems. The designers of the

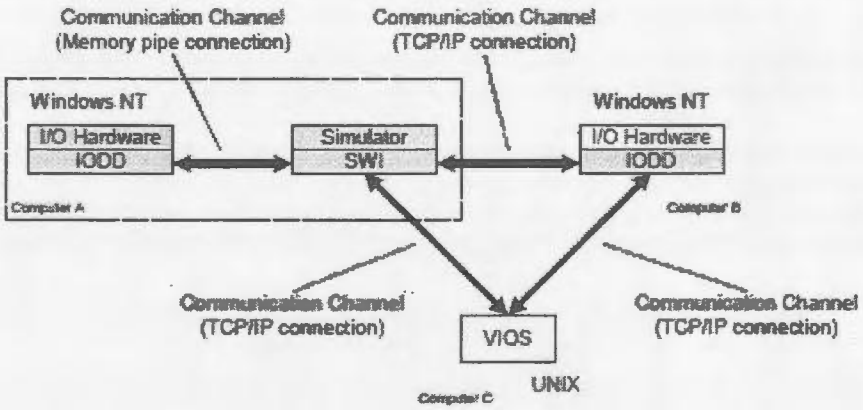

Figure 2-7: Communication Between Components Using TCP/IP [4]

SoftCom system decided to use regular operating systems because it was designed to have the ability to use commercial simulators and conventional I/O hardware. Most commercial simulators use regular operating systems such as UNIX and Windows and do not support real-time operating systems. Where this is true an event within a non real-time OS will always complete at run time and not at real-time. If a real-time operating system were to run on top of the non- real-time operating systems the real-time OS will be dependent on when the other OS is finished its process. This shows that the real-time OS will not be much of an advantage when running commercial simulators and conventional I/O hardware.

2.2 Distributed Systems

As defined previously, a distributed system is a collection of independent computers that appears to its users as a single coherent system [6]. This section will give some

background with distributed systems, distributed simulation, and distributed, real-time systems.

2.2.1 Dynamic Systems Control and Distributed Simulation

The publication, "A Synthetic Environment for Dynamic Systems Control and Distributed Simulation"[12] discusses how HILS and Man-In-The-Loop(MIL) integration gives the system under test a valid test environment in order to perform proper validation on the system. This research software provides a new synthetic environment for simulation and control synthesis of dynamic systems.

The main goal of this project is to implement a high performance simulation environment with flexibility and reusability of user components. With this achieved, the skills needed for such a prototype can be reduced to visually building system parts with tools such as Simulink which are connected to real world components with given I/O devices. As well, using the Internet protocol (IP) allows mixing heterogeneous simulator components. General communications are in Simulink and tools usually use TCP/IP or UDP/IP and make it fast and easy to create a distributed simulation environment. Real-time requirements can be transformed into speed requirements since every PC has a single task, and real-time synchronization can be performed with local real-time clocks or via the intercommunication system.

In order to perform HILS and Man-in-the-Loop(MIL) integration over a distributed network, a need for a fast and reliable communication link between the compo-

nents within distributed system. For example, you need a system to acquire the input data and generate synthetic versions of the outside world, a system to perform the model simulation, and a system to perform the control of the entire system. Hence, an array of computers connected via a network can be used to complete multiple, simultaneous tasks. As well, a deadlock free communication protocol must be present to allow for real-time synchronization between the simulation entities.

In order to select a communication network that is suitable for real-time distributed simulation, there are two issues that have to be recognized. First, is the speed of the network. The network must be fast enough to deliver information to a waiting system before the real-time deadline has expired. If the deadline expires, the information is no longer valid. The other issue is reliability where the information sent from one system is received by a waiting system without loss of data. In this article, the two networks that are discussed are TCP/IP and UDP/IP.

Both TCP/IP and UDP/IP are Internet protocol (IP) networks with several different properties. TCP/IP is a safe communication protocol that guarantees delivery and ordering queuing of transmitted packets. The problem with this protocol in a real-time distributed environment is the protocols requirement to establish a virtual communication link before data transmission. This requirement may lead to deadlock if the various components of the simulation perform initialization procedures in an unsupervised fashion. UDP/IP is a faster protocol than TCP/IP, but does not guarantee data reception. This could cause valuable data that is crucial to the simulation

to be lost, causing it to fail.

You must select the appropriate protocol for a particular application that the distributed system is being used. If network speed is essential and data loss is acceptable, UDP/IP would be a good choice. If data has to get to the next point and you are sure that the initialization of the connection will not cause deadlock, then TCP/IP would be the best choice.

To avoid deadlock in real-time distributed systems, necessary components of the system must be synchronized. Real-time synchronization is normally done by the use of Real-Time Clocks [12] (RTC). RTCs can be localized on every system or on a subset of the systems. It can also be performed with one, shared RTC only. Synchronization can be either explicit or implicit. Explicit synchronization occurs when there is a communication handshake. Implicit synchronization occurs when communications are between components of a simulation loop or when the communicating elements are all synchronized with their RTCs or with multiple synchronous RTCs.

The real-time distributed system project that is discussed within this article is DynaWORLDS[13]. This project is an attempt by the Department of Electrical Systems and Automation at the University of Pisa to build a low cost, comprehensive, distributed simulation system. This system is consists of a MATLAB toolbox and a C library that allows the implementation of a network of heterogeneous simulation systems. The network connections that are used are both the UDP/IP and TCP/IP protocols, but the same data stream can be sent on any transmission channel by

coding the proper device drivers that lets this distributed system have the ability to add any component that supports these protocols. In an effort to avoid deadlock issues, the designers of the system have implemented a deadlock-free protocol to allow for safe operation.

The integrated framework for scene design, object animation, and control panel design can create real world environments. The visual aspects of the system can be designed by the means of 3D objects that are imported by commercial CAD files. A control panel can then be designed interactively on-screen using output devices such as camera views, various instruments, and light indicators.

This work shows that it is possible to create a low cost, reliable, and flexible real-time distributed simulation system given all dynamic components, appropriate input devices, a world environment creator, and a fast and reliable communications network that provides real-time synchronization between the modules within the distributed system.

2.2.2 Remote Controller Design of a Networked Control System

The publication "Remote Controller Design of Networked Control System using Genetic Algorithm[14]", discusses the need and uses of networked control systems in modern control system design. It also discusses that time delay across a network can effect the performance of the system. The data transmitted across an industrial

network such as Profibus can be classified into two groups: real-time data and non-real-time data. Non real-time data does not have strict time restrictions on their delays during data exchange. Real-time data, on the other hand, does have strict time limits and the value of the transmitted data decays as time progresses. Real-time data can also be broken down into periodic and asynchronous data depending on the periodic nature of the data generation.

On many industrial networks, real-time data and non-real-time data share a common network even though they have different requirements on communication. Non-real-time data needs assurance of delivery without error while real-time data are mostly concerned with the time it takes to reach its destination.

This publication shows us how a Genetic Algorithm (GA)[5] is used to find PID control parameters for a control system over a Profibus-DP[15] network.

There are several factors discussed here that contribute to delay between components in a given system. Under complete system synchronization among the processes in the network, time delay is caused by the process time of each process and the polling time of the network (Profibus-DP). Complete synchronization can be difficult to achieve at all times. This lack of synchronization among the processes can also lead to excess delay do to poor synchronization of processes within the system[14].

The experimental system, as shown in figure 2-8(from [14]), consists of a controller and three DC motors that are all connected by Profibus-DP. The GA then determines the PID parameters and the system runs accordingly. The system is then tested

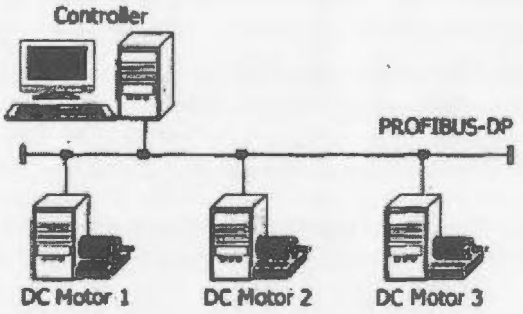

Figure 2-8: Experimental Platform for Networked Control System [5]

"directly" without the use of Profibus-DP and the GA then again determines the PID values. The result of this is that PID values are different and illustrate that network delay caused by Profibus-DP decreases the performance of the control system and the PID parameters must compensate for the system delay.

2.2.3 Network Design Consideration for Distributed Control Systems

The article "Network Design Consideration for Distributed Control Systems[16]" discusses the impact of network control and network architecture and the effect that it has on an entire control system. They discuss the quality of performance of the control system due to the quality of service of the network that is being used. With quality of service of the network being a major issue, time latency and the factors that cause time latency become important.

When dealing with a distributed control system over an industrial network, time

latency is a major issue that must be dealt with. Time latency has several main sources: Pre-processing time, waiting time, post processing time, and transmission time. The sum of all of these delays is the total delay within a system.

Pre-processing time is the time needed at the source node to acquire external data and encode it into the appropriate data format. This depends on the devices hardware and software and often, this time delay is constant or negligible. Waiting time at the source nodes is when a message is waiting in the sender's queue and could be blocked from transmitting by other messages on the network. Depending on the amount of data a node must send and the present amount of network traffic, the waiting time delay may be significant and is not constant. Post processing at the destination node is the time taken in order to decode network data into physical data format and send the data to an external environment. Like pre-processing time delay, post processing depends on the devices hardware and software. Lastly, transmission time delay is the time taken to send the data from one network node to another. This delay is the easiest to determine in most industrial networks.

When designing a network control system the factors of time delay must also be considered, however, the actual performance of a given control system must be considered. As illustrated in [16], sampling period does not affect continuous control systems where as in digital systems, the smaller the sampling period the better the performance. For network control systems, as the sampling period gets smaller the performance gets better until a threshold point is reached where the smaller the

sampling period, the worse the performance of the control system. The reason for this is that with smaller sampling frequency, the more transmissions and data load on the network. This will increase the performance of the control system until the threshold point is reached where the network becomes overloaded and the wait time increases significantly for each node causing a greater delay then if there was a greater sampling period.

The idea of delay and how it effects control systems is also present in the paper "Stability of Networked Control Systems: Explicit Analysis of Delay"[17]. They state that when the sampling period is small, the system can tolerate a delay up to one sampling period. As the sampling period becomes larger, the bound for the network delay becomes smaller. If the network delay is greater then the sampling period, data will be missed because the controller or the process could receive information at an instant where the data is no longer valid and new data that had just been sampled is now the valid data. This illustrates that the data sampled will always be new and the correct data at that particular instant in time.

2.2.4 Streamlining Real-Time Controller Design

The optimization of controllers within a real-time controller design can be a challenging task since the limits of the delay exists. The article "Streamlining Real-Time Controller Design: From Performance Specifications to End-to-End Timing Constraints"[18], discusses how there is a gap between control systems theory and

real-time scheduling theory. Such a gap makes it difficult for control engineers to take advantage of the advances in real-time scheduling. In an effort to establish integration of these two fields, this paper discusses an algorithm called period calibration method (PCM). This method takes a set of tasks along with their end-to-end timing constraints and derives a sampling period and a real-time deadline for each task.

The experiment that was performed is that a simulation consisting of a simulated controller on one PC and a plant simulator on another PC. Two variables were examined, the sampling period and the time latency. For the first test, the sampling period was fixed and the time latency was varied. The result was that when the added time latency, the maximum overshoot of the control system increased, resulting in a degradation of the control system. The second test had fixed time latency and a variable sampling period. The result was as the sampling period increased, so did the maximum overshoot, resulting in a degradation of the control system. The results of both of these tests shows the need for additional design parameters to be considered when designing a real-time control system.

With the effects of sampling period and time latency on the performance of the control system, the PCM algorithm can be used along with real-time scheduling theory to determine real-time deadlines and appropriate sampling periods for each task.

2.3 OPC

Object Linking and Embedding (OLE) for Process Control (OPC)[19] is the technological basis for the easy and effective link for automation components with field devices by the use of fieldbus communication. It also provides the ability for integration of information systems in order to analyze the system at hand.

During Microsoft's development of Windows NT, Distributed COM (DCOM) was developed as a continuation of OLE technology. When Windows NT was overwhelmingly accepted by industry, technologies such as HMI, SCADA, and DCS systems were made available for Windows NT.

With increased distribution of their products and growing number of communication protocols and fieldbuses, companies had to create countless drivers for each type of system. This turned out to be very costly to industry and a solution was necessary. In 1995, an OPC task force made of several key companies met to develop a standard to access real-time data under a Windows operating system. In 1996, the OPC Specification Version 1.0 was made available. Today the OPC task force has grown significantly and is called the *OPC Foundation*[19].

As OLE, OPC is a client/server application. Since OPC is based on COM technology, programming languages such as C++, VB, and scripting languages have access to the data produced by OPC.

The main block of an OPC server is the Data Access Server. This server provides Data Access Clients with access to different data sources. The data access server

in this project is the Automated Solutions Inc.[20] serial Modbus Server. Each data access server has a hierarchy established within it. The highest level within the hierarchy is the OPCServer object and is the root object for all other objects in the data access server. The next two levels of objects in the data access server are OPCGroup objects and OPCItem objects. Figure 2-9 shows the hierarchy of the OPC objects.

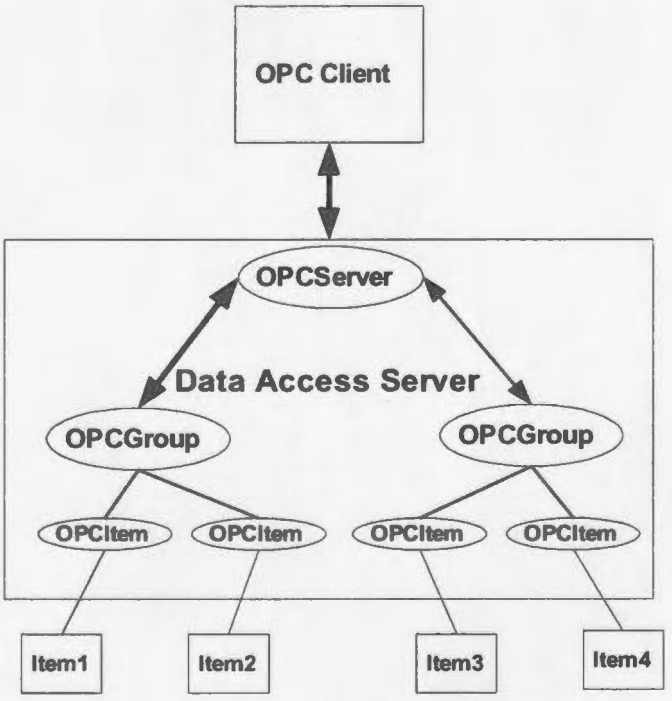

Figure 2-9: Hierarchy of the OPC.

The OPCGroup objects are simply a collection of OPCItem objects. The reason for this is to keep groups of relevant item objects together. This makes it easier when the user is trying to find a certain object within the data access server. The OPCItem objects represent actual items in the physical world. For example, an OPCItem object

may represent a location in the memory of a PLC where the data in that location is important data and needs to be accessed.

2.4 Concluding Remarks

This chapter gave a set of background literature in the areas of HILS, distributed systems, and OPC. The rest of this thesis will discuss the use of these three topics and how they are linked in order to study HILS over a distributed system.

Chapter 3

Design and Implementation of a Hardware-In-The-Loop Distributed Simulation

Presently, there is no technology available to interface the process simulation software tool HYSYS[21] with a real hardware controller over a network. One of the objectives of this research project is to design and implement a system where a HYSYS process simulation can use real hardware controllers over a Wide Area Network (WAN). Once implemented, this system will be used to study the effect of time latency on the control system when Hardware-In-The-Loop-Simulation (HILS) is performed over a WAN with significant time latency. This system includes a simple HYSYS simulation, an interface to access the data within the HYSYS simulation, communication software

to send the data over a WAN, a PLC with appropriate control software, an operator console, and an interface to the data within the PLC.

This chapter will discuss the design and implementation issues and challenges that occurred for the development of the components within the system and integration of the entire system.

3.1 An Overview of the System

The system as illustrated in figure 3-1 consists of two PC's and one Modicon PLC. The first PC in figure 3-1 contains:

- The Automated Solutions OPC server[20] for serial Modbus[22].
- The Automation .dll (AutoDA.dll) file that enables the user to create their own OPC clients using Visual Basic (VB)[23].
- A TCP/IP client dll called HysysPDClient.dll that enables communication to other systems over a TCP/IP network.
- A control wrapper created in C# .NET[23] called OPCWrapper.exe that takes the automation dll and combines it with the TCP/IP client dll in order to read and write data to a PLC via an OPC server.
- A Labview[24] controller to substitute a software controller instead of a PLC for testing purposes. This can also be used as an operator user interface for the

system.

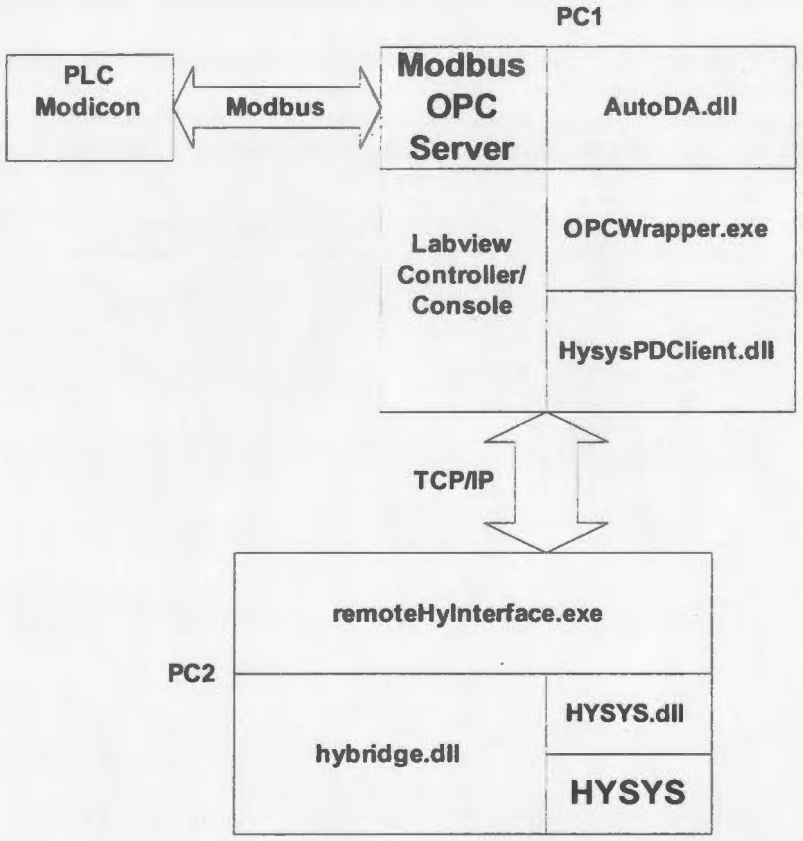

Figure 3-1: System Overview of HILS

The second PC in figure 3-1 contains:

- The HYSYS process simulation software and simulation.
- The automation dll (HYSYS.dll) that enables VB to access objects and data from a HYSYS simulation.
- The hybridge.dll file that is created in VB in order for other software tools, such as C# .NET, to access the appropriate objects and data within the HYSYS

simulation.

- A TCP/IP server called RemoteHyInterface.exe that enables communication across a TCP/IP network.

- A Labview user console to operate and observe the simulation.

This section gives a preview of the overall structure of the HIL distributed simulation system. Later in this chapter, the design and implementation of each component, along with the integration of each component will be discussed.

3.2 HYSYS Automation Interface

Automation is basically defined as the ability to drive one application from another. For example, if product A decided that it would be beneficial to give access to certain objects within itself, thereby making the objects available for automation. Since product B has the ability to access objects that have been enabled for automation, it can access the objects and the data that is available in product A[25]. HYSYS has the ability to use automation by giving access to almost all of its simulation objects and data.

In the early product planning stages, the HYSYS development team had a vision to begin exposing objects. This makes HYSYS a very powerful and useful tool in the design of hybrid solutions. Since access to an application through Automation is language-independent many different software languages such as C++ and VB can

access the data within HYSYS. However, HYSYS only supports two languages for automation; VB and its own Macro language[21].

Automation is a standard based on Microsoft's Component Object Model (COM)[23]. COM is the basic building block of automation. It gives software the ability to access or hide data within itself to an external application. The interface is represented by the use of a dynamic link library file (dll). The external application references the dll file into its development environment and then having access to all of the COM objects that belong to the application that contains the dll. This is described in figure 3-2 where it illustrates the use of HYSYS COM objects, the dll file, and external applications using a dll interface.

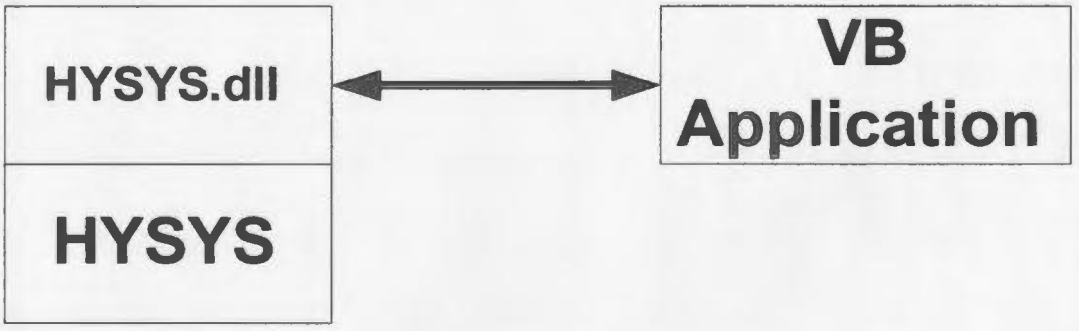

Figure 3-2: Interfacing HYSYS with a VB Application using COM and dll

Figure 3-2 is the HYSYS automation component of the entire system as described in figure 3-1. VB creates a wrapper COM object so other applications can access objects and data within HYSYS.

Automation evolved from what was once called Object Linking and Embedding (OLE)[19]. This allowed the user to take a particular object such as a spreadsheet

and embedded it into another object such as a text document. Changes to values in the spreadsheet would automatically be updated in the text document. This is a very powerful technique and is the basis that automation was founded upon. With the use of automation, an external tool can collect and change data within software that supports automation giving that tool the power to take advantage of other tools in order to give an optimal result for its operation.

HYSYS automation gives us full access to all simulation COM objects, however, finding the appropriate object to use and which method or data to use to get the desired result is a major task in itself. AspenTech provides very little documentation on the available objects that can be used. The only way that the objects can be located is by the Object Browser. This object browser does give information about each COM object within HYSYS, but finding the appropriate object may take some time due to the lack of documentation. Once the proper object is found, it is still difficult to use the method or data within the object properly because of the lack of description of the object within the object browser.

Either VB or the HYSYS macro language can only access the automation COM objects within HYSYS properly. Other languages, such as C# .NET, cannot access these objects in the way that is necessary to perform automation. However, other useful languages can communicate with VB objects. This gives the idea of using a VB wrapper hybridge.dll to communicate with HYSYS automation with other applications. Figure 3-1 shows that hybridge.dll communicates with the HYSYS.dll and

is the VB application in figure 3-2.

This wrapper will be an interface between HYSYS and the external application which is created in VB. In order to implement this interface, the objects and data that need to be accessed must first be identified. There is not a need to access all of the objects within HYSYS automation, only the objects that are needed for the particular application. Once this is identified, then a class is created that has public functions that read and write to the necessary data within HYSYS.

Once this wrapper class is created, compiled, and tested it is then converted into a dll file. The reason for creating another dll file is so other applications can use it instead of the HYSYS automation dll file. For example, C# .NET cannot use the HYSYS dll automation file directly. The wrapper class is then created in VB with interfaces that a C# application would require and then a dll file is created from the VB class. When the new wrapper dll file is created, it is then imported to the C# .NET project and then it can use the class and functions within that class which correspond to the COM objects and data within HYSYS. Figure 3-3 illustrates the architecture of the HYSYS COM objects, the VB wrapper dll file, and the C# .NET project that wishes to use the automation features of HYSYS. In this research project, the hybridge.dll is imported into the C# .NET application RemoteHyInterface.exe as shown in figure 3-1.

The data from the hybridge.dll COM object is data that is needed for a standard PID controller[26]. The control task to be performed is the water level in a tank with

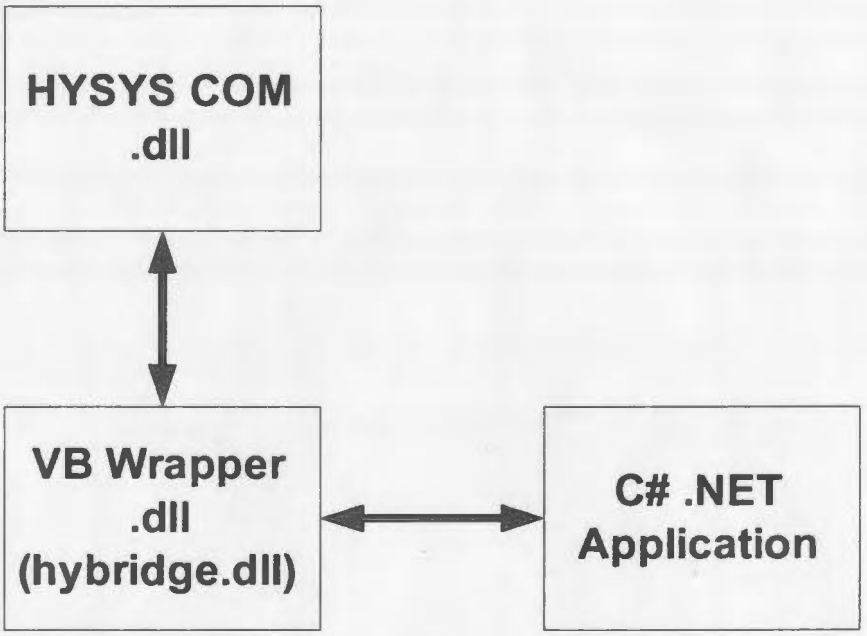

Figure 3-3: Architecture of HYSYS COM, the VB Wrapper .dll, and the C# .NET Application.

water flowing in and out of the tank. In PID control, as shown in figure 3-4, there is a PV (process variable), a set point (SP), and an output (OP). In this case, the PV is the height of water in the tank, the OP is the valve percent opening (the valve controls the flow of water into the tank), and the SP is the desired height of the water in the tank. The SP is already set in the PID controller, but there are two objects that have to be accessed in the HYSYS simulation, the height of water in the tank and the valve opening. The controller must sample the present PV, compare it to the SP, and then make the appropriate change to the OP. For the wrapper hybridge.dll, the only two objects within the HYSYS simulation that need to be accessed are the height of that particular tank and the percent opening of the valve that controls the

flow of water into that tank. This shows the data abstraction that the wrapper class has since it allows only those two objects to be accessed.

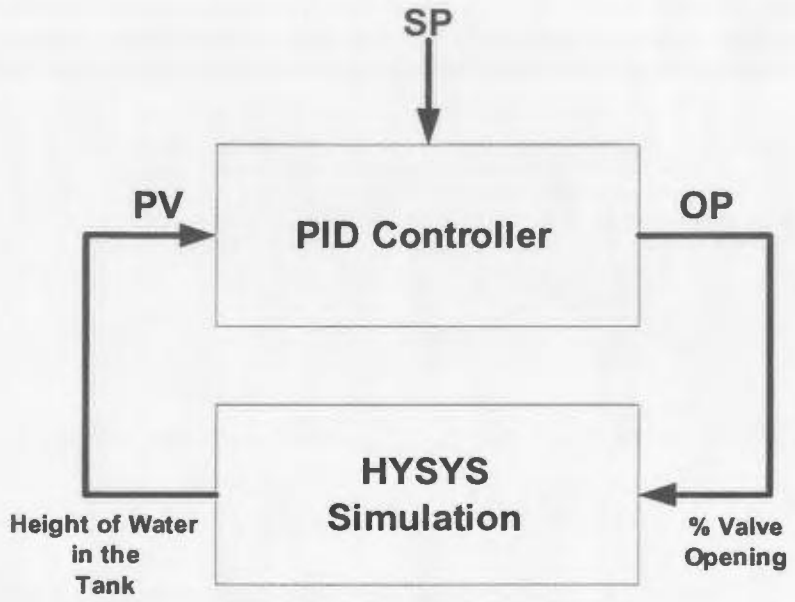

Figure 3-4: Interface Between the HYSYS Simulation and a PID Controller.

Creating the wrapper file hybridge.dll enables external applications other then VB and the HYSYS macro language the ability to access objects and data available in HYSYS automation. It also lets the creator of the wrapper file to decide on which objects and data that it wishes to either give access or hide from external applications.

3.3 Communication Software

The research conducted in this project involves the use of multiple computers. In order to achieve multiple computer simulation, communication must be performed between

two or more systems. The goal of this project is to complete simulation over a TCP/IP WAN. Figure 3-1 shows two PC's, PC2 for the HYSYS process simulation and PC1 for controlling the process simulation using a PLC or a Labview controller over the network. In order to achieve this goal, the data must have the ability to transport between each system in a reliable and timely manner. The first solution is to use a Distributed COM (DCOM)[19] object. This could be a solution if the network was a Local Area Network (LAN), but the goal is to perform simulation over a TCP/IP WAN. This rules out the DCOM solution. In order to perform communication over the TCP/IP network, custom software has to be created. This software could be created in a number of ways. It could be created in C++, VB, JAVA, etc., however, the tool that was chosen is the C# .NET platform.

3.3.1 Overview of the C# .NET Platform

In July of 2000, Microsoft announced the release of the .NET platform[27]. The .NET platform is a new development framework that provides a new application-programming interface (API) while keeping most of the old API's that existed in previous Windows operating systems and bringing together a number of disparate technologies that were developed by Microsoft. Along with the new set of API's came along a new programming language called C#.

The .NET platform still created new API's for VB and C++, hence, the creation of VB .NET and C++ .NET development platforms, but C# was the language that

was created for .NET. C# is a language based on C++ and JAVA that is user friendly when dealing with modern programming techniques, such as object-oriented programming, and utilizes the .NET platform to its full potential. For these reasons, it is the programming language that was chosen for this project. The .NET platform also has many API's that makes multiple computer communication over a TCP/IP network an easier task then if previous platforms were used.

3.3.2 Importing COM Objects into .NET

In order to use a native COM object in .NET, it must be imported into the project as a reference object. Referencing the dll file that corresponds to the native COM object does this. For this project, the HyInt class within hybridge.dll as shown in figure 3-1 and figure 3-2 is the COM object that is imported into the .NET project and once this object is imported, it can be used by that .NET project.

3.3.3 Marshaling and Remoting

When dealing with distributed systems, objects must be able to communicate with one another. The process of moving an object across a boundary is called *remoting* [27]. Many boundaries exist at many levels, but the boundary that is most common in distributed systems is objects running on different systems that need to communicate with one another.

In order to send an object across a boundary, it cannot just be sent as a raw object,

it must be prepared and packaged so it can be sent across that particular boundary. The process of preparing an object to be remoted and sent across a boundary is called *marshalling*[27].

A *process* is basically a running application. If an object in a spreadsheet wants to communicate with an object in a VB application, they must communicate across process boundaries. Processes are then divided into *application domains* [27] that are lightweight processes that run within a process. Often, objects are required to be marshalled across both process and application domain boundaries.

Marshalling can either be done by *pass by value* or *pass by reference* [27]. When pass by value occurs, a copy of the object that wants to cross a boundary is created and then the copy is passed to the other application crossing particular boundaries as illustrated in figure 3-5. Pass by reference occurs when the object is not sent across a boundary, but just a reference of that object is passed across the boundary and the remote user uses the reference to the object instead of copied version of the object. This saves on memory and bandwidth since a copy of the object does not need to be sent across the boundary. The actual reference that is sent across the boundary is called a *proxy*[27]. A proxy is a user interface to an object, not an actual object. Across the boundary it will appear that the object has been transported, but it hasn't. Only the proxy has been sent and it references the needed data to and from the object that wishes to cross a boundary as described in figure 3-6. Pass by value and pass by reference when marshalling using C# .NET, works in a similar way as it

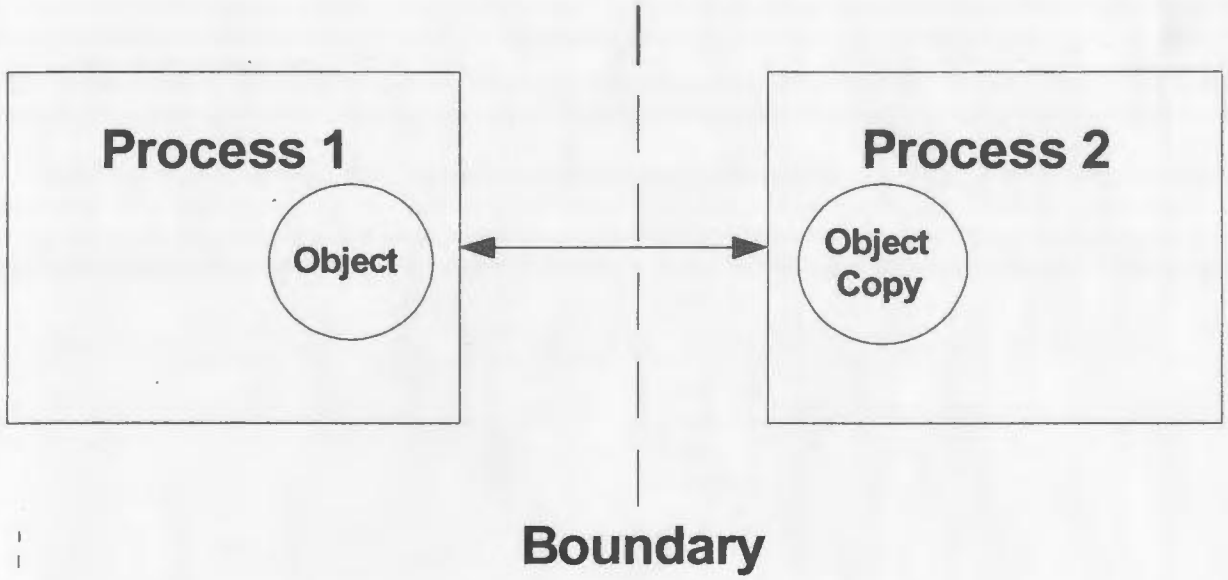

Figure 3-5: Object Marshalling using Pass by Value

does in C++[28] and other programming languages.

This project will use marshalling with pass by reference for reasons stated above. In order to do this the object that is being marshalled must be prepared for sending a proxy across a boundary. This is done in many programming languages, but it is often not trivial. C# .NET, however, makes marshalling an easier task than normal. First, the .NET platform contains the base class *MarshalByRefObject*[27]. This class gives the ability for a class to be marshalled by reference. In order for a class that has to be marshalled by reference, the class must use inheritance and inherit the class MarshalByRefObject. This gives the class that wishes to be marshalled by reference all of the properties of MarshalByRefObject and enables it to be sent across boundaries by the use of a proxy. Another component must be completed before an object can be marshalled by reference is that it must have an *interface* [27]. The

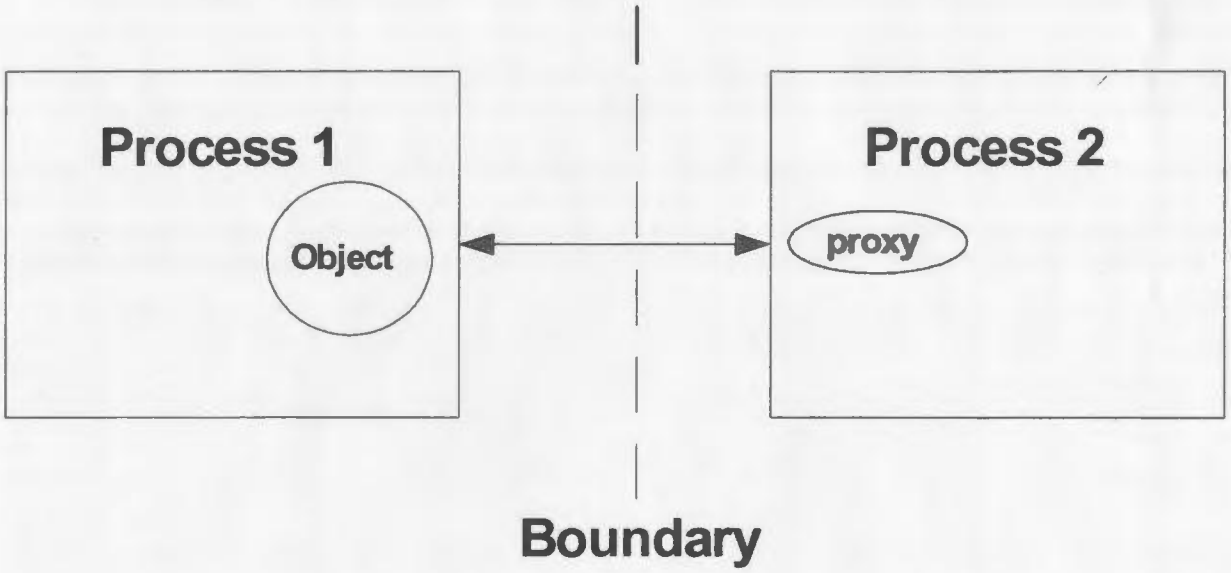

Figure 3-6: Object Marshalling using Pass by Reference

interface is a contract that guarantees to a user how a class will behave. The interface will dictate what properties of the class will be involved with the proxy. Once the interface is created, the class that is using the interface must inherit that interface. The class that is going to be marshalled by reference must have both an interface and the MarshalByRefObject and when it has both it can be sent across a boundary.

Remoting of objects, once they are marshalled is done using a client/server model. The object that has been marshalled and is being remoted to another system is done by the server application. The server application is used by the client application, for example, if a calculator object is created and a user wants to use it on another system, the calculator class will be the server and the user will be the client. One server can have multiple clients, but single clients can only correspond to one server. The server class, since it is the class that it is being marshalled and remoted, is the

class that must inherit the MarshalByRefObject and also have an interface created for it. As for communication across TCP/IP, the .NET platform has libraries that support the use of object transportation across TCP/IP.

In this project, the server class and executable called RemoteHyInterface. This server class is located in PC2 in the overall system that is illustrated in figure 3-1. This class is the class that is marshalled and remoted across the network and therefore must inherit the MarshalByRefObject class and an interface called IHysys. This interface contains only two functions related to the task; getPVLevel() and setValveOpening(). These two functions are fairly simple since that it allows the user of this class to get the level of the water in the tank in the HYSYS simulation and it lets the user change the valve opening that controls the flow of water into the tank. This gives a level of data abstraction to the client that it can only access these functions because it is all that the client needs. The class also imports the HyInt VB class through hybridge.dll so it can access the data within HYSYS. The constructor creates the HyInt object and requests the simulation path of the HYSYS simulation that is being used and then initialized the simulation for external use.

The client object that will be using the RemoteHyInterface server object is called HysysPDClient as illustrated in figure 3-1. This class will take the remote server object and use it in order to create a library on the client side. For this project, it will always be the controller system for either the Labview controller or the PLC. This client class receives the RemoteHyInterface as IHysys, its interface and proxy. The

way that this client object works is that when it is being constructed, it requests the location (IP address) of the server object. When it has the location of the server, the client receives the proxy of the server object and is able to use it. The HysysPDClient class is a class library for the remote machine and it has the getPVLevel() and set-ValveOpening() functions for applications on the remote system to use as described in figure 3-7.

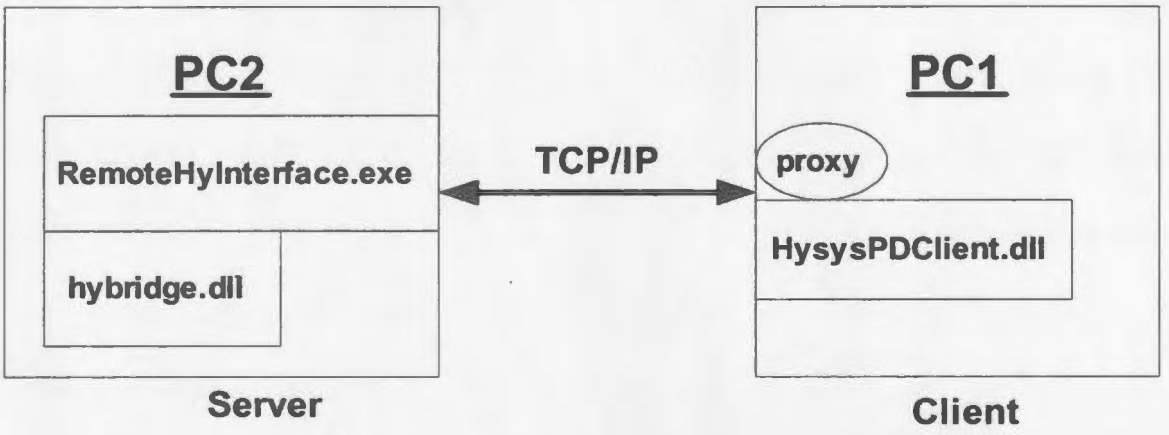

Figure 3-7: Remote Communication using Remoting and Marshalling over TCP/IP

3.3.4 Random Time Delay Generation

The overall distributed HILS system as described in figure 3-1 will be performed within the INCA lab. The issue here is that the network delay for TCP/IP within the lab is less then 10 ms between each system. In order to simulate poor bandwidth WAN conditions within the lab, time delays must be included in the test system.

The easiest place to add the time delays is within the software and can easily be

done in the C# .NET environment. The Thread.Sleep() function within the .NET platform will give you a time delay in ms that the user specifies. For example, Thread.Sleep(500) will suspend the program by 500 ms. However, using a constant delay does not model the behavior of a WAN. A WAN delay over a great distance is often unknown, what is known is that there will be a delay. A random delay can be obtained by passing a random integer into the Thread.Sleep() function.

The .NET platform also has a random number generator object where the user can specify the range of random integers that the software wishes to generate. Here a delay between 1 and 4 seconds will simulate poor WAN performance. The random number generator will then create an integer between 1000 and 4000 and this integer is then passed into the Thread.Sleep() function to obtain random delay.

This technique of generating random delay will then be added into the software where in the OPCWrapper.exe component of the system as shown in figure 3-8 and figure 3-1. The delay is generated when a read from the OP from the PLC takes place and when a write to the PV in the PLC is performed. This will simulate the delay that would be caused by a WAN with poor performance.

3.4 Labview Console and Controller

One part of this project is to include a software simulation operator console. This console will observe the actions of the simulation using Labview 7.0[24] and is shown in the overall system in PC2 in figure 3-1. Labview is a tool that can be used to create

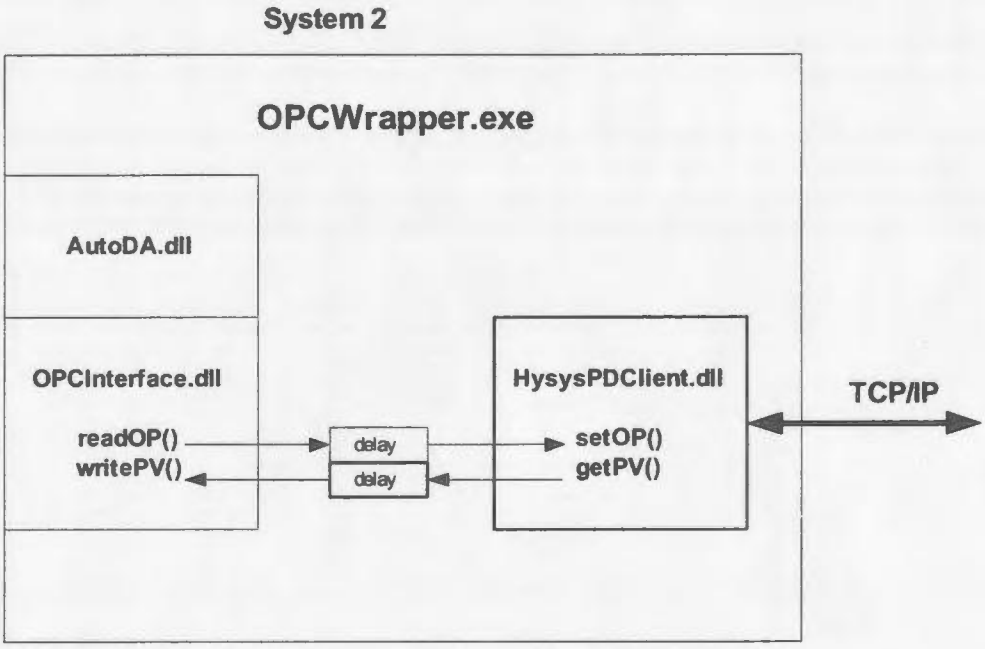

Figure 3-8: Random Time Generation in OPCWrapper.exe

PC user interfaces for the outside world. For example, given the correct hardware, Labview can be used for a data acquisition tool for collecting and analyzing data and giving an output display as an easy to create GUI. In this project, the user console will not have any external hardware to perform data acquisition, but the user will be able easily to observe the actions of the system through a GUI. Along with observing the data through the GUI, the Labview Virtual Instrument (VI) will record the data in a text file for record and further evaluation.

In order for Labview to access the data from the simulation it must access one of the objects that is involved with the simulation. Labview 7.0 has the ability to import COM and .NET objects, so importing the client library HysysPDClient.dll would be

the correct choice since this library was created for client applications to access proper data within the HYSYS simulation. The Labview console and HysysPDClient.dll correspond to one another according to the system overview diagram on figure 3-1. Once this object has been imported into the VI, it can use the functions that are associated with this object within the VI. With this object imported into the VI, Labview can easily create a GUI to observe the simulation variables and collect the relevant data.

Along with acting as a GUI, Labview can also be a software PID controller[29]. For this research project, there is a desire to compare the results of the PLC hardware simulation with a software controller and Labview can do this with its PID control module. This module is a standard VI PID control algorithm where all of the parameters of PID can be specified. Integrating this into the Labview simulation will give us a GUI to view this simulation and a text file to analyze the data later. The Labview PID controller can be used to get initial PID values for simulation and have a benchmark to compare with the HILS with the PLC. Figure 3-9 shows the Labview VI GUI with the PID control algorithm integrated into the VI.

The Labview console and controller can be placed on any other PC or system. All the system will need is the HysysPDClient.dll file and the IP address of the PC where the HYSYS simulation is taking place.

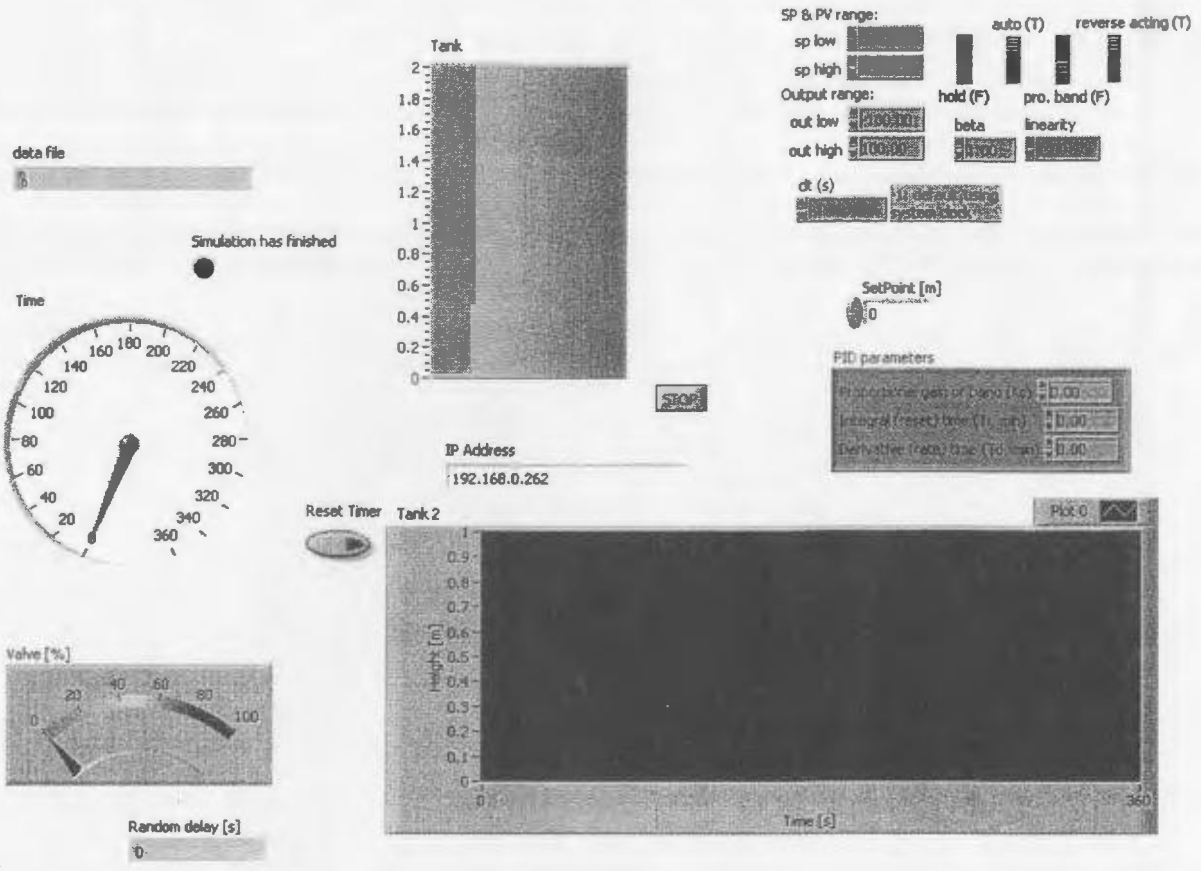

Figure 3-9: Screenshot of the Labview Operator Console with PID Controller.

3.5 The Programmable Logic Controller

According to the system overview for distributed HILS as illustrated in figure 3-1 a PLC must be connected to the system in order to perform HILS. A PLC[30] is a user friendly, microprocessor based specialized computer that carries out control functions at many different levels of complexity. The purpose of a PLC is to monitor process control parameters and adjust process operations according to a given specification.

The PLC that will be used in this research is the Modicon 170 AMM 090 00[31]. This PLC must have a PC with a RS-232 communication and a serial Modbus cable to

connect the PLC to the PC. Along with the hardware requirement, software is required to program and drive the PLC. The software that this project is using is a Modicon product called Concept V2.1[31]. INCA and Memorial University of Newfoundland, Faculty of Engineering and Applied Science provide both Concept and the Modicon PLC.

3.5.1 PLC Hardware

The PLC hardware for this research is fairly simplistic. The Modicon 170 AMM 090 00[32] requires a 24VDC power supply and the PLC must be wired properly for basic usage. For this project, the wiring will be for analog inputs and outputs. Figure 3-10 will only show the power wirings since the experiment will not use any of the physical I/O's.

As for communication, the Modicon 170 AMM 090 00 has a communication module as part of the PLC that is capable of communicating on a serial Modbus network or a Modbus plus network. This project will be using serial Modbus network communication.

3.5.2 PLC Software

The software that corresponds to the Modicon PLC is Concept. This software usually can be used to program a PLC in one of two ways: Latter Logic and Functional Block Diagrams[33]. Here, a Functional Block Diagram was used to write control software

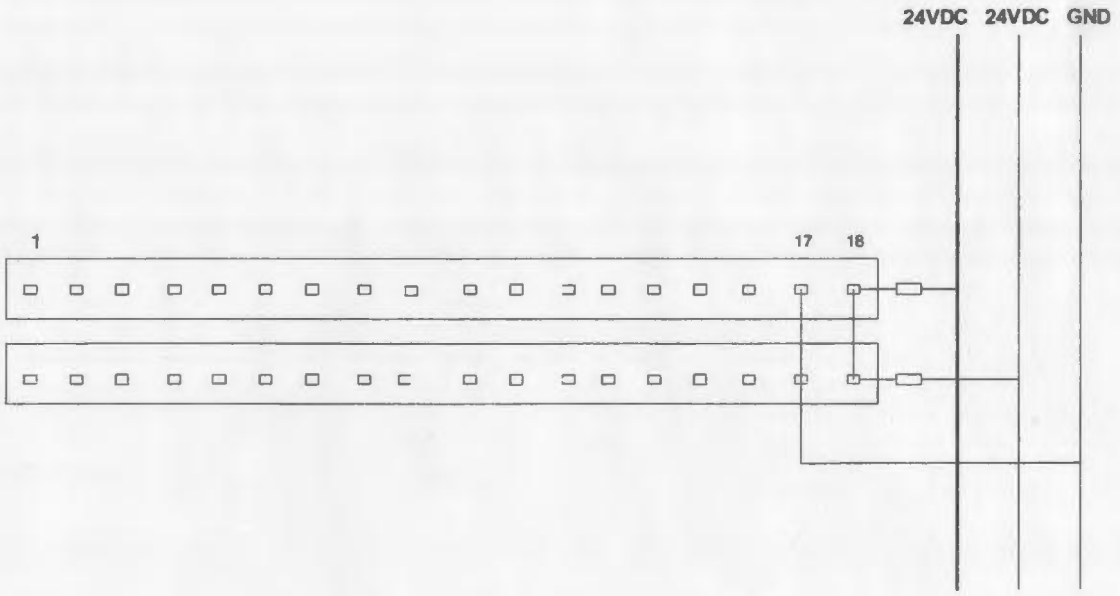

Figure 3-10: Basic Wiring Diagram for Modicon PLC

for the PLC. Functional Block Diagrams are much easier to create control programs then Latter Logic diagrams since a Functional Block Diagram is basically a block diagram with a library of blocks that can be used in order to create a control program. For example, the controller that is needed for this research is a PID controller. Concept Functional Block Diagram library has a PID block that can be used. This makes the task of creating a control program for this project a trivial one.

The PID function block must also be connected to blocks that represent the physical I/O of the PLC. Concept does have such blocks to sample data from the physical input and blocks that send data to a physical output. Since both the input and output of the controller are analog signals for this project, the input and output blocks must correspond to the analog input and outputs of the PLC. Concept does have

separate function blocks for discrete and analog I/O.

Since we are only concerned with the analog I/O of the PLC, we will only concern ourselves with the RAM locations of the analog I/O. The analog input registers are located in the 3xxxxx address space and the output analog registers are located in the 4xxxxx address space. In order to perform HILS, the data in the RAM corresponding to the memory locations of the physical analog I/O must be accessed through the Modbus network.

One major problem with reading and writing to these memory locations in the RAM of the PLC is that the 3xxxxx memory locations are read only. This is a problem because this is the analog input register and in order to perform HILS we must be able to write to the analog input through Modbus. To solve this issue, we need to bypass this memory location and write directly to the PID block in the control program. To complete this task, a register (memory location) that can be written to must be connected to the input of the PID block. Creating an intermediate register in the 4xxxxx memory block within the RAM can do this. The 4xxxxx memory block, even though output analog registers, have both read and write properties. If this is connected to the input of the desired PID block, it can be written to over the Modbus network. Figure 3-11 shows how this can be done using FBD.

In figure 3-11, the analog input register analog_in_reg is the analog input register that corresponds to the 3xxxxx memory locations and hence, the physical input. In a real world application, data on this register will be converted to a REAL type using

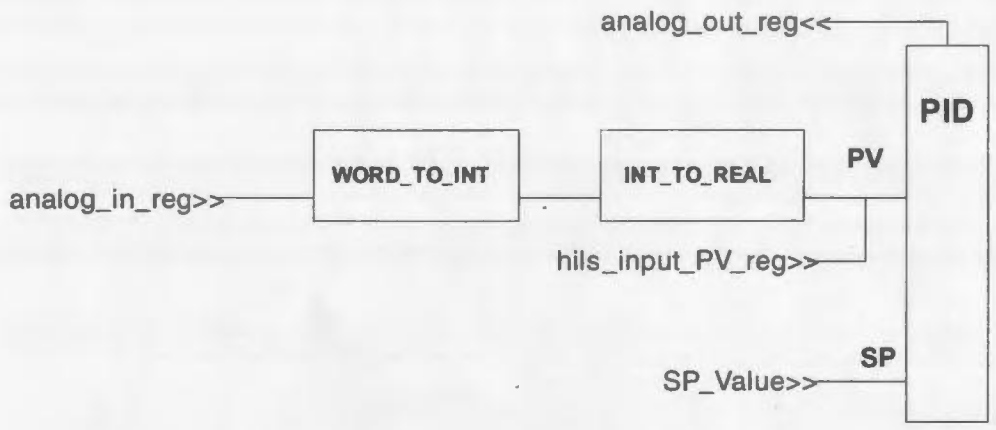

Figure 3-11: FBD with Intermediate Register for the Analog Input

the two function blocks and then the PID block can process the data. In this case, there is an intermediate register called hlis_input_PV_reg. This register is in the 4xxxxx memory block and can be written to by over the Modbus network. This register is already a REAL type and can be directly connected to the PID block, bypassing the necessary conversions that the register's analog_in_reg data had to perform.

Having the ability to read and write to the analog inputs and outputs with by using intermediate registers in the 4xxxxx memory block instead of read only registers in the 3xxxxx memory block enables the use of HILS by using the Modbus network. The only downfall is that it does not test the effects of the analog to digital, digital to analog converters, and actual physical I/O's within the PLC.

3.6 The OPC Interface

In order to access the data from serial Modbus, an interface must exist. This can be done using an OPC serial Modbus server. First, a serial Modbus server must be acquired and the vendor that was chosen was Automated Solutions Inc.[20] because of cost, availability, and simplicity. This server is a component of the distributed HILS system shown in figure 3-1. This OPC server also supports Data Automation 2.0[19]. This is needed in order to interface the data within the OPC server with the rest of the world by the use of Visual Basic.

Connecting to the serial port and PLC device can be a trivial task. Once this is done, the group object(s) can be declared as well as the item object. For the item objects, all that is needed is the addresses within the PLC where the relevant registers are declared and the OPC native type of each of the items. The OPC native type is type defined by OPC that are of a "VARIANT"[19] type meaning that they can be transformed to any other type necessary. For these items we will be using the OPC type VT_R4[19], which is variant type real4. VT_R4 is represented as a real 4-byte (32 bit) word that can be assigned to any other type. Once the OPC server has been initialized, it can be tested using Automated Solutions test client. This test client can read and write to all of the declared items within the OPC server.

OPC Data Automation 2.0 will be used with VB in order to interface the OPC server data with the rest of the system. The Automated Solutions OPC server has a file called AutoDA.dll. This file is the OPC Data Automation 2.0 interface class.

Using this class, VB can read and write data to any of the items that were defined in the OPC server. The code for reading and writing to the items within the OPC server was based on the sample code in the *OPC Data Access Automation Specification* [34]. The VB class was written as a dll with appropriate functions available for reading and writing to the proper items in the OPC server. For this research project, the proper items are references to the 4xxxxx memory block registers in the PLC that are responsible for the input and output to the PID control block as previously discussed in figure 3-11.

The dll that was created is called OPCInterface.dll as shown in figure 3-1, and it is then included into an OPCWrapper class that also included the HysysPDClient.dll. The OPCWrapper class is then compiled into an executable file to run on the system that contains the OPC server and the PLC connection.

The OPCWrapper class is created in C# .NET and uses both OPCInterface.dll and HysysPDClient.dll to interface the OPC data, that is from the PLC, to the RemoteHyInterface.exe server, shown in figure 3-7, that is located on another system across a TCP/IP network.

For this research project, the input to the OPC server will be the height of water in the tank in the HYSYS simulation and the output of the OPC server will be the valve percent opening of the valve that controls the flow of water into the tank. The height of water is the process variable, (PV), and the valve percent opening is the output (OP) for the PID control block within the PLC.

1. The OPCWrapper first reads the initial value of the output of the PID controller (OP) using the OPCInterface object.

2. The OP value is then written to the setOP() function in the HysysPDClient and then sent across the TCP/IP network to the RemoteHyInterface executable.

3. The PV is then read from the HysysPDClient that has come across the network from RemoteHyInterface.

4. Finally, this value is then written to the write_PV function within the OPCInterface object.

This procedure is then repeated until the simulation has stopped. Figure 3-12 illustrates the architecture of the OPCWrapper class.

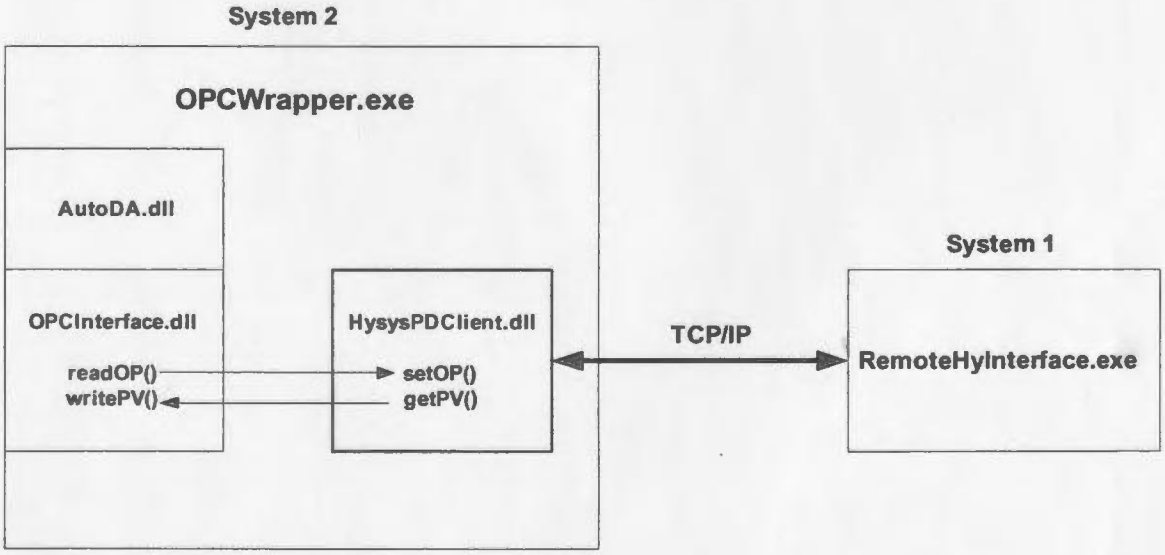

Figure 3-12: Architecture of the OPCWrapper class

As discussed before in figure 3-7, the RemoteHyInterface.exe will send and receive the proper data from the HYSYS simulation to and from the HysysPDClient object over the TCP/IP network. This completes the HILS system as shown in figure 3-1.

3.7 Concluding Remarks

This chapter gives an illustration of the design and implementation of the HILS environment using a distributed system over a WAN. This gives a platform to analyze the effects of time delay within the experiment outlined in Chapter 4.

Chapter 4

Design of Experiment and Analysis of Results

The study of distributed HILS over a WAN with poor performance is a goal of this research. Since the system as described in figure 3-1 has been implemented, it must be tested and evaluated. This chapter will give a discussion on the testing of the system and how performs under certain scenarios. First, it will discuss the set up and the design of the experiment and what will be the important variables that must be observed in the evaluation of the system. Next, it will discuss the results of each experiment scenario and give an analysis each particular experiment.

4.1 Design of Experiment

One of the reasons that this distributed HILS over a WAN was created is that there was a desire to study the effects of time latency to and from the controller (software or hardware) to a HYSYS process simulation over a WAN with significant delay. In order to analyze this system, we need to compare it with the same simulation on one PC without the time latency. Another issue that this research wishes to pursue is the difference between the uses of a hardware controller such as a PLC with the use of a software controller such a control algorithm in Labview.

As well as testing and analyzing time latency between a controller and a process over a WAN and comparing the difference between a software controller and a hardware controller, a combination of the two investigations should also be analyzed. This will give the experiment four test cases:

1. Labview software controller to a HYSYS simulation on one PC with no network delay.

2. Labview software controller to a HYSYS simulation on a distributed system with two PC's with random network delay.

3. PLC hardware controller to a HYSYS simulation on one PC with no network delay.

4. PLC hardware controller to a HYSYS simulation on a distributed system with two PC's with random network delay.

The test cases for this experiment are described on figure 4-1.

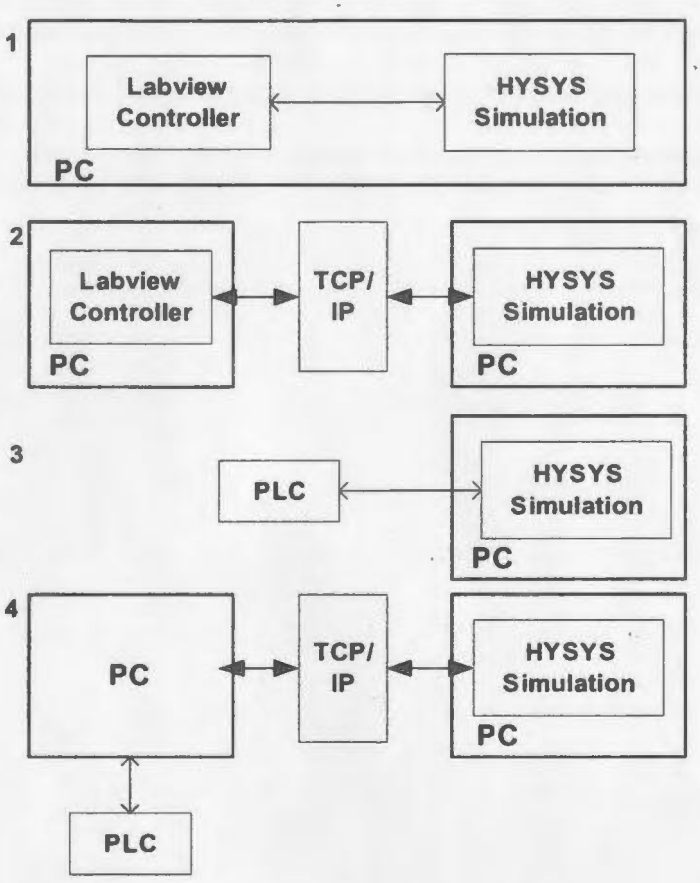

Figure 4-1: Experimental Design

4.1.1 Experimental Parameters

Before any of these tests can be executed, there are several parameters within the system that must be constant. First, a common set point that will be used for each of the tests must be selected. If different set points were selected for each of the tests, it would obviously be difficult to compare each of the simulations. Since the tank that is being observed is 2 metres high, the set point was set a little less then half way at

0.9 metres.

Second, the PID values must remain constant for each of the simulations. For this experiment, we wish to view the effect of time latency and using software vs. a hardware controller. If the PID values were changed for each of the tests, the comparison between each may not be accurate and the response may not be a desired response. This experiment needs a set of PID parameters that will give a steady response to the system. The method that is used to find the proper PID parameters in this experiment is called the Ziegler-Nichols[26] PID tuning method. To find the proper PID parameters, the Labview controller with HYSYS process simulation on one PC will be the test that will use the Ziegler-Nichols PID tuning method in order to find the PID parameters for this experiment. There are two reasons for this; the Labview software controller is easy to change PID parameters compared to the PLC and there is no delay in the system. With no delay, we can tune the controller so the system can obtain a desired response. Later tests will show what will happen to the same system with time latency.

Several other variables must also stay constant such as the PC's that are being used. The load on the PC's must be the same for every simulation, for example, no other unnecessary programs can run on the PC's that use the CPU and slow the entire simulation. Network load must also stay constant, which means that the experiment must be performed when there is little network usage.

After evaluating the Labview controller with HYSYS process simulation on one

PC using the Ziegler-Nichols PID tuning method to find the proper PID parameters and evaluating the simulation time with a given set point, the constant parameters in this experiment are:

- PID parameters:
  - $K_p = 98$
  - $T_I = 60\,s$
  - $T_d = 12\,s$
- Simulation time is 6 minutes (360 seconds).
- Setpoint is 0.9 m.
- All PCs, PC workload, PLC, and the network must remain the same throughout the experiment.

4.1.2 HYSYS Process Simulation

The HYSYS simulation that is used for this experiment is a simple, second-order system that consists of two tanks and a control valve. The valve is a linear valve[7] that controls the flow of water into the first tank and the outflow of the first tank goes into the input of the second tank. Then the water drains from the second tank as shown in figure 4-2. This system is a second order, non-interacting lag process[7].

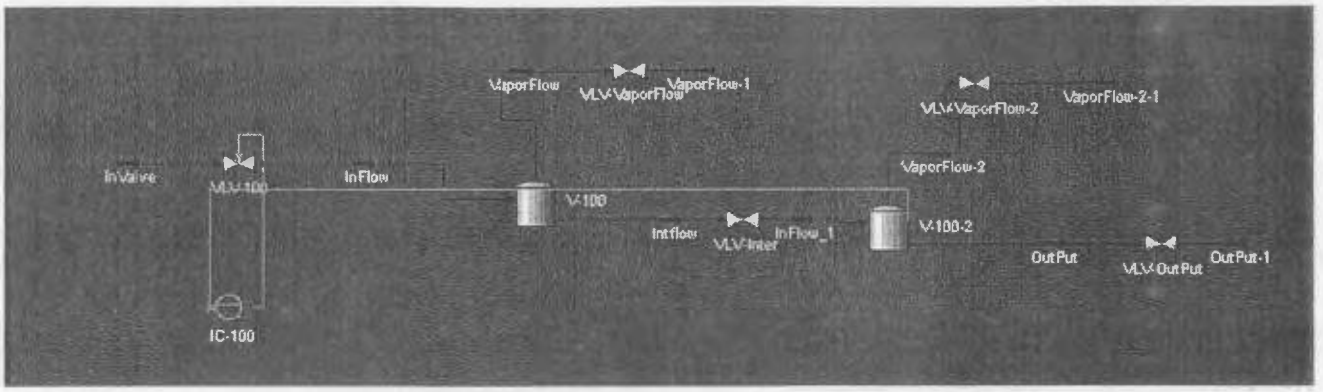

Figure 4-2: HYSYS Simulation used in this Experiment

The intermediate valves in the simulation are only there for simulation purposes. The only valve that is being manipulated is the very first valve (the valve, VLV-100, furthest to the left in figure 4-2). This simulation is run in real-time to give the effect of an overall real-time simulation.

4.1.3 Software and Hardware Controllers

This experiment will be using two different controllers. The first controller that will be used is a Labview software controller PID toolkit[29]. This PID controller is used to control any system given the proper process variable, set point, output, and PID parameters. The GUI for the Labview controller is shown on figure 3-9.

The second controller used in the experiment is the Modicon 170 AMM 090 00 PLC that is illustrated on figure 4-3. This PLC has a 24VDC power supply and a Modbus communication cable. The Modbus communication cable is connected to a PC that has a Modbus OPC server that allows external applications within the PC

to access the data within the PLC via Modbus communications.

Figure 4-3: Lab Setup for the PLC, Modbus, and OPC Server.

4.1.4 Labview Controller to HYSYS Simulation on one PC

The first test is the Labview software controller to the HYSYS process simulation on one PC as described in figure 4-4 and figure 4-1.

This simulation will run on one PC with a Labview controller and console as shown in figure 3-9 and a simple second order HYSYS process simulation with two tanks and an input flow linear control valve as shown in figure 4-2.

This test will evaluate the performance of the Labview controller with the HYSYS simulation. It will also give a benchmark in order to compare the later tests as shown in figure 4-1.

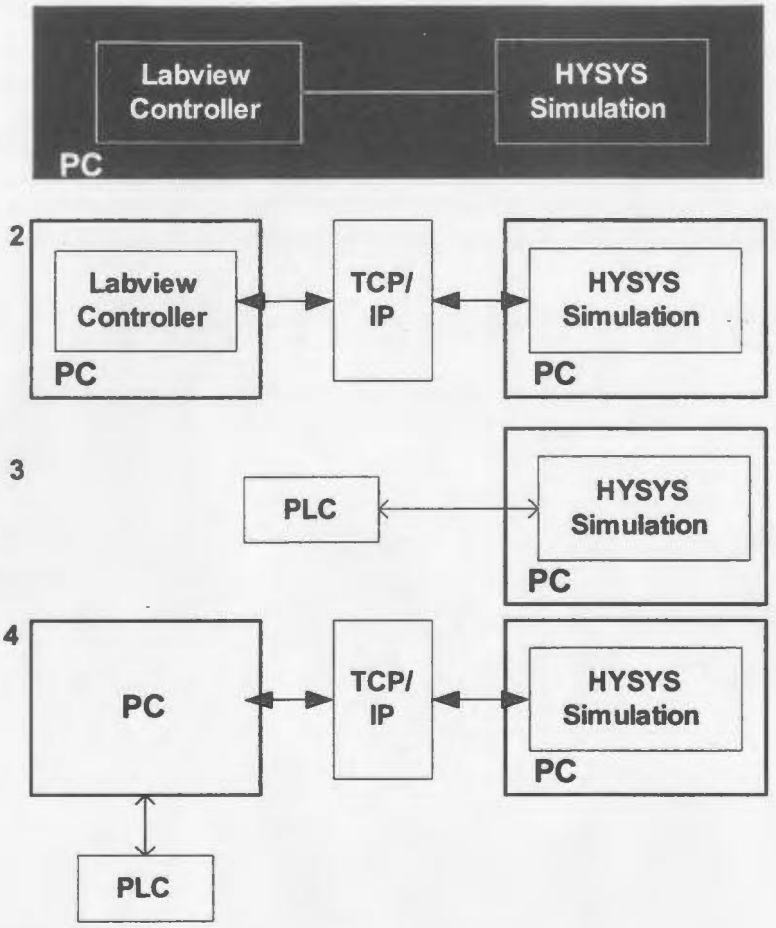

Figure 4-4: Labview to Hysys Test with One PC.

4.1.5 Labview Controller to HYSYS Simulation on Two PC's over a WAN

This test, as shown in figure 4-5, two PC's are communicating over a TCP/IP network with one PC consisting of the HYSYS simulation and the other PC consisting of the Labview controller and console. This test will illustrate the performance of using a Labview software controller with a HYSYS simulation over a network with long and

uncertain delay. The delay between each computer is random and is set between 1 and 4 seconds. We can compare this test with the other tests with special attention paid to the test where the Labview controller and HYSYS simulation are on one PC with no delay.

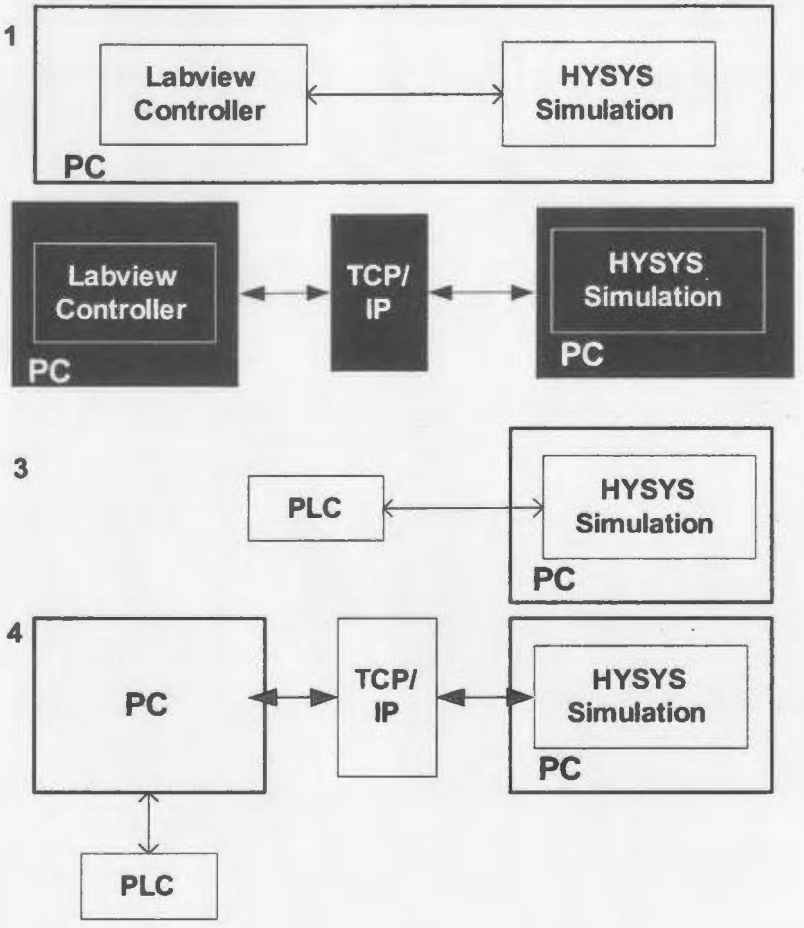

Figure 4-5: Labview Controller and HYSYS Simulation over a TCP/IP network

4.1.6 PLC hardware controller to a HYSYS simulation on one PC

This test, as shown in figure 4-6, will consist of the Modicon PLC and the HYSYS simulation all on one PC as shown on figure 4-3. The PLC will be connected to the PC via serial Modbus and the OPC server on the PC will access the data within the PLC's memory. This simulation will not have random delay between the controller and the process simulation and data will be observed and collected by the Labview console.

4.1.7 PLC hardware controller to a HYSYS simulation on a distributed system with two PC's over a WAN

The fourth test, as shown in figure 4-7, has the HYSYS simulation on one PC and the PLC and OPC server with Modbus LAN communications on the other PC. The two PC's share data via a TCP/IP network and this network introduces random delay between PC's from 1 to 4 seconds, that simulates poor TCP/IP performance which is what is desired for this test. The result of this test will be compared with the other tests with special attention to the test where there is the PLC and the HYSYS simulation on one PC. The data is observed and collected by the Labview console that is also connected to the test platform.

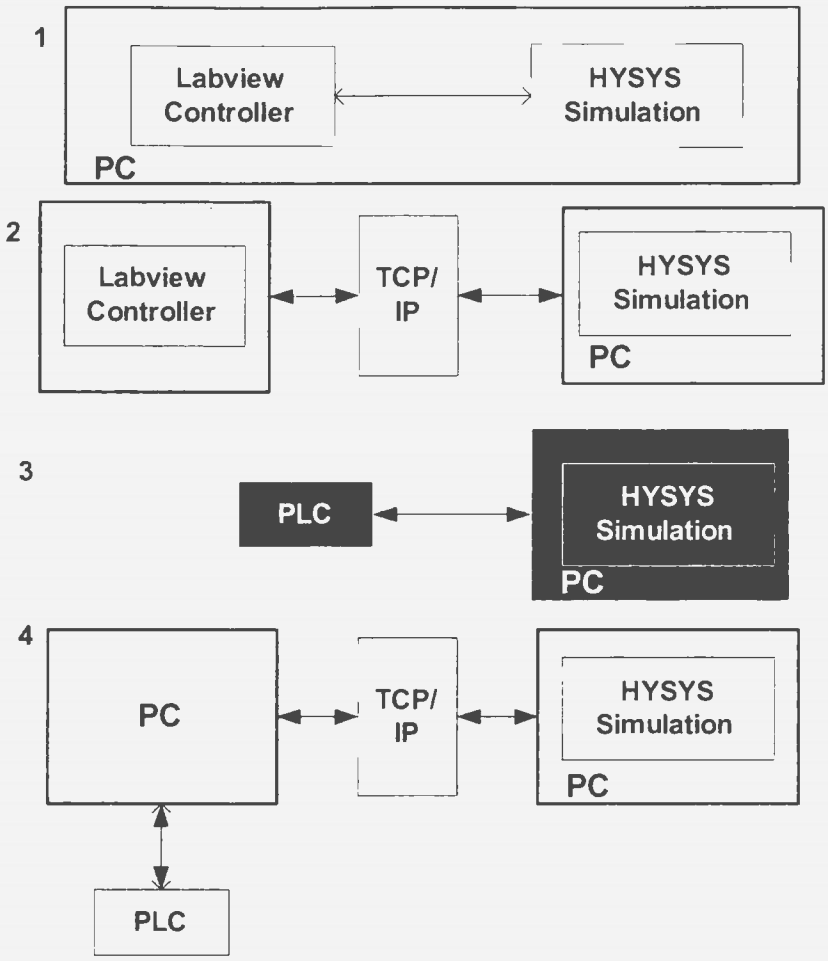

Figure 4-6: HYSYS Simulation with a PLC using a single PC.

4.2 Analysis of Results

The results and analysis of the four tests will be discussed in this section. First, a Matlab/Simulink simulation of a similar system will be analyzed, and then the results of the four tests will be discussed and compared to the Matlab/Simulink model and with each other.

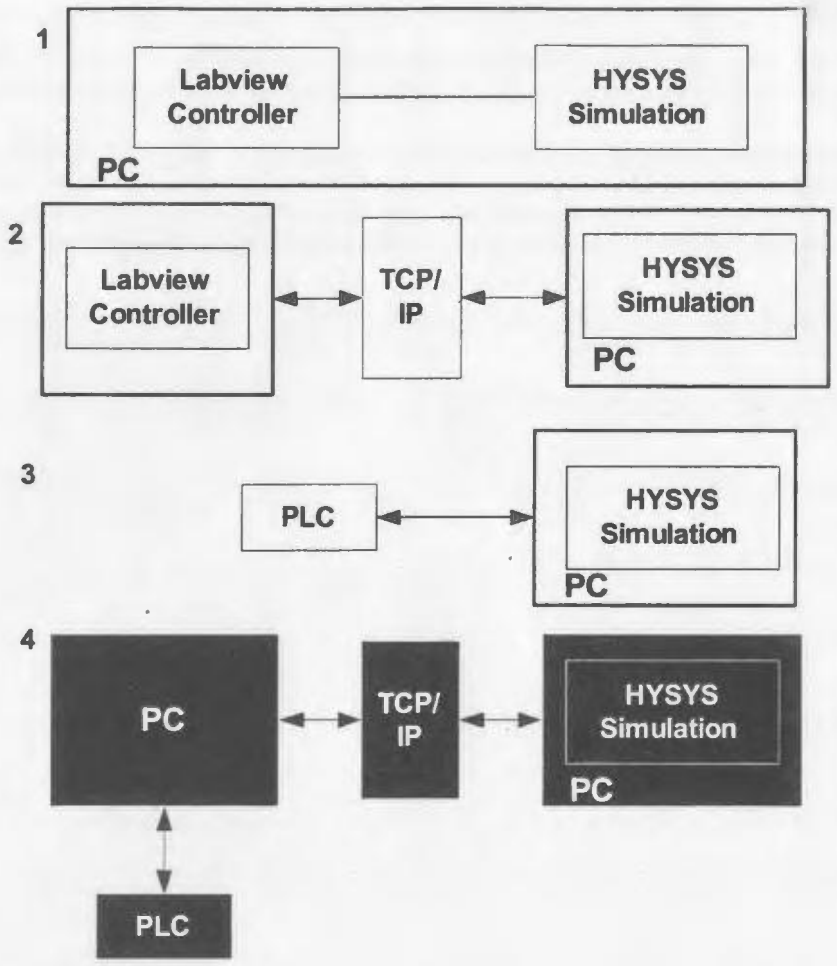

Figure 4-7: PLC to HYSYS Simulation over a TCP/IP Network.

4.2.1 Matlab and Simulink Simulations

In order to predict the outcome of the experiment, a Simulink model that is similar to the HYSYS process and a Simulink PID controller with the same parameters as the PID controllers in the experiment were included. The first component of this simulation that was needed is the process or plant transfer function that has a similar response as the HYSYS simulation that is shown in figure. Knowing that this system

is a non-interacting second-order lag process, the transfer function will be of the form[7]:

$$F(s) = \frac{G}{1 + A_1 s + A_2 s^2} \tag{4.1}$$

Where G is the gain of the system, A_1 is the sum of the *time constants[7]* $\tau_1 + \tau_2$. The time constants τ_1 and τ_2 are independent time constants for each of the independent systems in the second order system. A_2 is the multiple of the two time constants ($\tau_1 \tau_2$). For this system, τ_1 and τ_2 are equal in an effort to simplify the system.

The *time constant* is the response time that it takes in order to reach 63.2% of the systems maximum with a given *step input.* In this case, the step input will be when the input valve of the HYSYS simulation is open at 100% and the time that it takes to fill a single tank with this input is 40 seconds. However, the approximate time recorded that it takes for the tank to reach 63.2% of its maximum (1.264 m) is 25.28 s. The time constant found was recorded within the Labview VI. This program will record the time when the tank reaches 63.2% of its maximum. This time recorded is the time constant τ for one of the first order systems within the overall second order system. This time constant is an approximate value and does not represent the exact time constant for one of the first order system. It does give the experiment a time constant that can be used to illustrate the behavior of a similar system when discrete time delay is added.

Assuming the gain $G = 1$, the transfer function of the second-order lag process is:

$$F(s) = \frac{1}{1 + 50.56s + 639s^2} \tag{4.2}$$

The first simulation is described in figure 4-8. This simulation will show the step response of the transfer function 4.2 with a PID controller with the parameters that are used in the experiment.

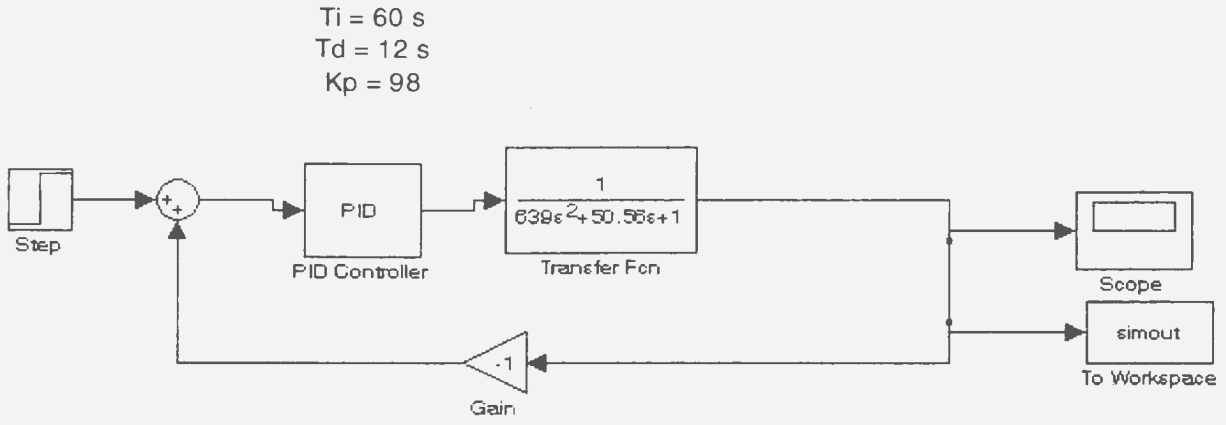

Figure 4-8: Simulink Diagram of Step Responce of the Model Simulation

The simulation shown in figure 4-8 is the initial Simulink simulation of the model system. This simulation will consist of the model process and the PID controller, but it will not consist of any delay. The result of this simulation over a six minute time period is shown in figure 4-9.

The step response shown in figure 4-9 is a desired response for the control system for the model process and the model PID controller.

When a discrete time delay is added to the system, the response should change.

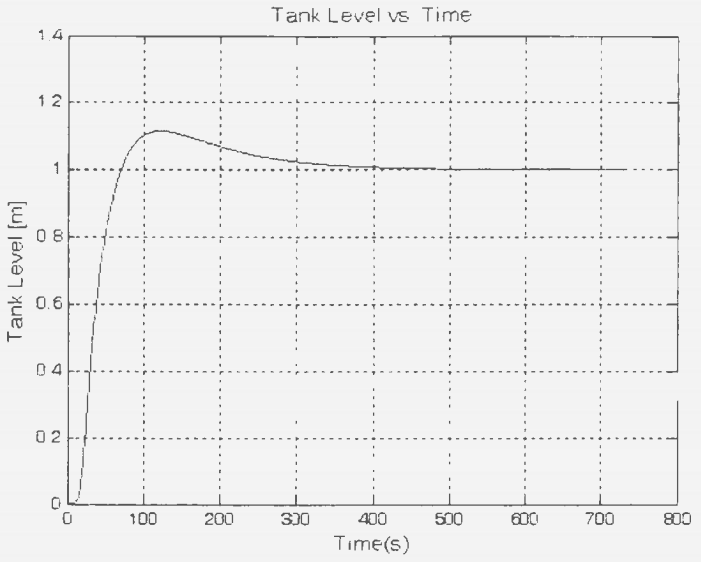

Figure 4-9: Simulink Simulation of the Model System with no Delay.

In the next simulation we add a discrete time delay because the delays within the experiment are of a discrete nature and not a continuous nature. A continuous delay would only cause a time shift in the overall response of the system. Figure 4-10 is the Simulink simulation of the model system with a discrete unit delay $\frac{1}{z}$.

The result of the simulation illustrated in figure 4-10 is shown in figure 4-11. This shows that with an added discrete time delay the system response changes response and decreases the performance of the control system while using the same PID control parameters.

The next simulation will consist of another discrete unit time delay added to the system, but this time the delay will be added into the beginning of the system before the PID controller as shown in figure 4-12.

With the addition of another time delay, the system response changes again and

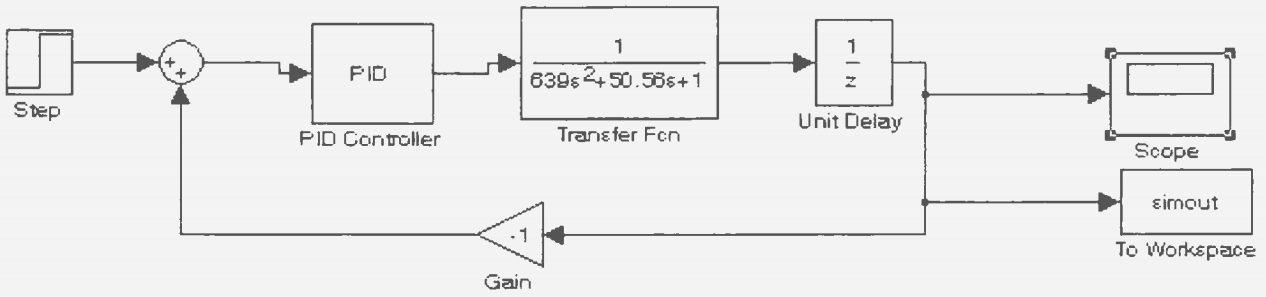

Figure 4-10: Simulink Diagram of Step Response of the Model Simulation with a Discrete Unit Time Delay.

the performance of the system decreases again as shown in figure 4-13.

The last model Simulink simulation will consist of three discrete unit time delays. From the previous simulations, the expectation of the simulation is that the system response will change and the performance of the system will decrease once again. Figure 4-14 shows the Simulink simulation with three discrete unit time delays.

The result of the simulation shown in figure 4-15 shows that when a certain amount of delay is added, the system will become unstable.

The Simulink model simulations demonstrate how a discrete time delay can affect a system model that is similar to the system in this experiment. The result of these simulations show that with added discrete time delay, the performance of the system will decrease and the same can be expected in the experiment.

4.2.2 Labview Controller to HYSYS Simulation on one PC

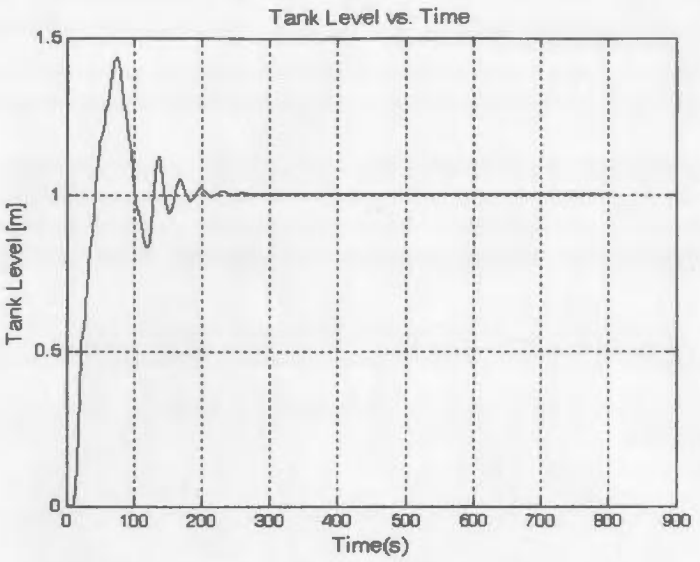

Figure 4-11: Simulink Simulation of the Model System with a Single Unit Delay.

The first test as shown in figure 4-1 and figure 4-4 is a simulation with the HYSYS process simulation and a Labview software controller. According to the Simulink simulation illustrated in figure 4-8 gives the assumption that the result response should be a stable one similar to the response in figure 4-9. The result of test one is shown in figure 4-16 and the response is a stable one, however, the tank does not fill to 0.9 m. Once it reaches a point close to 0.8m it tends to level off and never reaching the set point of 0.9 m.

The curve does support a good controller for the process simulation. The value never quite reaches the set point, but it never goes above it either. The response difference between the Simulink model and this test could be that the process model does not model the actual HYSYS process precisely. The Simulink model is just a benchmark to get an idea of what the result should be, but it doesn't have to be

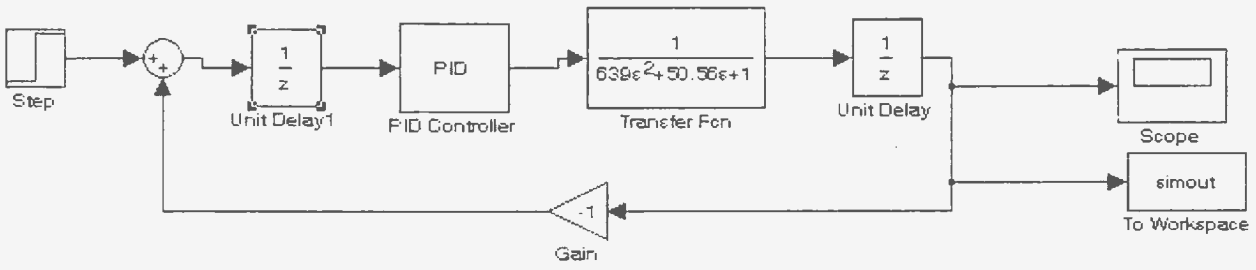

Figure 4-12: Simulink Diagram of Step Response of the Model Simulation with two Discrete Unit Time Delays.

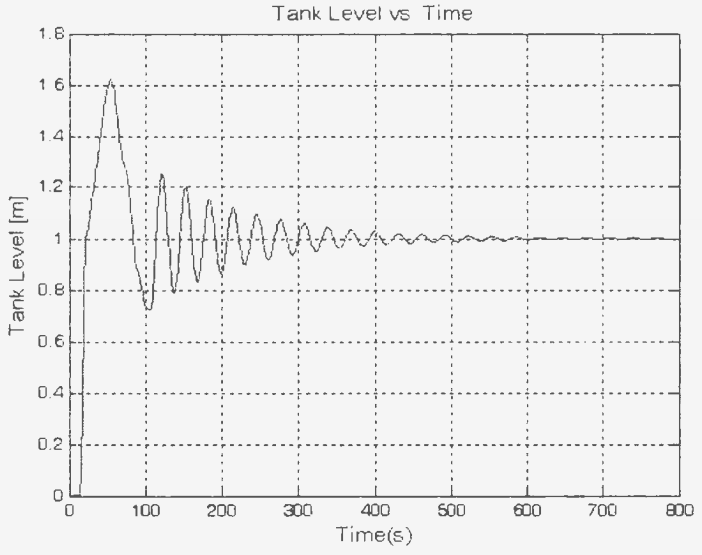

Figure 4-13: Simulink Simulation of the Model System with two Discrete Unit Delays.

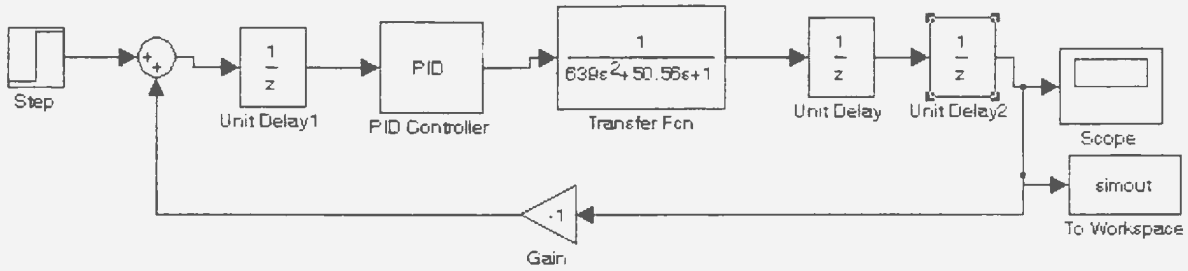

Figure 4-14: Simulink Diagram of Step Response of the Model Simulation with three Discrete Unit Time Delays.

perfectly accurate. The test response, however, does support that its system response is similar to the Simulink simulations systems response, but there are differences since the Simulink simulations are ideal and the actual simulations are not. Another difference could due to the CPU processing efficiency. When the CPU is running HYSYS or Labview on a Windows platform, the programs can slow down affecting the performance of the simulation due to other processes that the Windows operating systems must perform.

The CPU and Labview can cause other issues with the test. For instance, sometimes the CPU may need to perform other tasks that may take away from the Labview and HYSYS programs completely. This will cause one or both of the simulations to stop while the operating system deals with another process for a short time. This short time can be critical to the simulation. Figure 4-17 shows what can happen when the CPU and operating system reallocate their resources elsewhere for a short

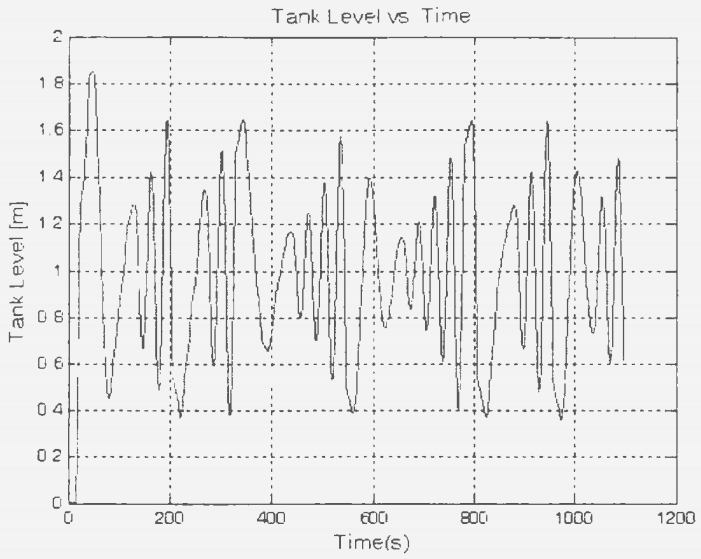

Figure 4-15: Simulink Simulation of the Model System with three Discrete Unit Delays.

period of time.

When this tests reaches 250 seconds, the tank empties with no effort by the Labview controller to recover to the set point level. Several seconds later, the controller opens the valve to 100% in order to recover the set point, however, the set point is not recovered in the six minute simulation. The tank is usually due to a reallocation of computer resources from Labview to another process running on the Windows operating system.

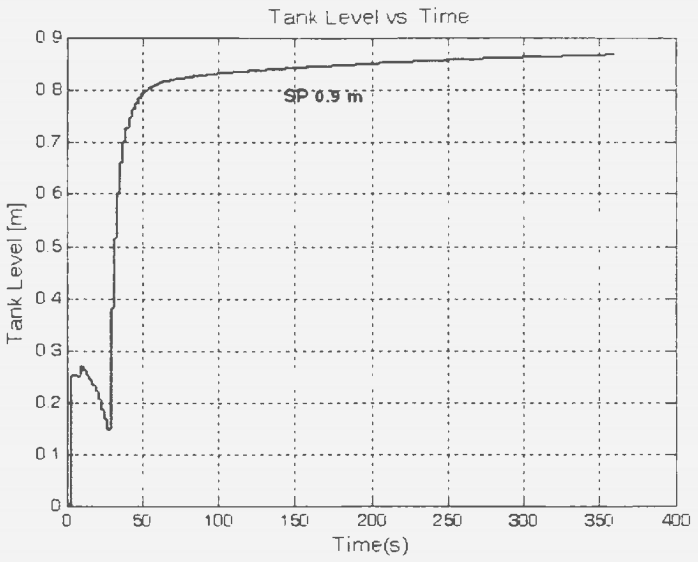

Figure 4-16: Result of Labview Controller and HYSYS Process Simulation on 1 PC.

4.2.3 Labview Controller to HYSYS Simulation on Two PC's over a WAN

The second test, as shown in figure 4-1 and figure 4-5, will consist of a Labview controller and a HYSYS process simulation, but this time it will be performed over a distributed system using a poor performing WAN. Several Simulink simulations, as shown in figures 4-10, 4-12, and 4-14 demonstrate what can happen to similar a system when a discrete time delay is added. The results of these simulations given in figures 4-11, 4-13, and 4-15 shows that with an added time latency, the performance of the control system decreases. The result of this test gives the same results illustrated in figure 4-18.

The result of second test shows that with the introduction of a random delay

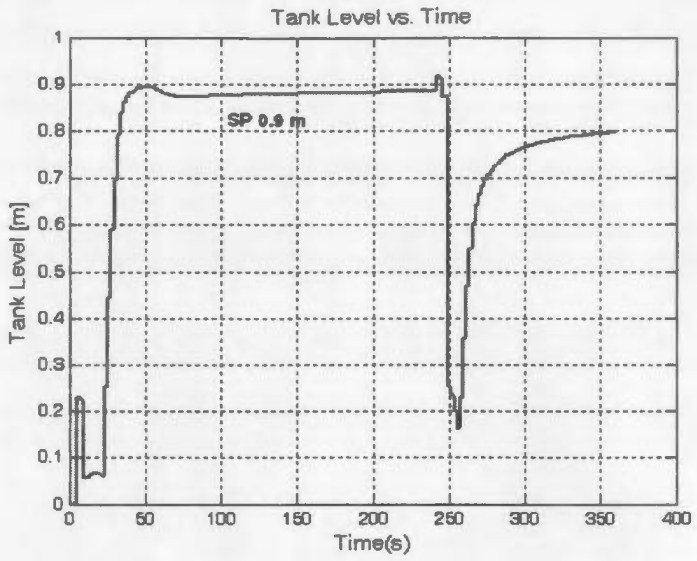

Figure 4-17: Labview and HYSYS Simulation on one PC with a OS Resource Reallocation

caused by the poor performing WAN does decrease the performance of the control system. Figure 4-18 compared to the results in test one (figure 4-16) shows that test one with little to no delay will give you better control system performance.

4.2.4 PLC and HYSYS simulation using one PC

The third test, as shown in figure 4-1 and figure 4-6, will be similar to first test because it is simulated on one system and there is little time delay, but this test uses HILS. The hardware controller that is used is a PLC and is connected to the PC via Modbus where the HYSYS simulation is located. The PLC is connected to the PC via a Modbus LAN connection and the PC by an OPC server accesses the data within the PLC.

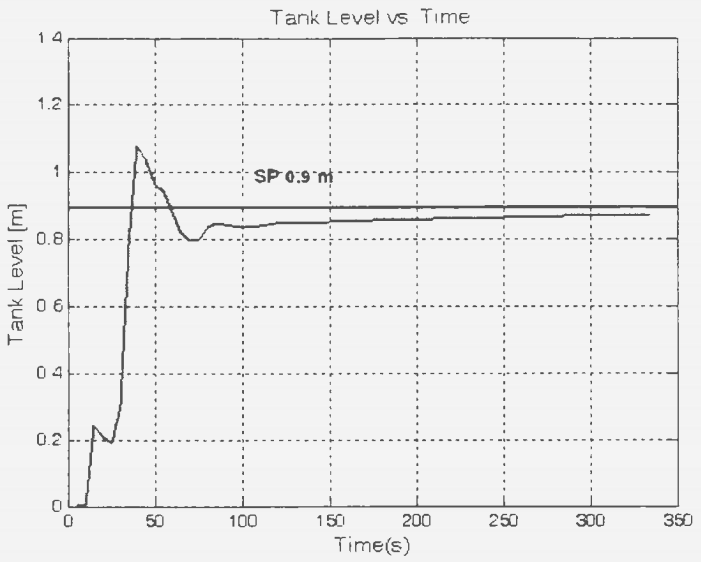

Figure 4-18: Labview Controller and HYSYS Process simulation over a WAN

The Simulink simulation results illustrated in figure 4-9 gives the step response of the model system with a PID controller with the same parameters as in the PLC. Again, this will be a benchmark for the results in this test given in figure 4-19.

The results of the third test are very similar to the result of the step response Simulink simulation shown in figure 4-9. This supports that the result of the third test is an expected result.

4.2.5 PLC and HYSYS Simulation on a Distributed System over a WAN

The forth and final test is the most important test out of the four. This test is a HILS over a distributed system using a WAN with poor performance. One PC within

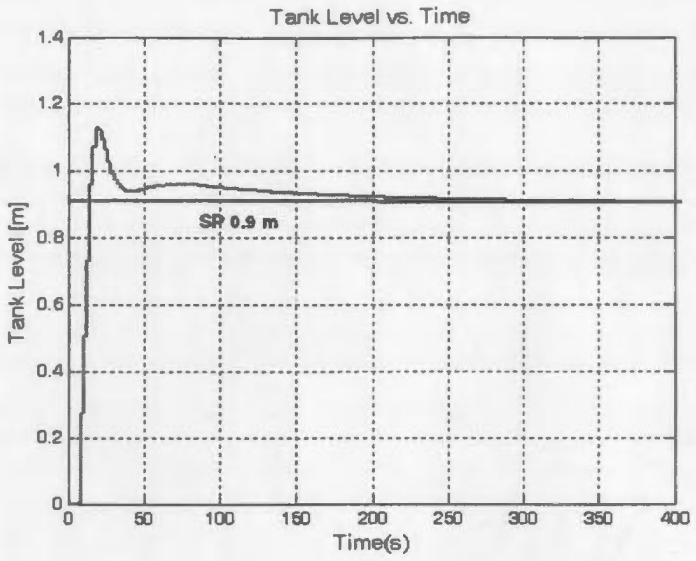

Figure 4-19: PLC to HYSYS Process Simulation on 1 PC and a Modbus LAN.

the distributed system will consist of the HYSYS process simulation and the second PC will consist of the Modbus LAN connected to the PLC as shown in figure 4-1 and figure 4-7.

The result of this test, shown in figure 4-20, should be first compared with the results of the Simulink simulations shown if figures 4-10, 4-12, and 4-14. As stated before, the results of these Simulink simulations illustrated in figures 4-11, 4-13, and 4-15 show that with added discrete time delay, the performance of the control system decreases.

The fourth test doesn't match any of the responses exactly, however the response of the HILS using a distributed system with random time latency did have a decrease in performance of the control system. This can be compared to the result in the third test illustrated in figure 4-19 where this same simulation using the same controller and

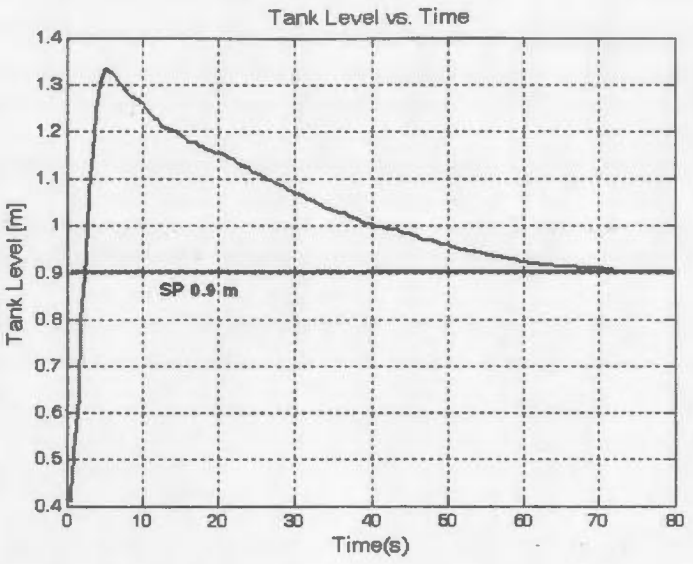

Figure 4-20: HYSYS and PLC Simulation over a Distributed System using a WAN.

process simulation, but the random time latency did not exist. This proves that with significant random time latency added into the simulation will effect the performance of the entire HILS environment.

4.2.6 Discussion of Results

According to the Simulink simulations, the more discrete delay that there is in a system, the greater loss in performance in a control system. The systems become underdamped due to the added delay and this is true in this experiment. Tests one and two where the Labview software controller was used illustrated this fact. The first test where the Labview controller and the HYSYS simulation were on one PC, the performance of the controller was good. The second test where the Labview controller and the HYSYS simulation were on different PC's communicating over a WAN with

1 to 4 seconds of random delay between each system the performance of the control system decreased.

The effect of the delay produced when performing a simulation over a distributed system with significant delay is also shown in the two tests that involved the PLC with a Modbus connection and the HYSYS simulation. The test where the PLC is directly connected to the PC with the HYSYS simulation via Modbus gave very good performance results. The controller acted very well to the HYSYS simulation showing that HILS can work well for this type of simulation. The next test had the HYSYS simulation on one PC and the PLC and Modbus connection on another with each PC communicating via WAN with 1 to 4 seconds delay. The result of this test also shows that with added random delay, the performance of the control system decreased as the system became underdamped. This proves that when there is an uncertain delay in the system, there is a negative effect on the performance of the overall system.

Not only did the results of this experiment show the effects of time delay over a distributed system, but it also illustrates the difference between using a real hardware controller and a software controller. The actual controller that will be used in the field will be the PLC and not a third party software controller such as the Labview PID toolkit. Both software and hardware controllers did perform control algorithms properly and the results of each were similar, however, they were not the same. This shows that if the Labview PID software controller was used to validate the HYSYS process, the resultant controller could be invalid. Also, as shown in figure 4-17,

software controllers can timeout and cause serious damage to a system as well as skewing the simulation results.

4.3 Concluding Remarks

This chapter first gives an outline of the experiment and the four tests that was performed to analyze time latency within a HILS environment using a distributed system with a poor performing WAN. It also analyzes the difference between using a real hardware controller and a third party software controller. The next chapter will give the conclusion, suggestions, and future work that can be performed within this area of research.

Chapter 5

Conclusions and Future Work

The research that has been presented in this thesis demonstrates that HILS is achievable and very useful in the validation of controllers with software process simulations. Also, it shows how a controller performs in a HILS environment when the process and the controller are communicating over a WAN with significant delay. This chapter discusses the accomplishments of the research conducted and show that the industry and project goals that were discussed in chapter one were fulfilled. As well, this chapter will discuss suggestions and some future work that could be done for this research.

5.1 Goals and Results

The introduction chapter in this thesis discussed both industry and research goals of this project. For the industry goals, three possible industry scenarios that could

happen with the White Rose FPSO project are:

1. Before the PLC's that will be commissioned to the FPSO, they must be verified in a simulation environment. The best way to verify the PLC's before sending controllers to Marystown to be commissioned is using a hardware-in-the-loop simulation method using the PLC's as the real hardware and the HYSYS process model of the topside of the FPSO. A platform that is relatively easy to use that could perform this task would be very useful in commissioning of the controllers because is would significantly decrease the chance of failing in the field. They will work the first time instead of making too many changes in the field that, at times, can be very costly to the project.

2. A controller may have to be verified in Marystown, but all of the HYSYS simulation tools are in St. John's. In this case, hardware-in-the-loop simulation may be performed over a TCP/IP network to save time and money. The HILS platform must have the ability to connect to the simulation in St. John's and perform HILS as if the controller was in St. John's. Such a simulation platform could be very useful and valuable to the FPSO topside project due remoteness of the construction site.

3. When the FPSO is in operation at sea, it may be very dangerous and costly to tune a controller in the field since changes in the field of this nature could affect the process and have unwanted results. A way the controller could be tuned is by HILS, but again the HYSYS process simulation and tools are in St. John's.

A simulation platform, similar to the second scenario, could be used to simulate the controller over a TCP/IP network with the HYSYS process simulation in St. John's without any unwanted effects with tuning a controller with the actual process.

The first scenario could be solved by the creation of a HILS environment for HYSYS and the controller of choice. In this project, a platform was created to perform HILS with a HYSYS simulation using a Modicon PLC over a Modbus network and it was found that this from of simulation is very effective and better then using the PID models within HYSYS. However, for the White Rose project Siemens PLC's with a Profibus network is being used. The only change that would have to be made is the OPC server. The present OPC server is for serial Modbus, but there are OPC servers available for all control networks. All that would have to change is that the Modbus serial OPC server would have to be replaced with a Profibus OPC server. This illisturates that this platform is universal to any technology that supports OPC.

The second and third scenarios can be solved by HILS with a distributed system over a WAN. This project produced a HILS platform that could be used over a TCP/IP network. This means that the HYSYS process model could be anywhere in the world and with the proper software developed in this project, a simulation can be performed over a TCP/IP network with the HYSYS simulation on one side and the PLC on the other side. Also, a user console is also available in Labview so the user can observe the results of the simulation. This solves the second and third scenarios

because the simulations can be performed if the HYSYS simulation is in St. John's and if the controller is in Marystown or at sea. All that the user needs to know is the IP address of the HYSYS simulation in St. John's and have the necessary software that this project created.

Using an open WAN has never been used in industry or in a simulation environment within the process control industry. The research presented in this thesis not only proves that it can be done, but shows that with further research and development that it may one day change the way distributed control is implemented in real world situations.

This thesis also set five research goals. The goals are:

1. How to interface the correct data from the process simulator to the PLC and from the PLC to the process simulator?

2. How do you do this over a communication network such as TCP/IP?

3. How do you get a PLC to act in a way that it does not know that it is connect a simulator instead of an actual process?

4. What is the difference between a software controller to software process and a hardware controller to software process?

5. What effects will time latency over a WAN have on the controller to process simulation?

The first three goals have been met and answered in chapter 3, Design and Implementation of HIL Distributed Simulation. The interface for the correct data to and from the HYSYS simulator and the PLC are done by the use of OPC and software automation. Data within HYSYS is manipulated by custom automation software and this automation software is connected to the PLC via an OPC server. The OPC server also supports automation and enables the automation software to send the proper data from HYSYS to the memory locations within the PLCs RAM that corresponds to the process variable and the output variable of a PID control algorithm within the PLC.

Communication software was created in order to send data over a TCP/IP network. This was done using the C# .NET framework where the automation objects within HYSYS and the OPC server were imported into the C# .NET environment and by using Marshalling and Remoting techniques. The data within HYSYS and the controller could be sent over a TCP/IP network.

The fourth and fifth research goals for this project are met and discussed in chapter 4, Design of Experiment and Analysis of Results. The fourth goal is met when a comparison of a Labview software controller with a actual PLC was discussed. The results show that a third party software controller does not behave the exact same way as a real hardware controller. The fifth goal was met because the analysis was conducted between simulation over a WAN with significant random delay and a simulation with little delay on a single computer. Tests were performed on the

two cases with both hardware and software controllers and it was found that with added random time delay, the performance of the controller decreases. A Simulink simulation with a similar model process, PID controller with the same parameters, and discrete time delay also verified this result.

The process control industry has never used an open WAN to perform distributed control or used a WAN in a simulation environment. This research shows the effects of a control system using the TCP/IP network protocol with a random delay added to simulate a poor performing network. Random delay of this magnitude has never been considered in control systems in the past, but is being considered in the future of process control. This research goal gives a result that shows future engineers what are the possibilities of using such a network in a control design.

On top of the industry and research goals for this thesis, the PPSC project also had several objectives that had to be met. These objectives are:

1. Design a Universal Simulation Interface (USI) module to study connectivity over a heterogeneous platform of DCS and process control simulators over a LAN (Local Area Network).

2. Extend the USI connectivity over a WAN (Wide Area Network) to study dynamics of control loops that are closed over Ethernet or similar networking protocols.

3. Establish a closed-loop performance benchmark of the USI module by artificially degrading networking throughout between agents distributed within a cluster.

The first goal set by the PPSC has been met by the research done in this thesis. A universal simulation interface module was created to connect a distributed control system (or PLC) and a process control simulator (HYSYS) over a LAN. Again, this platform was created using OPC (an industry standard) with automation software that acted as a bridge between the OPC server and the HYSYS simulation. The LAN that was used in this research was Modbus, which is the communication protocol that was used as the communication link between the PLC and the PC where the HYSYS simulator is located.

The second goal was completed when the HILS over a distributed system using a WAN was completed. Creating an interface using C# .NET technology to send the appropriate data over a TCP/IP network meets the requirements for this objective.

The third goal that the PPSC set was also met in this thesis with the experiments in chapter 4. The HILS over a distributed system using the TCP/IP WAN was created and a random delay generator that generated a delay between 1 and 4 seconds was added to the simulation environment. This generator is the agent that artificially degraded the network. A closed loop simulation was performed between a HYSYS process model on one PC in the distributed network and the PC that is connected to the PLC via Modbus is connected on the other side of the distributed network. The two computers connected over the WAN represents a cluster and the results of the closed loop simulations are discussed and compared with simulations that are simulated in a local environment.

5.2 Suggestions and Future Work

The research that was conducted in this thesis was a success. However, several suggestions for future work could be added. Software interfacing, synchronization, PID tuning algorithms when dealing with network delay, and timed automata are several aspects have been considered for further research for this project.

1. A new, user-friendly software GUI could be created for this project. Presently, there is no external support for selecting objects within the HYSYS simulation and OPC. A GUI would be useful for selecting particular objects within either HYSYS or OPC without having to set the particular objects within the custom software.

2. Synchronization is one area that we did not consider in this research because we were using continuous PID controller with a mechanical process. The PID controller will actually start when the HYSYS simulation starts. However, synchronizing the controller and process using a real-time clock and a particular synchronization algorithm would be very interesting to investigate and can optimize the simulation. Another area that could be researched is the use of predictive algorithms in order to predict and compensate for the WAN delay. This could be done by reading the average data rate of the network and with a model of the system, predict the appropriate PID control values for that system for that particular time.

3. This research shows that it is possible to simulate a real hardware controller, such as a PLC, with a HYSYS process simulator. It also shows that the performance of the control system decreases with the increased time delay. This means that in order to tune this controller properly, the delay must be accounted for. A possible continuation of work could be the creation of a PID tuning algorithm that handles the effect of random delay within a system. The algorithm could take in account the delay and then change the PID parameters according to the delay, giving the real controller the proper PID parameters that will react properly to the real process.

4. Using formal methods such as timed automata[35] could be an area of continued research with this project. Timed automata models can validate the entire system and the effects of random time delay could be studied using this form of automata. It can also be used to synthesize a controller mathematically using the specification of the process and the entire system.

The last three suggestions are topics could possiblity be used for future work in the area of distributed control systems using an open WAN. The main issue that must be addressed is random time latency within the system. Synchronization, predictive algorithms, and timed automata are researchs areas that could address the issue of time latency. The research done in this thesis proves that hardware in the loop simulation over an open WAN could be done and an evalulation of time latency. The next logical step would be to take the results of this research and find ways to deal

with the random time latency in an open WAN.

5.3 Concluding Remarks

This thesis first discusses the background of HILS and distributed systems. It then illustrates the design and implementation of a HILS environment using a distributed system over a WAN. The system that was implemented was a HYSYS process simulation communicating with either a PLC or a Labview controller over a TCP/IP network. Next, simulations were performed where the difference between using control simulation with a HYSYS process with a PLC or a Labview controller either locally on one PC or over a TCP/IP network. Two issues were studied. The first issue was the effects of time delay on the entire system and how the performance of the system decreases with added time delay. The second issue that was discussed is the difference between using a real hardware controller or a third party software controller with the HYSYS process simulation. The results clearly show that the software controller does not perform exactly the same as the real hardware controller. This chapter then discusses the conclusions for the thesis and future work that could be performed in this area of research.

Appendix A

C# Software

A.1 Client

```
using System;
    using System.Runtime.Remoting;
    using System.Threading;
    using System.Runtime.Remoting.Channels;
    using System.Runtime.Remoting.Channels.Http;
    namespace RemoteHyInterface
    {
        /// <summary>
        /// Summary description for Class1.
        /// </summary>
```

```
public class HysysPDClient
{
    public HysysPDClient()
    {
        System.Console.WriteLine("This is the Client constructor.");
    }

    public static void Main()
    {
        // create an Http channel and register it
        // uses port 0 to indicate won't be listening
        HttpChannel chan = new HttpChannel(0);
        ChannelServices.RegisterChannel(chan);
        // get my object from across the http channel
        //IP 192.168.0.209 is my IP address.
        MarshalByRefObject obj =(MarshalByRefObject) RemotingServices.Connect(t
"http://192.168.0.209:65100/theEndPoint");

        try
        {
```

```
                // cast the object to our interface (typecasting!!!)
                RemoteHyInterface.IHysys2 ThisHysys = obj as RemoteHyIn-
terface.IHysys2;
                //now use the interface class to call the needed methods.
                ContInt ControllerObj = new ContInt(); //constructor for Hysys
controller container class
                //string dumb = null; //testing
                //Labview interface
                while(true)
                {
                    ThisHysys.runSimulation();//revalulate the variables in Hysys.
                    Thread.Sleep(50);//50ms delay
                    ControllerObj.setLevel(ThisHysys.getPVLevel()); //gets the
tank level from the Hysys simulation.
                    Thread.Sleep(100);
                    ThisHysys.setValveOpening(ControllerObj.getAct());
                    Thread.Sleep(100);
                    //may have a deadlock issue here.
                }

            }
```

```
            catch( System.Exception ex )
            {
                Console.WriteLine("//////////////////////////////");
                Console.WriteLine("Problem with Remote interface!!!");
                Console.WriteLine("Exception caught: ");
                Console.WriteLine(ex.Message);
                string tempStr = null;
                Console.WriteLine("//////////////////////////////");
                while(tempStr != "OK")
                {
                    Console.WriteLine("Enter OK to end:");
                    tempStr = Console.ReadLine();
                }
                Console.WriteLine("//////////////////////////////");
                Console.WriteLine("Exception Finished!!!!!");
            }
        }
}
public class ContInt //container class
{
    private double setPT;
```

```
        private double Level;
        private double actuator;
        public ContInt()
        {
            setPT = 0.0;
            Level = 0.0;
            actuator = 0.0;
            //constructor
        }
        public void setSetpoint(double sp){setPT = sp;}
        public void setLevel(double lev){Level = lev;}
        public void setAct(double actPos){actuator = actPos;}
        public double getAct(){return actuator;}
        public double getLevel(){return Level;}
    } //container class
}
```

A.2 Server

```
using System;
    using System.Runtime.InteropServices;
    using System.Threading;
```

```
using System.Runtime.Remoting;
using System.Runtime.Remoting.Channels;
using System.Runtime.Remoting.Channels.Http;
using System.Windows.Forms;
using HyInt;
namespace RemoteHyInterface
{
    /// <summary>
    /// Summary description for Class1.
    /// </summary>
public class ServerHysys : MarshalByRefObject, IHysys2
    {
    private VBHysysIntClass HyObj;
    private double valveOpening_OP;
    private double PVLevel;
    private string simPath;
    private OpenFileDialog FileBox;
    private string dumb;
    private Random R;

    public ServerHysys()
```

```
        {
        System.Console.WriteLine("The Hysys server constructor has been acti-
vated!!!");
        dumb = null;

        try
            {
                Console.WriteLine("Hello!!!!!");
                HyObj = new VBHysysIntClass();
                System.Console.WriteLine("The VB object has been created!");
                PVLevel = 0.0;
                valveOpening_OP = 0.0;
                R = new Random(); //random number generator object.
                FileBox = new OpenFileDialog(); //creates file box
                FileBox.ShowDialog(); //displays the file box.
                simPath = FileBox.FileName; //gets the selected file and path
as a string.
                HyObj.setSimPath(ref simPath); //send the Hysys file path to
the Hysys tool.
                HyObj.Form_Load(); //initialize simulation.
            }
```

```
        catch(System.Exception ex)
        {
            Console.WriteLine("Exception thrown in Server:");
            Console.WriteLine(ex.Message);
            Console.WriteLine(ex.Source);
            Console.WriteLine("Hit Enter to continue:");
            dumb = Console.ReadLine();
        }
    }

    public void setValveOpening(double ValOP)
    {
        int del = R.Next(1000, 4000);
        Console.WriteLine("The delay is.....");
        Console.WriteLine(del);
        Thread.Sleep(del); //creates a delay between 1 to 4 seconds.
        HyObj.changeActuator(ref ValOP);
        valveOpening_OP = ValOP;
        HyObj.Form_Load();
    }
    public double getValveOpening(){return valveOpening_OP;}
```

```
public double getPVLevel()
{
    int del = R.Next(1000, 4000);
    Console.WriteLine("The delay is.....");
    Console.WriteLine(del);
    Thread.Sleep(del); //creates a delay between 1 to 4 seconds.
    HyObj.Form_Load();
    PVLevel = HyObj.PV_Value();
    return PVLevel;
}
public void runSimulation()
{
    HyObj.Form_Load();
}
public static void Main()
{
    // create a channel and register it
    HttpChannel chan = new HttpChannel(65100);
    ChannelServices.RegisterChannel(chan);
    Type HyType = Type.GetType("RemoteHyInterface.ServerHysys");
    // register our well-known type and tell the server
```

```
            // to connect the type to the endpoint "theEndPoint"
            RemotingConfiguration.RegisterWellKnownServiceType( HyType,
                "theEndPoint",
                WellKnownObjectMode.Singleton ):
            while(true)
            {
                Thread.Sleep(50);// 50ms delay.

            }//busy while loop. The sever is running constantly!!!
        }
    }
}
```

A.3 OPC Wrapper

```
using System;
    using System.Runtime.InteropServices;
    using System.Threading;
    using System.Runtime.Remoting;
    using Project1; //OPC project
    using RemoteHyInterface;
    namespace OPCWrapper
```

```
{
    /// <summary>
    /// Summary description for Class1.
    /// </summary>
    public class OPCtoTCP
    {
        private double PV_Value;
        private double OP_Value;
        public OPCtoTCP()//contstructor
        {
            PV_Value = 0.0;
            OP_Value = 0.0;
        }
        public double getPVValue(){return PV_Value;}
        public double getOPValue(){return OP_Value;}

        public static int Main()
        {
            try
            {
                Project1.OPCInterfaceVBClass OPC_PLC = new OPCInter-
```

```
faceVBClass();
                    Random R = new Random();

                    Console.WriteLine("Enter IP Address:");
                    string IP_Addr = Console.ReadLine();

                    RemoteHyInterface.HysysPDClient TCPClient = new HysysPD-
Client(IP_Addr);
                    OPC_PLC.Connect_Server();
                    Console.WriteLine("OPC server connected!!!");
                    double inOP = 0.0;
                    double outPV = 0.0;
                int passNumber = 0;

                    while(true)
                    {
                        //This gets the OP Value from HYSYS over the network
and sends it
                        //to the OPC server that then sends it to the actual PLC
over Modbus.
                        //OPC_PLC.Write_Sync_OP(ref TCPClient.getPV());
```

```
            Console.Write("Pass number:");
            passNumber++;
            Console.WriteLine(passNumber);
            OPC_PLC.Read_Sync_OP(ref inOP);

            Thread.Sleep(10); //short delay
            TCPClient.setOP(inOP);
            Thread.Sleep(10); //50ms delay
            outPV = TCPClient.getPV();
            Thread.Sleep(10); //short delay
            OPC_PLC.Write_Sync_PV(ref outPV);
            Thread.Sleep(10); //short delay
            //testing.
            //OPC_PLC.Value_Written_PV(ref tempPV);
            //delay for testing purposes.
        }//while
    }//try
    catch(Exception ex)
    {
        Console.WriteLine("Exception Thrown!!!");
        Console.WriteLine("————————————");
```

```
                Console.WriteLine(ex.Message);
                Console.WriteLine("— ———————      ");
                Console.WriteLine("Hit Enter to Continue:");
                string dumb = Console.ReadLine();
            }//catch
            return 0;//end program.
        }//main

    }
}
```

Appendix B

Visual Basic Software

B.1 HYSYS Test Code

```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

    ''Example of a HYSYS automation in Visual Basic........

    ''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

    Option Explicit

    'Public Objects

    Public HyAppl As Object

    Public HyCase As SimulationCase

    Public HyFlowsheet As Flowsheet

    Public PropControl_PV As Object

    Public PropControl_SP As Object
```

```
Public PropControl_OP As Object "need to declare this object
'variables
Dim PropControl As Controller
Dim ValveSetting As Valve
Dim Actuator_SP As Double
Dim ActuatorPos As Double
Dim CompProp As Variant
Dim temp As Variant
Dim actPres As Double
Private Sub Form_Load()
'Start
'use this for now. We will be using the GetObject for this later.
\
'Connect to HYSYS objects
Set HyCase = GetObject("c:\Masters\Hysys_Bridge\dyntut3.hsc")
'Connect to applications object
Set HyAppl = HyCase.Application
Set HyCase = HyAppl.ActiveDocument
'Connect to flowsheet in active simulation case
Set HyFlowsheet = HyCase.Flowsheet
'declare controller object
```

```
Set PropControl = HyFlowsheet.Operations.Item("PropOxide FC")

Set ValveSetting = HyFlowsheet.Operations.Item("VLV-Prop Oxide")

'PropControl is now the controller object for PropOXide PID

'assign PV, SP values to object

Set PropControl_PV = PropControl.PV

MsgBox PropControl_PV

Set PropControl_SP = PropControl.SP

Set PropControl_OP = PropControl.OP

Actuator_SP = ValveSetting.ActuatorSPValue

ActuatorPos = ValveSetting.ActuatorPosition

MsgBox Actuator_SP

MsgBox ActuatorPos

MsgBox "Set new actuator value of 28.1"

ValveSetting.ActuatorSPValue = 28.1

ActuatorPos = ValveSetting.ActuatorPosition

Actuator_SP = ValveSetting.ActuatorSPValue

MsgBox Actuator_SP

MsgBox ActuatorPos

MsgBox "Wait for Change!!!"

ActuatorPos = ValveSetting.ActuatorPosition

Actuator_SP = ValveSetting.ActuatorSPValue
```

MsgBox Actuator_SP

MsgBox ActuatorPos

"does VB pass by reference or by value? By reference.

MsgBox "Done!!!"

End Sub

B.2 OPC

Option Explicit

'July 7, 2004

'Paul Handrigan

Dim WithEvents AnOPCServer As OPCServer

Dim ARealOPCServer As String

Dim MyGroups As OPCGroups

Dim DefaultGroupUpdateRate As Long

Dim OneGroup As OPCGroup

Dim ServerName As String

Dim AnOPCItemCollection As OPCItems

Dim AnOPCItem As OPCItem

Dim ClientHandles(2) As Long

Dim AnOPCItemIDs(2) As String

```
Dim AnOPCItemServerHandles() As Long

Dim AnOPCItemServerErrors() As Long

Dim Source As Integer

Dim ServerHandles(2) As Long

Dim Values() As Variant

Dim Errors() As Long

Dim Qualities() As Variant

Dim TimeStamps() As Variant

Dim testGroup As OPCGroup

Dim inputValues(2) As Variant

Dim ProgID As String

Public Sub Connect_Server() "Must be form load!!!

Set AnOPCServer = New OPCServer

ServerName = "AutomatedSolutions.ASMBSERIALOPC"

AnOPCServer.Connect (ServerName)

AnOPCServer.OPCGroups.DefaultGroupIsActive = True

Set OneGroup = AnOPCServer.OPCGroups.Add("Group1")

Set AnOPCItemCollection = OneGroup.OPCItems

Set testGroup = AnOPCServer.OPCGroups.GetOPCGroup("Group1") 'test

'This works ok!!!

OneGroup.IsActive = True
```

```
    OneGroup.IsSubscribed = True
    'This works ok!!!
    Set AnOPCItemCollection = OneGroup.OPCItems
    'Add two items
    ClientHandles(1) = 2
    AnOPCItemIDs(1) = "PLC.Group1.input" 'PV value.
    ClientHandles(2) = 3
    AnOPCItemIDs(2) = "PLC.Group1.output" 'OP value.
    'add in the items with a client handle and the proper Item ID.
    'initialize arrays to 0
    'Add the Items to the group within the server
    AnOPCItemCollection.AddItems 2, AnOPCItemIDs, ClientHandles, AnOPCItem-
ServerHandles, Errors
    'initialize server handle array
    ServerHandles(1) = 0
    ServerHandles(2) = 0
    ServerHandles(1) = AnOPCItemServerHandles(1)
    ServerHandles(2) = AnOPCItemServerHandles(2)
    'this works so far!!!
    ' inputValues(1) = 50
    ' inputValues(2) = 15
```

```
End Sub

Public Sub Read_Sync_OP(output As Double)

Dim TempArr(2) As Long

TempArr(1) = ServerHandles(2) "we want to read the OP value from the PLC.

Source = OPCDevice "THIS worked!!!!!!!

OneGroup.SyncRead Source, 1, TempArr, Values, Errors, Qualities, TimeStamps

'OneGroup.SyncRead Source, 1, ServerHandles, Values, Errors, Qualities, TimeStamps

output = Values(1) "first and only value in the Values array

End Sub

Public Sub Write_Sync_PV(op_in As Double)

inputValues(1) = op_in

OneGroup.SyncWrite 1, ServerHandles, inputValues, Errors

End Sub

Public Sub Value_Written_PV(outVal As Double)

Source = OPCDevice "THIS worked!!!!!!!

OneGroup.SyncRead Source, 1, ServerHandles, Values, Errors, Qualities, TimeStamps

outVal = Values(1)

End Sub
```

Bibliography

[1] C. F. D. of Electrical Engineering University of Pretoria Pretoria 0002 Republic of South Africa, "Control system analysis of hardware-in-the-loop simulation." *IEEE Transactions on Aerospace and Electronic Systems*, vol. 26, no. 4, pp. 666–669, July 1990.

[2] T. K. Peter Terwiesch and E. Scheiben, "Rail vehicle control system intergration testing using digital hardware-in-the-loop simulation," *IEEE Transactions on Control System Technology*, vol. 7, no. 3, pp. 352–362, May 1999.

[3] U. o. M. Wojciech Grega, Department of Automatics and P. Metallurgy 30-059 Krakow, Al.Mickiewicza 30, "Hardware-in-the-loop simulation and its application in control education," *29th ASEE/IEEE Frontiers in Education Conference*, November 1999.

[4] M. V. Harald Schludermann, Thomas Kirchmair, "Soft-commissioning: Hardware-in-the-loop-based verification of controller software," *Proceddings ofthe*

2000 Winter Simulation Conference, pp. 893–899, 2000, j.A Foines; R. R. Barton, K. Kang, and P. A. Fishwick, eds.

[5] D. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley, 1989.

[6] M. v. S. Andrew S. Tanenbaum, *Distributed Systems. Principles and Paradigms.* Prentice Hall, 2002.

[7] R. N. Bateson, *Control System Technology*, 5th ed. Prentice Hall, 1996.

[8] D. S. O'Young, "The development of an interface module for a distributed and multi-platform process control and simulation cluster," 2002, pPSC project outline and objectives.

[9] H. Buhler, "A driving simulator for investigating the static and dynamic characteristics of speed regulating systems for traction vehicles," *Bull Oerlikon*, no. 364/365, pp. 15–22, 1965.

[10] H. P. Wenk, "Simstar-ein modernes simulationswerkzeug. abb verkehrssysteme ag," ABB, Tech. Rep. 3, 1992.

[11] M. The MathWork Inc., Natick, "Matlab and simulink."

[12] L. Pollini and M. Innocenti, "A synthetic environment for dynamic systems control and distributed simulation," *IEEE Control Systems Magazine*, pp. 49–61, April 2000.

[13] M. Innocenti and I. Pollini, "A synthetic environment for simulation and visualization of dynamic systems," in *American Control Confrence*, San Diego, CA, June 1999.

[14] K. Chang and S. Lee, "Remote controller design of networked control system using genetic algorithm," in *ISIE*, School of Mechanical Engineering, Pusan National University, Korea. IEEE, 2001, pp. 1845–1850.

[15] *Profibus Specification - Normative Parts of Profibus-FMS, DP, PA According to the European Standard. Vol.*, 1998.

[16] J. M. Feng-Li Lian and D. Tilbury, "Network design consideration for distributed control systems," *IEEE Transactions on Control Systems Technology*, vol. 10, no. 2, pp. 297–307, March 2002.

[17] *Stability of Networked Control Systems: Explicit Analysis of Delay.* Chicago, Illinois: American Control Conference, June 2000.

[18] M. S. Minsoo Ryu, Seongsoo Hong, "Streamlining real-time controller design: From performance specifications to end-to-end timing constraints," *IEEE Control Systems*, pp. 91–99, 1997.

[19] F. Iwanitz and J. Lange, *OLE for Process Control.* Huthig Verlag Heidelberg, 2001.

[20] "Automated solutions inc." www.automatedsolutions.com.

[21] "Aspentech," www.aspentech.com.

[22] "Modbus-IDA," www.modbus.org.

[23] "Microsoft corporation," www.microsoft.com.

[24] "National insturments," www.ni.com.

[25] *HYSYS Customization Guide*, 2nd ed., AspenTech, 2000.

[26] K. Ogata, *Modern Control engineering*, third edition ed. Prentice Hall, 1997.

[27] J. Liberty, *Programming C Sharp*, 3rd ed. O'Reilly, 2003.

[28] B. Stroustrup, *The C++ Programming Language*, 3rd ed. Addison-Wesley, 1997.

[29] *PID Control Toolset User Manual*, National Instruments, 1997, 2001.

[30] J. W. Webb and R. A. Reis, *Programmable Logic Controllers, Principles and Applications*, 5th ed. Prentice Hall, 2003.

[31] "Schneider electric," www.modicon.com.

[32] *Modicon TSX Momentum I/O Base Users Guide*, 4th ed., Schneider Electric, 2003.

[33] *Concept User Manual*, 2nd ed., Schneider Electric, 1999.

[34] *OPC Data Access Automation Specification*, 2nd ed., OPC Foundation, 1998.

[35] S. L. Christos G Cassandras. *Introduction to Discrete Event Systems.* Kluwer Academic Publishers, 1999.