Abstract

The control of large, complicated engineered systems such as oil/gas pipelines, electric power grids, transportation systems, and manufacturing systems is increasingly being accomplished using sophisticated supervisory control and data acquisition (SCADA) systems. SCADA generally refers to an industrial control system which monitors and controls processes in real time. Oil pipeline supervisory control and data acquisition systems monitor and control the flow/transportation of oil through pipelines by collecting data from various sensors at a plant or in other remote locations. The SCADA system uses a Master Station which controls all the operations after going through the information/data collected by Remote Terminal Units from field instruments. Security is the main feature while developing a SCADA system for oil pipelines. In this project we validate security of SCADA system using two techniques simulation and emulation. The simulation and emulation of SCADA system for oil pipelines are developed using Eclipse IDE.
Chapter 1
Introduction

Today we are in a communication era where the technology of telecommunication is evolving in every minute. With the evolution of various technologies in the field of computer science engineering, in present day to day life, it has become a vital need to have a highly efficient, fast and effective control system in industrial sector. During the last decade, great advances have been made in areas such as computer software, hardware and communications to achieve this. These advances have facilitated the building of very sophisticated Supervisory Data Acquisition and Control systems for wide range industrial systems such as chemical, petrochemical and electric power production and distribution, water distribution, oil and gas pipelines, nuclear plants, intelligent buildings and vehicle traffic control [5]. A supervisory system provides the user with the capability to exercise control over a specific device and to confirm its performance in accordance with the directed action. The supervisory system encompasses all control, indicating, and associated telemetry equipments at the master station, and all of the complementary devices at the remote stations. The smallest supervisory system consists of a single motor, single remote usually referred as one-on-one system. With the increase of complexity of the system, the major change is usually in the number of RTUs to collect numerous data, number of application programs or special functions, and the sophistication and customization of the interface between the user and the system [4, 5].

Wide area operations require control and monitoring of complex and distributed functions which cover large geographical and remote areas. SCADA system process can be infrastructure, industrial or facility based on the requirement in public or private sectors such as oil and gas pipelines, nuclear plants, intelligent buildings and vehicle traffic control. Industrial
control system (SCADA system) is basically a computer system for monitoring and controlling a process. Wide area controlling and monitoring systems are essentially based on the SCADA system [1].

Modern SCADA systems typically possess a hierarchical structure characterized by continuous dynamics at the lower levels and logical decision-making at the upper levels. SCADA accomplish functions that include supervision and real-time control of local or remote processes. These systems are composed of computers, software and devices used for data acquisition and digital input/output in order to perform process interaction.

These systems were traditionally monolithic and “closed” systems, using proprietary protocols and communications, which often lacked adequate reporting functions that are important to manage system. After few years, SCADA systems have moved away from the traditional architecture, to an “open” standard, using communications technology, which can be managed and maintained by highly skilled employees [6].
Chapter 2

SCADA System

In this chapter we will go through the SCADA system and its applications in oil pipelines.

The acronym SCADA stands for Supervisory Control and Data Acquisition. The primary purpose of SCADA is to monitor, control and alarm plant or regional operating systems from a central location.

2.1 What makes up a SCADA system?

The major components of SCADA system are

- RTU’s (Remote Terminal Units)
- Communication Interface
- Master station and HMI (Human Machine Interface)

Remote Terminal Unit(s) acquire digital and analog measurements for SCADA system. RTUs are installed at various locations of different stations to acquire complete analog and digital data of the station. These RTUs are getting digital data from field instruments connected with relays to show and operate live status of circuit breakers. Each RTU is connected to sensors and collects information at selected locations of different stations, while communications bring that information from the various locations or regional RTU sites to a central system, and returns instructions to the RTU depending on a particular event. The RTUs must provide data reliability and data security.
Programmable Logic Controller(s) are field devices which are used for data acquisition, because they are more economical, versatile, flexible, and configurable than special-purpose RTUs.

The HMI displays this information in an easily understood graphics form, archives the data received, transmits alarms and permits operator control as required. The HMI is essentially a PC system running powerful graphic and alarm software programs.

A supervisory system (master station) communicates with the field equipment (RTUs, PLCs, etc), and then to the HMI software running on workstations in the control room.

Communication within a plant will be by data cable, wire or fiber-optic, while regional systems utilize radio. The basic SCADA system is as shown in Figure 1 [1, 7].

![Figure 1: Basic SCADA System]
2.2 The phases in a functional SCADA system

Phase 1: The design of the system architecture

This includes the all communication system, and with a regional system utilizing radio communication often involves a radio path survey. Site instrumentation is used to monitor desired parameters.

Phase 2:

This phase involves the supply of RTU, communication interface, HMI equipment, PC system and the necessary powerful graphic and alarm software programs.

Phase 3:

The programming of the communication equipment, HMI graphic and alarm software programs is performed in this phase.

Phase 4:

This involves the installation of the communication equipment and the PC system. The former task is typically much more involved.

Phase 5:

The commissioning of the system, during which communication and HMI programming problems are solved, the system is proven to the client, operator training and system documentation is provided in this phase.

The functional layers of SCADA architecture are shown in Figure 2, see [4].
SCADA systems are different from traditional Distributed Control Systems (DCS) which are generally found in plant sites. SCADA systems cover much larger geographic areas, whereas DCS's cover the plant site. Often SCADA Systems are required to interface to a plant site DCS if there are remote sensors, motors and pumps that must be controlled/monitored by the plant site DCS. Certain types of applications like those in Oil & Gas, Electrical & Water Utilities, Water &
Wastewater and Environmental Monitoring inherently require SCADA communications because of the remoteness of the assets (i.e. Oil wells, water wells, generator stations). Due to the remoteness many of these often require the use of wireless communications [1].

2.3 SCADA system for oil pipelines

Major energy resources like petroleum, oil and natural gas are transported through pipelines which are connected to issues like geopolitics and international security. Apart from these we also need to consider the construction and control of oil and gas pipelines as they run over thousands of miles and across some of the most volatile areas in the world. Getting oil from the well to the refinery and from there to the service station requires a complex transportation and storage system [3]. Thus modern SCADA systems are developed to overcome all these problems. One of the main features of the SCADA system for oil pipeline is to control the oil flow through the pipelines. Oil pipeline SCADA systems provide operators with many useful features such as emergency shutdown, leak detection and batch tracking which improves operators’ productivity.

As transportation pipelines move oil or gas from one place to other place, this network requires several pump stations to control the flow of crude oil through the pipelines. This can be achieved by placing various field instruments, programmable logic controllers and remote terminal units at different places throughout the pipeline. Master station uses this data to control the oil flow rate in oil pipelines.

Field devices are instrumentation and data gathering units. The field instrumentation includes flow, pressure and temperature gauges and other devices to measure the relevant data required. The field instruments are installed along the pipelines on specific locations. These field
instruments are connected to RTUs. The information measured by these field instruments is then gathered local RTUs which in turn transfer this data to a master station using communication systems. Pipelines are controlled and operated remotely by the master station. The data that is received from multiple RTUs along the pipeline is stored in a central database. The SCADA system at the master station receives all the field data and presents it to the pipeline operator through human machine interface, showing operational conditions of the pipeline. The operator monitors the conditions of the pipeline and sends appropriate operational commands such as increase/decrease pressure by opening/closing valves through the SCADA system to the field instruments. Since all these RTUs communicate with SCADA master system, rapid exchange of data takes place between field instruments and SCADA system [2].

Figure 3: The SCADA System for pipelines
Chapter 3

Simulation of SCADA systems for Oil Pipelines

This chapter briefly describes what simulation is and how simulation is implemented in SCADA system for oil pipelines.

Simulation is the technique of representing the real world by a computer program; "a simulation should imitate the internal processes and not merely the results of the thing being simulated". In simulation the system behavior can be studied by creating a mathematical model of the system. Then the system’s variables are altered to determine the effects on other variables. The mathematical model can be developed by writing special purpose computer program or using a general simulation package [8].

3.1 Oil Pipeline SCADA Simulator

In present industry there are various kinds of simulation packages are available such as simSCADA and simulink. Opal Software developed simSCADA, a software-based simulation package for SCADA which is widely used [9, 10, 11].

In the present scenario we develop a mathematical model for the SCADA based oil pipeline system which simulates the system. The main goal for developing a simulator is to analyze the behavior of SCADA based oil pipeline system. Therefore for analyzing, we develop a SCADA simulator for oil pipelines and make predictions about the system based on the mathematical model. SCADA simulator for oil pipelines is a single threaded program based on the described mathematical model.
3.1.1 The mathematical model:

The fluid state in pipelines can be determined by using equations of conservation of mass, energy and momentum. The basic governing equation for a steady state incompressible viscous flow in the pipeline is the one dimensional energy equation [5, 14]:

\[ P_{Total} = P_{Dynamic} + P_{Static} \]

The data used to estimate the total pressure are:

- Threshold pressure in bar(b) or dyn/cm2 ~ ~35-bars [13]
- Maximum pressure that a pipe can withstand in bar(b) or dyn/cm2 ~ ~120-bar [12]
- Static fluid pressure in bar(b) or dyn/cm2
- Dynamic fluid pressure in bar(b) or dyn/cm2
- Fluid velocity in cm/s
- Fluid density in g/cm3
- Distance travelled in cm

Here is a brief description of the SCADA simulator for oil pipelines (SSOP). In this we construct a mathematical model for predicting pressure variations and then manipulate various determinants of the pressure such as temperature, fluid velocity, density and distance travelled by the oil in pipelines to determine how these changes affect the pressure. Here we assumed that the temperature inside the oil pipeline is constant. In this SensorSystem class initiates the whole process. Initially the static fluid pressure was assumed to be 35 bars and the dynamic fluid
pressure was 0 bars since the pressure is at steady state. Once the SensorSystem thread initiates, it repeatedly verifies the pressure reading and observers whether it falls in the required pressure ranges or not. If not it adjusts the pressure reading so that its value falls in the given range. If the pressure value reaches threshold point, it adjusts the pressure reading to a threshold pressure. If the pressure value is above maximum pressure, then it adjusts the pressure reading to a maximum pressure. The class diagram for SSOP is shown in Figure 4. The Sequence diagram for SSOP is shown in Figure 5 and brief description is provided in Figure 6.

The complete code in Java for “SCADA Simulator for Oil Pipelines” is given in the Appendix C.

<table>
<thead>
<tr>
<th>SensorSystem class</th>
</tr>
</thead>
<tbody>
<tr>
<td>d: Double</td>
</tr>
<tr>
<td>pth: Integer</td>
</tr>
<tr>
<td>ps: Integer</td>
</tr>
<tr>
<td>pd: Integer</td>
</tr>
<tr>
<td>pt: Integer</td>
</tr>
<tr>
<td>pmax: Integer</td>
</tr>
<tr>
<td>t: Integer</td>
</tr>
<tr>
<td>v: Integer</td>
</tr>
<tr>
<td>x: Integer</td>
</tr>
<tr>
<td>temp: Integer</td>
</tr>
</tbody>
</table>

Figure 4: Class Diagram for SSOP
Figure 5: Sequence Diagram for SSOP
1. The SensorSystem class initiates the process. The user provides required data through console.

2. Console provides required data to SensorSystem class.

3. Then the SensorSystem control class repeatedly verifies the pressure reading and observes whether they fall in the required pressure range or not. If not it adjusts the pressure reading to a threshold value and recalculates pressure up to a certain time period.

4. The SensorSystem control class sends all the pressure values to the SensorSystem output report class.

5. Then the user is provided with a successful completion message of required operations through the console.

Figure 6: Message Description for Figure 5
Figure 7: Pressure values observed by SSOP and corresponding graphical representation of pressure values in bars with time
3.2 Security Analysis

Security is the ability of a system to withstand against danger and loss which is one of the important aspects of SCADA system. Since it is used in large sophisticated system such as oil, power plant systems, it is important to simulate all security scenarios that are possible before deploying in real time applications. This is to test the system’s ability under critical security conditions. The security of these devices is essential for both safety and business continuity. The move from proprietary technologies to more standardized and open solutions together with the increased number of connections between SCADA systems and office networks and the internet has made them more vulnerable to attacks.

Some of the security scenarios for an oil pipeline SCADA system include what would happen if authentication fails, what would happens if output results are altered (data integrity). What happens if the data provided was incorrect? We simulate the system response to each of these scenarios’ and augment the system to improve reliability.

3.2.1 Authentication scenario analysis

As SCADA systems are used in wide range industrial applications, only authorized personnel should be allowed to operate SSOP in order to ensure proper usage of the system. In order to allow authorized personnel, authentication credentials are used to validate users at logon. If the user enters wrong credentials, they will not be allowed to access the system. The sequence diagram for the authentication failure scenario is shown in Figure 8 and the messages description is given in Figure 9.
1. The user provides authentication credentials through console class.

2. If the provided details don’t match with the values in the records, then the users are not allowed to access the SensorSystem class.

3. The console class is provided with an unsuccessful message.

4. An unsuccessful message was sent to the user through the console class.

**Figure 8: Sequence diagram for the Authentication failure scenario**

**Figure 9: Message Description for Figure 8**
Figure 10: Output Screens
Figure 11: Sequence diagram for the Successful Authentication scenario
1. The user provides authentication credentials through console class.

2. The console class verifies authentication credentials.

3. If the provided details match with the values in the records, then the authorized personnel are allowed to initiate the SensorSystem class by providing required values.

4. Then the SensorSystem control class repeatedly verifies the pressure reading and observes whether they fall in the required pressure range or not. If not it adjusts the pressure reading to a threshold value/maximum value and recalculates pressure up to a certain time period and reports to SensorSystem output report class.

5. The SensorSystem output report class sends all pressure values to the SensorSystem control class.

6. These messages are transferred to authorized personnel.

7. The console displays all the messages provided by SensorSystem output class.

Figure 12: Message Description for Figure 11
3.2.2 Data Integrity scenario analysis

Once the authorized personnel access the system, the results are stored into a data file. These results are helpful in future reference while developing the actual system. If the data is altered we may end up with wrong results. In order to assure the data retrieved from the SensorSystem is safe, the data is encrypted to other format using cryptographic techniques, once SensorSystem takes the decision. So that no one can alter the data in future. The Sequence diagram for SSOP is shown in Figure 14 and brief description is provided in Figure 15.
Figure 14: Sequence diagram for the Data Integrity scenario

1. The user accesses the data control class through the console class.

2. The console class provides required data to the data control class.

3. The data control class performs the encryption technique and provides data to the output report class.

4. The output report class provides the results to the console class.

5. The console displays the results to the user.

Figure 15: Message Description for Figure 14
3.2.3 Damage to physical equipment scenario analysis

If there is a physical damage to the system, then the dynamic pressure may become zero as the fluid velocity approaches to zero. In this scenario the system is shutdown for a while so that we can prevent the system going into a constant dead state. The Sequence diagram for SSOP is shown in Figure 17 and brief description is provided in Figure 18.
1. The user access the data control class through the console class.

2. The console class provides required data to the data control class.

3. The SensorSystem control class performs all the arithmetic operations provided in the mathematical model and provides data to the output report class. If the SensorSystem
control class observers a sudden drop in the dynamic pressure it shutdowns system for a while.

4. The output report class provides the results to the console.

5. The console displays the results to the user.

Figure 18: Message Description for Figure 17

Figure 19: Output Screens
Chapter 4

Emulation of SCADA systems for Oil Pipelines

In this chapter we will discuss what emulation is and how SCADA emulator is implemented in oil pipelines.

*Emulation Architecture:*

Emulation is the process of imitation the function of another system, as by modifications to hardware or software that allow the imitating system to accept the same data, execute the same programs, and achieve the same results as the imitated system.

In this one system performs in the same way as another, through perhaps not at the same speed. We can use emulation as a replacement for a system

4.1 Oil Pipeline SCADA Emulator:

An emulator duplicates the functions of one system using a different system, so that the second system behaves like and appears to be the first system. There are various kinds of emulators are available such as Orion Emulator. The basic modules of an emulator are a CPU emulator, I/O devices emulators and a memory subsystem module. Using these components the emulator presents an imitated system. The main emphasis of developing an emulator is to provide secure transmission of data from field instruments to RTUs and then to master station. The other security issues like leak detection, authentication and cryptography are also considered while developing the system.

Therefore for analyzing security scenarios related to oil pipelines, we developed a SCADA emulator for oil pipelines to emulate SCADA system for oil pipelines. The SCADA
emulator for oil pipelines is a multithreaded application which allows message passing among the threads in Java. As multithreading enables parallel execution in the real world, the SCADA emulator for oil pipelines resembles real time system.

The SCADA system for oil pipelines can be described as follows: The class diagram of the SCADA emulator for oil pipelines (SEOP) is shown in figure 20. The SCADA system observes various parameters like pressure, temperature and density of the oil inside the pipeline. For the present system we assume temperature inside the pipeline is constant throughout the pipeline. Field instruments are used for measuring pressure. For this purpose, we use a thread namely pressureSensor. Other classes that are used to gather information from field instruments, transfer collected data to master station and to make appropriate decisions includes rtuProcessor, rtuCommunicator, masterCommunicator, masterProcessor, and pump. For security scenarios some other classes are added in addition to SensorSystem class.

The sequence diagram for SCADA system is shown in Figure 21 and brief description is provided in Figure 22. The SensorSystem class initiates the thread pressureSensor. The pressureSensor thread reads the pressure values of the oil flowing inside the pipeline and sends them to the rtuProcessor class. The rtuProcessor class sends these values to the rtuCommunicator class which are collected from the pressureSensor thread. The rtuCommunicator class sends these values to the masterCommunicator class. The masterCommunicator class sends these values to the masterProcessor class. The masterProcessor class writes these values to a file and checks these values. If the values are in range, then it sends a message “Safe” to rtuProcessor class. If the pressure value is falling low, then it sends an “Increase” message to the rtuProcessor class asking it to increase pressure. When the rtuProcessor class receives an “Increase” message from the masterProcessor class, it starts the pump class.
When the masterProcessor class sends a message to the rtuProcessor class, it follows the following path.

1. masterProcessor to masterCommunicator
2. masterCommunicator to rtuCommunicator
3. rtuCommunicator to rtuProcessor.

Similarly when the rtuProcessor class sends a message to the masterProcessor class, it follows the following path.

1. rtuProcessor to rtuCommunicator
2. rtuCommunicator to masterCommunicator
3. masterCommunicator to masterProcessor.

The complete code in Java for SEOP is given in the Appendix D.
Figure 20: Class Diagram for the SEOP
Figure 21: Sequence diagram for SEOP
1. pressureSensor thread reads the pressure inside the pipeline and sends the values to the rtuProcessor class.

2. rtuProcessor class verifies for the message from where it came. If the message is from the pressureSensor thread, it sends the pressure value to the rtuCommunicator class.

3. rtuCommunicator class checks for where the message came from. If the message is from the rtuProcessor class, it sends the pressure value to the masterCommunicator class.

4. The masterCommunicator class checks for where the message came from. If the message is from the rtuCommunicator class, it sends the pressure value to the masterProcessor class.

5. The masterProcessor class stores these values in a file and checks if the pressure values are in range. If they are in range it sends a “Safe” message to the masterCommunicator class. If the pressure value is low, it sends an “Increase” message to the masterCommunicator class.

6. The masterCommunicator class checks for where the message came from. If the message came from the masterProcessor class, it sends the same message to the rtuCommunicator class.

7. The rtuCommunicator class checks for where the message came from. If the message is from the masterCommunicator class, it sends the same message to the rtuProcessor class.

8. The rtuProcessor class checks for where the message came from. If the message is from the rtuCommunicator class, it checks for if the message is “Safe” or “Increase”. If the message is “Increase” it sends a message to the pump class asking it to activate.

**Figure 22: Message passing information details for Figure 21 among various threads**
The basic structure of the emulated system is as shown in Figure 23. The main parts of system are Sensors, Remote Terminal Unit and a Master Unit (SCADA system). Each Remote Terminal Unit consists of an rtuProcessor and an rtuCommunicator. We considered the vMBusX-SP wireless pressure sensor (datasheet given in Appendix A) and the C3-ilex 9300 family of Remote Terminal Units (RTU) (datasheet given in Appendix B) for emulation. The master consists of a masterProcessor and a masterCommunicator. There is a pressureSensor for each RTU and one pump for a set of RTU’s.

**Figure 23: Typical Oil Pipeline SCADA system architecture (C3-ilex)**
The sample output from the SEOP is shown in Figure 24. These are the pressure values at respective RTUs. The average pressure inside the pipeline is between 120 and 35 bars. We can see that there is gradual decrease in the readings as the oil moves through the pipeline; its velocity gradually decreases due to the friction between the oil and the surface of the pipeline. So as the pressure reading goes less than 35 bars, we need to start the pump (or motor) to increase the flow rate of the oil. So at 30 bars, when we start the pump the pressure increases to 120 bars and again gradually decreases as it moves.

Figure 24: Pressure Values Observed by SEOP
The graph for the pressure values would look like Figure 25. We can see that the pressure decreases gradually as the oil flows through the pipeline. When it reaches below 35 bars, the pump gets activated and boosts the oil, its pressure increases and then it again decreases gradually. The horizontal axis shows the RTU numbers and the vertical axis shows the pressure values in bars.

Figure 25: Graphical representation of pressure values in bars with time at each RTU


4.2 Security Analysis

SCADA technology is used for automated chemical, broadcasting, rail, power, gas, water and oil plant systems. The security of these devices is essential for both safety and business continuity. The move from proprietary technologies to more standardized and open solutions together with the increased number of connections between SCADA systems and office networks and the internet has made them more vulnerable to attacks.

Security researchers are concerned mainly about:

- Authentication in the design
- Cryptographic techniques
- Damage to the physical equipment

4.2.1 Authentication scenario analysis

The security of these SCADA systems is important because compromise or destruction of these systems would impact multiple areas of society far removed from the original compromise. In this project we provide cryptographic mechanism to the SensorSystem to secure SCADA system for oil pipelines. In order provide secure mechanism for the system we generate a authentication key for the system. This can be achieved by taking another class Generatekey which calls encoder to encrypt the message. In this initially administrator of the system generates a key for all the authenticated users using specified encryption key. These encrypted keys are stored into an output file which is used in the SensorSystem class. Once the administrator generates encrypted keys for all the authenticated users, the registered keys are placed in the SensorSystem class. When the registered user accesses the SEOP, it asks for the user credentials.
If the user credentials match with the administrator generated keys, then the system will be further accessed. In this manner we can provide Authentication to the secure SCADA system for oil pipelines. The sequence diagram and output results are shown in Figure 26 and 27 respectively.
1. The user provides authentication credentials through console class.

2. The console class verifies authentication credentials.

3. If the provided details match with the values in the records, then the authorized personnel are allowed to initiate the SensorSystem class by providing required values.

4. Then the SensorSystem control class which includes masterProcessor repeatedly verifies the pressure reading and observes whether they fall in the required pressure range or not. If not it adjusts the pressure reading to a threshold value/maximum value and recalculates pressure up to a certain time period and reports to SensorSystem output report class.

5. The SensorSystem output report class sends all pressure values to the SensorSystem control class.

6. These messages are transferred to authorized personnel.

7. The console displays all the messages provided by SensorSystem output class.

Figure 26: Sequence Diagram and corresponding message description for SEOP
4.2.2 Data Integrity scenario analysis

In order to assure the data retrieved from the RTUs is safe, the data is encrypted to other format using cryptographic techniques, once SensorSystem takes the decision. So that no one can alter the data in future. The Sequence diagram for SEOP is shown in Figure.
1. The user access the data control class through the console class.

2. The console class provides required data to the data control class.

3. The data control class performs the encryption technique and provides data to the output report class.

4. The output report class provides the results to the console class.

5. The console displays the results to the user.

Figure 28: Sequence diagram for the Data Integrity scenario and corresponding message description
4.2.3 Damage to Physical Equipment Scenario

There are also chances of damage to physical equipment such as pump failure. When there is a leak in the pump (it can be made by other persons), then the pressure readings would drop continuously. Due to the friction between the oil and pipeline inside the pipeline its velocity decreases gradually and if no observes this failure, it takes forever for the oil to move from the source to the destination. One alternative for this scenario is to adjust the pressure values automatically when it falls below minimum pressure value or we can use a backup pump. Following figures show the sequence diagram for the pump failure scenario and the corresponding messages.
Figure 30: Sequence diagram for the Pump failure
1. pressureSensor reads the pressure inside the pipeline sends the values to the rtuProcessor.

2. rtuProcessor sends the pressure value to the rtuCommunicator.

3. rtuCommunicator checks sends the pressure value to the masterCommunicator.

4. The masterCommunicator sends the pressure values to the masterProcessor.

5. The masterProcessor stores these values in a file and checks if the pressure values are in range. If they are in range it sends a “Safe” message to the masterCommunicator. If the pressure value is low, it sends an “Increase” message to the masterCommunicator.

6. The masterCommunicator sends the same message to the rtuCommunicator.

7. The rtuCommunicator sends the same message to the rtuProcessor.

8. The rtuProcessor checks for if the message is “Safe” or “Increase”. If the message is “Increase” it sends a message to the backupMotor asking it to activate.

**Figure 31: Messages Description for Figure 30**
Figure 32: Output for the pump failure scenario (using backup pump)

Figure 33: Graphical representation of pressure values in bars with time at each RTU for the pump failure scenario (using backup pump)
If we do not adjust pressure values then the pressure values drop continuously as shown in Figure 34.

Figure 34: Graphical representation without backup pump during failure
Chapter 5

Conclusions and Future Work

In this project we analyzed the SCADA system for oil pipelines using both simulator and emulator. First we analyzed the system using SSOP. In this we developed a mathematical model using all the parameters that affect the functioning of the system. Here simulation involves simulating all the devices within system. In this we analyzed the system and make predictions about it the process. Then we analyzed the system using SEOP. In the emulation some functional part of the model is carried out by a part of the real system. Since it is an approximation of real system, there exists a measurable difference between the performance of the real system and the model. Emulation presents a virtual system.

Simulation models provide imbalance between experiences and new ideas. They are used to test and develop different solutions in order to arrive at a best solution, based on test cases. Using simulation we can demonstrate functionality and results in a flexible environment. Whereas Emulation models are used to define the system more precisely. These systems require highly skilled employees to run. Emulation models are not used for experimentation in the same way that simulation models are.

The emulation model reflects more precisely the system that will be implemented and uses verification procedures to ensure the performance or reaction of the control system. System exists to automate the execution of these tests, and runs them in parallel in order reflect like a real time system. Whereas simulation can’t be implemented as a real time system.

Simulation model provides a high speed execution and covers all different possibilities. Emulation model provides a real time execution. Although simulation models provide responses
faster than real time, control systems timers cannot adopt this. Therefore running at speeds greater than real time should be avoided. Simulation provides instant decisions. Emulation model reflects reality and incorporate decisions that take a finite time. Depending on the available resources we can analyze the system using an emulator or simulator.

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Security Improvement Technique</th>
<th>Effect on performance</th>
<th>Effect on cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authentication</td>
<td>Authentication techniques are introduced to ensure authorized people are allowed</td>
<td>Speed reduces as additional data is needed to access the sensor system master station</td>
<td>A small increase in cost as we use additional techniques</td>
</tr>
<tr>
<td>Data Integrity</td>
<td>Cryptographic techniques are introduced to ensure actual data transmission</td>
<td>Speed decreases as output data is further converted to another format</td>
<td>Cost increases slightly</td>
</tr>
<tr>
<td>Physical Equipment Damage</td>
<td>Backup pump</td>
<td>Performance decreases since we need to decide which pump to use</td>
<td>Cost increases as we need another pump</td>
</tr>
</tbody>
</table>

Table 1: Security Scenarios and Improvement Techniques
The main purpose of developing SCADA systems is to reduce the man power and to increase the efficiency of industrial systems in remote areas such as deserts, nuclear power plant and undersea in real time. SCADA systems became more popular in late 1980’s. Before the existence of SCADA systems, we need to send the people around every plant location to collect the data. Communication between the system functions were conducted over phone from an operator at the plant and the central facilities. The first generation SCADA systems are closed systems which are less vulnerable, but lacked adequate reported functions that are important to manage the system which used proprietary protocols and communications. With the evolution of new technologies, the traditional closed systems are converted to open systems using standard communication technology and protocols. Due to the usage of standard protocols, the systems are potentially vulnerable to remote cyber attacks. In this project we discussed some of the security scenarios such as authentication, data integrity problems and developed a system to overcome these scenarios.

In the present simulation scenario, we assumed the pipe to be relatively straight, such that changes in pressure due to elevation and wall friction are ignored and calculated the total pressure in a section of uniform pipe. In the future work we can generalize this equation by using the variables head loss, pressure losses due to friction and minor losses like losses in bends, branches, valves etc. As boundary conditions affect the flow characteristics we can include boundary conditions to achieve probability distributed functions. This system can be further developed to work on some other security issues like architecture issues and network protocol structure depending on the availability of resources. The next generation SCADA and software will be focused to provide strategic information to decision makers in a real time environment with more accuracy using more secured standard protocols.
Chapter 6

Bibliography

9300 RTU Family Features

Hardware:
- 9310 RTU for small or pole mount applications
- 9300 card cage design allows modular selection of I/O card types, including Indication, Control, DC Analog, Transducerless AC Analog, and IEDs
- 6 Slot design for small applications
- 10 Slot 19" Rack Mount design for larger configurations
- Optional Fiber Optic Distributed Chassis design
- Power Supply Voltages:
  - VDC: 12, 24, 48, 125
  - VAC: 120, 220 with battery back up
- 386 processor with RAM memory
- IEEE / ANSI C37.90.1 SWC Surge protection

Firmware:
- Self Scanning, Real-time Exception reporting
- Software Configurable Parameters
- Updates downloadable into Flash Memory
- Built-in internal diagnostic card alarms and LEDs
- Optional 1 ms Sequence of Events (SOE) resolution

Communications:
- On board RS-232 and internal 1200 bps Modem
- 10/100 Mbs WAN card – DNP 3.0, TCP/IP, Ethernet
- Dial-up Communication Port
- Expandable to 4 separate master Communication ports
- Software Configurable Modes & Data rates

IED support:
- Generic DNP 3.0 configuration tool kit
- Protocols: DNP 3.0, Modbus, PG&E 2179 and many more
- Cooper, SEL, ABB, Siemens and many others
- Onboard Fiber, RS-232, and RS-485 connectors

RTU Protocols:
- DNP 3.0
- ComTel 2020
- L&G 8979
- Cannon IDLC
- Harris 5000
- Large Library Available

Service:
- Repair facility
- Help Desk support
- Web site updates and FAQs

Installation:
- Onsite or phone support
- 19" rack mountable
- Field wiring conversion kits

Select the 9310 Single Board RTU for small or pole mount applications

Choose our 9300 6 slot or 10 slot RTU for larger applications

Just add the cards you need for your application!

Add additional chassis and cards for future expansion or larger applications

Easily match your I/O needs through our modular design
9300 Capacity

The 9300 RTU supports up to:
- 1024 Indication inputs
- 1024 Sequence of Events inputs
- 512 SBO Relay Control outputs
- 50 Raise/Lower Controls
- 512 Analog inputs (combination AC and DC analog inputs)
- 256 Analog outputs
- 256 Pulse Accumulators
- 4 RTU to Master Comm. channels
- 16 IED Interface ports
  - Internal 1200 bps Modem
  - Dial-up Modem

9310 Capacity

The 9310 RTU supports:
- 8 Indication or Accumulator inputs
- 4 SBO Controls or 8 relays
- 8 DC analog inputs
- 120 VAC or 12 VDC power
- Onboard battery charger
- RS-232 Comm. Interface
- IED Interface ports
- 1200 bps Modem
- Dial-up Modem

RTU Configuration Software

Setup, define, and maintain your RTU Configuration with our easy-to-use software.
- Up & down-load data to and from the RTU
- Store individual RTU Configuration files
- Define Multiple Master communication Ports with separate Protocols for each port
- Define individual points with a separate setup page
- Select & define individual IED points
- Generic DNP 3.0 Tool Kit
- Software available on CD or from our Web site

Diagnostic Test Set Software

Provides a user friendly approach to diagnosing the health of RTU, Communication links, and Master connections.
- Tests all RTU functions, provides individual RTU card diagnostics, and listens in on communications with the master
- Allows the user to create a test database including Multiple IEDs
- Provides visual confirmation of data transmissions
- Displays Sent & Received messages both to and from the RTU
- Connects via RS-232 or Modem
C3-ilex provides high-quality products and services for the real-time monitoring and control market place. Our employees are highly skilled in a variety of technologies and applications. Our products span from data collection and control to end-user applications for system management. We also offer numerous services to help integrate legacy systems with new technologies. We welcome the opportunity to learn about your technology needs and to create cost-effective solutions to achieve your goals.

For more information on C3-ilex products and services, please find us on the web at www.c3ilex.com or contact us at:

C3-ilex, LLC (Headquarters)
47009 Benicia Street
Fremont, CA 94538
Phone: (510) 650-8300
Fax: (510) 650-8302

C3-ilex, LLC
1519 North 23rd Street
Wilmington, NC 28405
Phone: (910) 251-1330
Fax: (910) 251-1107
Appendix B

vMBusX-SP - is a solution to meet the demands of the oil and gas industry; for a reliable and cost effective means to remotely monitor their applications. The vMBusX-SP incorporates our powerful VM-Micro RTU with our proven battery operated wireless pressure sensor to produce a small, low power, explosion proof RTU-Sensor system that can be mounted directly to the process line (e.g. wellheads, pipelines, gas heads, etc.), combined with a Gateway, the vMBusX-SP, installed at the remote site, can acquire real-time data for analysis and detect alarms. Easily customizable for many applications needing a wireless pressure sensor.

General Specifications

**Inputs**
- 2 Analog channels: 0-5 VDC or 0-20 mA
- 1 Opto-Isolated Digital Input Channels (3-24 VAC or DC)
- Digital channel monitors the status of a digital switch used for task monitoring & detecting alarms
- Independent 1/A channel used for battery voltage level
- Independent 1/A channel used for temperature
- Independent 1/A channel used for radio diagnostics

**Processor and Memory**
- 6552 Micro-Controller running at 12.5 MHz with 64 KB Flash Memory

**Radio**
- Plug-in radio modem operating at a frequency of 900 MHz: 2.4 GHz or Zigbee

**Serial Ports**
- 1 Serial Interface port bandwidth up to 115,200 bps

**Relays**
- 1 Solid State Relays that can support up to 350 mA continuous current
- On-Chip Filtering

**Power**
- Input Range 3.3 to 12.2 VDC
- Low Power-Operating Mode with <20 µA in sleep mode with wake up timer running
- Terminal blocks for providing regulated 6.25 V ± 10 V to power sensors (providing up to 200 mA to power sensors or devices)
- Power consumption <10 mA without radio

**Operation Modes**
- Continuous
- Stand-alone
- Timer/Count
- On-board
- Wake-up timer adjustable from 1 sec. to 255 hours

**Operating Temperature**
- -40°C to +85°C (-40°F to +185°F)
- Humidity Range 5-90% non-condensing

**Dimensions**
- 6.0 cm x 6.0 cm

**Support**
- vMBus Communications Protocol
- MODBUS Protocol Support

**Firmware**
- Linux OS, True Counter/Accumulator, Square Root Extraction, Engineering Units, Conversion/Scaling

For more information on our products and services or to place an order, please check out our website at www.vmonitor.com or contact us at +1.866.514.4935 to speak to one of our engineers. Let us help you make the virtual oilfield a reality.
vMBusX-SP

Wireless Pressure Sensor

**Radio Specifications for vMBusX-SP** (Range and Transmit Power may vary due to various factors, such as type and elevation of antenna, line of sight, the environment, and power output. A vMonitor Engineer can assist you with determining range, frequency, transmit power and other factors to develop a solution to meet your requirements.)

<table>
<thead>
<tr>
<th>Compatible Radios</th>
<th>Range</th>
<th>Transmit Power</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.4 GHz</td>
<td>Up to 16 km (10 miles)</td>
<td>50 mW-100 mW</td>
</tr>
<tr>
<td>2.4 GHz ZigBee</td>
<td>Up to 0.12 km (400 ft)</td>
<td>125 mW</td>
</tr>
<tr>
<td>900 MHz</td>
<td>Up to 32 km (20 miles)</td>
<td>5 mW-1000 mW variable</td>
</tr>
</tbody>
</table>

Other Wireless Products from the vMBus Family

- **vMBusX-G** (Gateway)
- **vMBusX-C RTU Rev. 5.1** (5AD, 2AO, 2DI, 2DO, ETD, BAT)
- **vMBus-X Nano RTU Rev. 2.1** (1AD, 1DI, 1DO, 1ETD, 1BAT, RSSI)
- **vMBusX-NM10 RTU Rev. 1.0** (8AD, 8DI, 2FC/TC, 6RLY, 6AO, R5405/R5232, I2C)
- **vMBusX-SP RTU Rev. 3.0** (6AD, 1AO, 2DI, 2RLY, 2RTD, BAT)

**Keys:**
- AD  Analog Input
- AO  Analog Output
- DI  Digital Input
- DO  Digital Output
- FC  Frequency Counter
- PC  Pulse Counter
- RLY Solid State Relay
- RTD Resistance Temp Device
- RTU Remote Terminal Unit
- TM  Tank Monitoring

**Typical vMonitor Network Setup**

For more information on our products and services or to place an order, please check out our website at www.vMonitor.com or contact us at +1.866.514.4935 to speak to one of our engineers. Let us help you make the virtual oilfield a reality.
Wireless Pressure Sensor

Specifications

<table>
<thead>
<tr>
<th>Specifications without model designation apply for all models.</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Pressure range</td>
<td></td>
</tr>
<tr>
<td>Minimum pressure</td>
<td></td>
</tr>
<tr>
<td>Maximum pressure</td>
<td></td>
</tr>
<tr>
<td>Burst pressure**</td>
<td></td>
</tr>
<tr>
<td>Pressure range</td>
<td></td>
</tr>
<tr>
<td>Maximum pressure</td>
<td></td>
</tr>
<tr>
<td>Burst pressure**</td>
<td></td>
</tr>
<tr>
<td>Maximum pressure</td>
<td></td>
</tr>
<tr>
<td>Burst pressure**</td>
<td></td>
</tr>
</tbody>
</table>

(vacuum, gauge pressure, compound ranges, and absolute pressure references are available)

Range only available with Type 3-20.

For type 5-21 the Burst pressure is limited to 3000PSI unless the pressure gas is accomplished by using the sealing ring underneath the hex.

*Pressures applied up to the maximum rating will cause permanent changes in specifications but may lead to zero and span drifts.

**Exceeding the burst pressure may result in destruction of the transmitter and possible loss of media.

Materials

- **Welded parts**
  - Stainless steel
  - (for other materials see WHA diaphragm seal program)

- **Diaphragm**
  - Models 8-21: 316L SS
  - Models 15-21: 316L SS (Hamershaw CD)

- **Casing**
  - Stainless steel
  - (for other materials see WHA diaphragm seal program)

- **Internal transmission fluid**
  - Synthetic oil (fluorocarbon oil for oxygen applications) (See Table 5-21 for fluid applications)

Power supply $U_s$

- DC 5
- $10 < U_s < 50$ (11 $U_s < 50$ with Model 15-21 F)

Signal output and maximum load $R_L$

- $F_a = \frac{U_a}{U_0}$
- $F_a = \frac{U_a}{U_0}$

Accuracy

- $\pm 0.05$ of span (for pressure ranges above 1000 PSI)

Repeatability

- $\pm 0.05$

1-year stability

- $\pm 0.2$ (at reference conditions)

Remote Monitoring Telemetry Oil & Gas Utility Waste Water & Water

Monitor

Creating the virtual oilfield.

For more information on our products and services or to place an order, please check out our website at www.vMonitor.com or contact us at +1.866.514.4935 to speak to one of our engineers. Let us help you make the virtual oilfield a reality.
Appendix C

Testcase.java

import javax.swing.*;
import java.io.*;
import java.util.*;
import java.math.*;
import java.lang.*;
import java.text.*;

public class testcase{

    public static void main(String[] args) throws IOException{

        int pth = 35;  //threshold pressure in bar(b) or dyn/cm2
        int pmax = 120;  //maximum pressure that a pipe can withstand in bar(b) or dyn/cm2
        int ps;  //static fluid pressure in bar(b) or dyn/cm2
        int pd;  //dynamic fluid pressure in bar(b) or dyn/cm2
        int pt;  //total fluid pressure in bar(b) or dyn/cm2
        int v;  //fluid velocity in cm/s
        double d;  //fluid density g/cm3
        int x;  //distance travelled in cm
        int t;  //time variable
        int temp = 10;  //temporary variable

        Scanner input = new Scanner(System.in);

        pd = 0;
        ps = 35;
pt = pd + ps;
"Total Pressure in pipes initially " + pt);

StringBuffer sb = new StringBuffer();
d = 4;
x = 0;
for(t = 1; t <= 25; ++t)
{
x = 25;
if( pt > pth && pt < pmax)
{
    v = (int)x / t; //calculates the fluid velocity
    pd = (int)(d*v*v) / 2; //calculate the dynamic pressure
    pt = pd + ps; //calculate the total pressure
    System.out.println("Total Pressure at t = " + t + " is " + pt);
}
else if(pt <= pth)
{
    pt = pth + pt;
    System.out.println("Increase the pressure reading");
    System.out.println("Total Pressure at t = " + t + " is " + pt);
}
else if(pt >= pmax)
{
pt = pmax - temp;

System.out.println("Decrease the pressure reading");

System.out.println("Total Pressure at t = " + t + " is " + pt);

sb.append(pt + "\t");

//writing all the pressure values to the file named "pressureoutput.txt"

for(t = 1; t <= 15; ++t)
{
    try {
        FileWriter fstream = new FileWriter("pressureoutput.txt");
        BufferedWriter out = new BufferedWriter(fstream);
        out.write("The pressure values are : " + " \t " + sb);
        out.close();
    } catch (Exception e) {
        //TO DO handle exception
    }
}

}
Appendix D

PressureSensor.java

import java.util.Observable;
import java.util.Observer;

public class PressureSensor extends Observable implements Runnable, Observer {
    int pressure = 120;

    public PressureSensor(String name) {
        Thread prThread = new Thread(this);
        prThread.start();
    }

    public void run() {
        int i = 0;
        while(++i <= 10) {
            pressure -= 15;
            deleteObservers();
            addObserver(SensorSystem.getRtuProcessor());
            synchronized (SensorSystem.lock) {
                System.out.println("Reading by Pressure Sensor = " + pressure);
                setChanged();
                notifyObservers(new Integer(pressure));
                SensorSystem.lock.notify();
            }
            System.out.println("Pressure Sensor Cycle completed");
        }
    }
}
synchronized public void update(Observable rtuProcessor, Object value) {
    if (pressure < 35) {
        System.out.println("Resetting Pressure");
        pressure = 120;
    }
}

Pump.java

import java.util.Observable;
import java.util.Observer;

public class Pump extends Observable implements Observer {
    public Observer getSensor() {
        return sensor;
    }

    public void setSensor(Observer sensor) {
        this.sensor = sensor;
    }

    Observer sensor;

    public void update(Observable rtuProcessor, Object value) {
        System.out.println("RtuProcessor ------ > Pump : " + value);
        String strValue = (String) value;
    }
}
if (strValue.equals("increase")) {
    addObserver(sensor);
    System.out.println("Activating pump");
    setChanged();
    notifyObservers("increase");
}

RtuCommunicator.java

import java.util.Observable;
import java.util.Observer;

public class RtuCommunicator extends Observable implements Observer{
    synchronized public void update(Observable sender, Object value) {
        deleteObservers();
        if (sender instanceof RtuProcessor) {
            addObserver(SensorSystem.getMasterCommunicator());
            System.out.println("RtuProcessor \-------- > RtuCommunicator : " + value);
            setChanged();
            notifyObservers(value);
        } else if (sender instanceof MasterCommunicator) {
            addObserver(SensorSystem.getRtuProcessor());
        }
    }
}
```java
import java.util.Observable;
import java.util.Observer;

public class RtuProcessor extends Observable implements Observer {
    private Pump pump = new Pump();
    private Object lock = new Object();
    private Observer sensor;

    synchronized public void update(Observable sender, Object value) {
        if (sender instanceof PressureSensor) {
            sensor = (Observer) sender;
            pump.setSensor(sensor);
            synchronized (lock) {
                deleteObservers();
                addObserver(SensorSystem.getRtuCommunicator());
            }
            System.out.println("PressureSensor -------- > RtuProcessor : " + value);
        }
    }
}
```

RtuProcessor.java
setChanged();
notifyObservers(value);
}
}
} else if (sender instanceof RtuCommunicator) {
synchronized (lock) {
deleteObservers();
String strValue = (String) value;
System.out.println("RtuCommunicator -------- > RtuProcessor: "+ value);
if (strValue.equals("increase")) {
addObserver(pump);
addObserver(sensor);
setChanged();
notifyObservers("increase");
} else if (strValue.equals("safe")) {

//System.out.println("RtuCommunicator -------- > RtuProcessor: Pressure is Safe");

}
MasterCommunicator.java

import java.util.Observable;
import java.util.Observer;

public class MasterCommunicator extends Observable implements Observer{
    synchronized public void update(Observable sender, Object value) {
        deleteObservers();

        if (sender instanceof RtuCommunicator) {
            addObserver(SensorSystem.getMasterProcessor());
            System.out.println("RtuCommunicator -------- > MasterCommunicator :");
            setChanged();
            notifyObservers(value);
        } else if (sender instanceof MasterProcessor) {
            addObserver(SensorSystem.getRtuCommunicator());
            System.out.println("MasterProcessor -------- > MasterCommunicator :");
            setChanged();
            notifyObservers(value);
        }
    }
}

MasterProcessor.java

import java.io.BufferedWriter;
import java.io.FileWriter;
import java.util.Observable;
import java.util.Observer;
import java.util.Vector;

public class MasterProcessor extends Observable implements Observer{
    Vector values = new Vector();
    private int minPressure = 35;
    synchronized public void update(Observable reader, Object value) {
        deleteObservers();
        addObserver(SensorSystem.getMasterCommunicator());
        if (reader instanceof MasterCommunicator) {
            System.out.println("MasterCommunicator -------- > MasterProcessor : " + value);
            values.add(value);
            StringBuffer sb = new StringBuffer();
            for (int i = 0; i < values.size(); i++) {
                sb.append(values.get(i) + "\t");
            }
            try {
                FileWriter fstream = new FileWriter("output.txt");
                BufferedWriter out = new BufferedWriter(fstream);
                out.write(sb.toString());
                out.close();
            } catch (Exception e) {// TODO: handle exception
            }
            int reading = ((Integer) value).intValue();
if (reading >= minPressure) {
    deleteObservers();
    addObserver(SensorSystem.getMasterCommunicator());
    setChanged();
    notifyObservers("safe");
} else if (reading < minPressure) {
    deleteObservers();
    addObserver(SensorSystem.getMasterCommunicator());
    setChanged();
    notifyObservers("increase");
}

SensorSystem.java

public class SensorSystem {

    private static MasterProcessor masterProcessor = new MasterProcessor();
    private static MasterCommunicator masterCommunicator = new MasterCommunicator();
    private static RtuProcessor rtuProcessor = new RtuProcessor();
    private static RtuCommunicator rtuCommunicator = new RtuCommunicator();
    private static Object lock = new Object();
    private static Object pressureLock = new Object();

    public static MasterProcessor getMasterProcessor() {

return masterProcessor;
}

public static MasterCommunicator getMasterCommunicator() {
    return masterCommunicator;
}

public static RtuProcessor getRtuProcessor() {
    return rtuProcessor;
}

synchronized public static RtuCommunicator getRtuCommunicator() {
    return rtuCommunicator;
}

public static int getI() {
    return i;
}

static int i=0;

public static void main(String[] args)
{
    PressureSensor ps = new PressureSensor("PressureSensor1");
}
}