



University of
Stavanger

Faculty of Science and Technology

MASTER'S THESIS

Study program/ Specialization: Computer Science	Spring semester, 2011 Open
Writer: Reyhaneh Ghergherehchi (Writer's signature)
Faculty supervisors: Chunming Rong (UiS) Tomasz Wiktor Wlodarczyk (UiS) External supervisor: Børre Heggernes (Amitec)	
Titel of thesis: Benefits of Using Stream Insight and Fuzzy Logic with PI system for Industrial Alarm System	
Credits (ECTS): 30	
Key words: Real-time monitoring Electrical Submersible Pump OSIsoft PI system Microsoft StreamInsight Fuzzy Logic Complex event processing	Pages: 80 Stavanger, June 2011 30.06.2011

ACKNOWLEDGMENT

Any attempt at any level cannot be satisfactorily completed without the support and guidance of learned people. I would like to express my gratitude to Professor Chunming Rong and Research Fellow, Tomasz Wiktor Włodarczyk from University of Stavanger and Børre Heggernes from Amitec for their constant support and motivation that has encouraged me to come up with thesis.

My deepest gratefulness goes to my parents, my sister and brother-in-law for their tenacious love and support throughout my life specially these two years of master study in Norway. At the end I would like to thank the following people for their help during this period:

- Roman Shindlauer
- Erwin Gove
- Bodil Sømme
- Shahram Aminzade

ABSTRACT

Pumps and their associated systems are essential in oil and gas facilities for the efficient transportation of fluids. Monitoring and controlling an ESP pump is vital and essential in management of ESP. Reliable and efficient alarm system with low-latency and real-time monitoring, helps operators to assess pump performance, avoid malfunction and result in increase in product and decrease in cost.

One of the objectives of this thesis is to make an alarm system which will be used for monitoring an ESP pump implemented for Talisman Energy Company using OSIsoft PI ACE.

The same target has been achieved by developing an application using different technology namely StreamInsight which is a complex event processing engine. The results and performance of these two technologies are discussed and compared.

Also a fuzzy logic solution is proposed to add functionality to alarm system and improve the pump life time and performance.

TABLE OF CONTENTS

Acknowledgment	i
Abstract.....	ii
Table of Figures	v
1 Introduction	1
1.1 Background.....	1
1.2 Related Work.....	1
1.3 Motivation.....	2
1.4 Thesis Overview	3
2 Technologies.....	4
2.1 Electronic Submersible Pump.....	5
2.1.1 Surveillance and Monitoring of ESP	7
2.1.2 The Reasons and Importance of Monitoring and Surveillance.....	8
2.1.3 Start up and Shutdown the ESP	8
2.1.4 Temperature.....	9
2.1.5 Gas Lock	9
2.1.6 Alarm conditions	9
2.2 PI System.....	12
2.2.1 PI Interface Node	12
2.2.2 PI Server	14
2.2.3 PI Points and PI Tags.....	16
2.2.4 Point Attributes.....	16
2.2.5 Time in PI.....	17
2.2.6 PI Process Book	18
2.2.7 PI Performance Equation.....	18
2.2.8 PI Advanced Computing Engine.....	18
2.2.9 Advantages of ACE over Performance Equation.....	22
2.3 Fuzzy Logic.....	23
2.3.1 Fuzzy Set and Membership Function	23
2.3.2 Fuzzy Operators	23
2.3.3 Linguistic Variables and Fuzzy Rules	24
2.3.4 Fuzzification	25
2.3.5 Inferencing.....	25
2.3.6 Defuzzification.....	26
2.3.7 Fuzzy Controller.....	26
2.4 Complex Event Processing and Stream Insight.....	28
2.4.1 Events.....	29
2.4.2 Adapters.....	29
2.4.3 Query.....	30

3	Implementation	31
3.1	Alarm System Using ACE.....	32
3.1.1	Environment and System Prerequisites	32
3.1.2	Development of ACE Modules	32
3.2	Applying Fuzzy Logic	36
3.2.1	Environment and System Prerequisites	36
3.2.2	Development and Challenges	36
3.3	Alarm system Using StreamInsight and PI Adapters	40
3.3.1	Environment and System Prerequisites	40
3.3.2	Development and Challenges	40
4	Conclusion	45
4.1	Conclusion.....	46
4.2	Further work	47
5	References.....	48
	Appendix A: PI Point Attributes	50
	Appendix B: Membership Functions.....	55
	Appendix C: Code and Screen Dumps.....	58

TABLE OF FIGURES

Figure 1. Artificial Lift methods.....	5
Figure 2. Typical ESP configuration.....	6
Figure 3. ESP operations block diagram illustrating the slow and fast feedback loops.....	8
Figure 4. PI System data flow.....	12
Figure 5. Exception reporting.....	13
Figure 6. Data flow in PI.....	14
Figure 7. Attribute classes.....	17
Figure 8. ACE components and their common data.....	19
Figure 9. Structure of ACE Modules.....	20
Figure 10. StreamInsight architectural overview.....	28
Figure 11. Module database and aliases.....	33
Figure 12. Code for Reduced Inflow Alarm.....	34
Figure 13. Reduced Inflow Alarm output.....	35
Figure 14. Motor Current membership function.....	37
Figure 15. Motor Frequency membership function.....	37
Figure 16. The Frequency results after applying fuzzy logic.....	39
Figure 17. Results of average function for ACE and StreamInsight.....	43
Figure 18. Example membership functions for fuzzy Sets.....	57
Figure 19. ACE Wizard.....	58
Figure 20. Define module and context.....	59
Figure 21. Selecting the input and output tags.....	60
Figure 22. OutageBlock Alarm module.....	61
Figure 23. Debugging a module.....	62
Figure 24. Registering the module.....	63
Figure 25. ACE Manager.....	64
Figure 26. StreamInsight application output.....	79

1 INTRODUCTION

1.1 BACKGROUND

In mature fields, when there is not enough energy in reservoir for producing the oil and gas to the surface, we need to use artificial lifting. “More than 90% of producing oil wells require some form of artificial lift” (Bates, Cosad et al. 2004). One of the common used artificial lifting is Electrical Submersible Pump.

Monitoring of production and pump parameters is common use in the oil industry. In order to make the everlasting and fast decision, production engineers need to have interminable downhole monitoring systems. Real-time monitoring systems without doubt provide operators with the possibility of improvements of production conditions because the easy and fast access to data allows engineers to plan effective and timely counteractive jobs(Moffatt and Craig 2001).

1.2 RELATED WORK

Since the Electrical submersible pumps started to be used, one of the concerns has been the surveillance and monitoring of them. Several technologies, software, and methods have been used and developed to make this possible. One of the earliest ways was the use of data which was recorded throughout various paper files. This made it very hard and takes long time to record run day averages, track failure modes, analyze past performance and deal with production problems(Gray 1994) and later using the relational data bases.

Schlumberger has developed the espWatcher surveillance system for Electrical Submersible pumps to connect production teams to their wells and fields in real-time, even on wells without SCADA installations. An alert and alarm system instantly warns users of developing problems, expediting corrective planning and minimizing pump downtime. The espWatcher system allowed real-time data polling and historical interrogation of stored data for pump and well-performance analysis. The result of implementing this system was moving production management from a reactive process to one that is proactive. Also ESP surveillance helped to keep the oil flowing from the reservoir and ensured healthy wells, fields and reservoirs(Bates, Cosad et al. 2004).

TOTAL lunched a corporate program called “Field Monitoring” to profit from affiliate previous experience. For gas-lifted wells, it resulted in the development of the “Well Performance Monitoring” (WPM) tool implemented in December 2006 at Sendji field located offshore Cango. This tool deployed at corporate level and extended to ESP-lifted wells, gas injectors or producers, and to subsea wells in deep offshore environment. The WPM application was essentially based on the real-time data historian PI commercialized by OSIsoft, which benefits from their corresponding tools, primarily data compression and storage. The system provided an easy access to data and ensured an improved detection and reactivity to production and well performance. The result was 2% gain in total current(Danquigny, Daian et al. 2007).

Conoco Philips and CNOOC (O.C Olmstead 2009) for the project in China in 2009 developed ESP monitoring and alarm Philosophy for manned multi-platform operation which had over 150 producing wells using ESP. The philosophy wraps up monitoring and alarms requirement for operation engineering. To achieve this, software called Life of Well

Information system (LOWIS by Weatherford production optimization) has been used and ensured that reliable and sufficient operation of ESPs is accruing throughout the asset(O.C Olmstead 2009)

Camilleri and Macdonald in their paper (Camilleri and Macdonald 2010) explained the development and implementation in the Schlumberger artificial lift real-time surveillance centre of a workflow to track ESP alarms from their initial detection through their classification to the analysis of their root-cause. The workflow uses a QHSE database with Web access to enable collaboration between the surveillance centre and field locations throughout Europe and Africa. The consequences of this real-time surveillance was t an increase in ESP run life, by preventing the ESP from being miss operated and experiencing excessive stress.

“Monitoring a producing well implies the ability to track, in real-time, any changes in fluid composition, flow rates, or pressure and temperature profiles. Multiphase Flow Metering (MFM) plays a key role in this scenario”(Alimonti and Falcone 2002). Alimonti anf Falcone in their paper suggests how MFM, Knowledge Discovery in Databases (KDD) and Fuzzy Logic (FL) can offer an alternative approach to the analysis of producing wells.

Thothill and Zhu(Thornhill and Zhu 2009) illustrate how fuzzy logic together with some specialized knowledge of submersible pumping system performance, can be used to afford near real-time analytics using limited number of signals typically acquired from an ESP system. They came to conclusion than “when incorporated into a SCADA system, fuzzy logic alarms enable the operator to respond to adverse developments in a timely manner, thereby optimizing workover costs and protecting the system run life”.

1.3 MOTIVATION

When it comes to the run life of the pump system and well performance, the surveillance and monitoring are considered as a valuable keys (Bates, Cosad et al. 2004).”Industry experts believe that two-thirds of producing wells on artificial lift could benefit operationally and economically from improved surveillance”(Bates, Cosad et al. 2004) . Obviously by minimizing the likelihood of premature failure of pump systems, caused by inappropriate operating conditions, we can maximize the productivity of the project. (Cohen 1997).

To achieve these goals, there is a need to have a reliable and highly available monitoring system, because in one hand we are dealing with a large amount of data with a high rate coming from sensors and need to be processed, on the other hand without the ability to look at historical information as well as present-time information, the ability to accurately assess performance is significantly vulnerable(Haapanen and Gagner 2010).

To reach this aim and monitor the ESP, in first place we need a process historian system which could gather event-driven data in real-time from multiple sources across the plant and apply analytical calculations and business rules to contextualize and analyze this data. To do so, OSIsoft PI system has been chosen.

Using the calculation engine for PI, an alarm system has been implemented to monitor the pump. Another contribution is implementing the same logic using Microsoft StreamInsight complex event processing engine and adapters for PI which are published in July 1th 2011.

Further contribution is to be benefited from applying fuzzy logic to an alarm condition and make enhancement in the pump surveillance and life time.

1.4 THESIS OVERVIEW

The remaining part of the thesis is organized as follow:

In the second chapter, first some information about ESP and the reasons and parameters which needed to be monitor in order to make the alarms is described, later on, an over view of PI system and its components, fuzzy logic and StreamInsight is provided. In this chapter the reader will be more familiar with terms and logics are used, and get more information about how these technologies work.

The third chapter is a description of implementation phase and how the project has been done with PI and Fuzzy logic in one hand and also using PI adapter for StreamInsight engine, all the challenges and result are also described in this chapter.

Last chapter is the conclusion; the conclusion and works needed to be done further are expressed here. The code and results are provided in Appendix C.

2 TECHNOLOGIES

2.1 ELECTRONIC SUBMERSIBLE PUMP

The choice of proper artificial lifting system is a complicated decision and depends on several characteristics of reservoir such as: pressure, temperature, fluid properties, surface facilities and the type of energy to provide the lift (Bates, Cosad et al. 2004) .

There are several artificial lifting method listed below:

- Road Pump
- Progressing cavity pump
- Gas lift
- Hydraulic lift
- Electronic submersible pump

Among these, Electronic submersible pump is the second-most common method after the Road pump(Bates, Cosad et al. 2004). Electrical submersible pumps have been in used for almost 100 years in the oil and gas industry.(Haapanen and Gagner 2010)

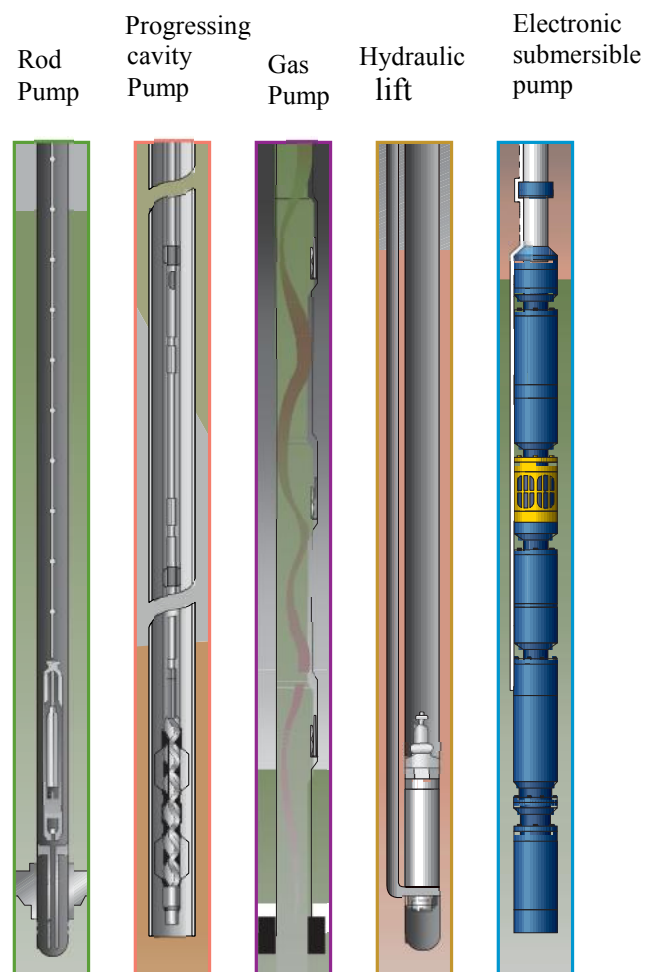


Figure 1. Artificial Lift methods (Bates, Cosad et al. 2004)

“Basic ESP systems include an electric motor, a protector, a gas separator, a multiple stage centrifugal-pump section, a power cable, a motor controller, transformers and a power source. For monitoring of downhole well and pump-operating conditions, downhole instrumentation is also installed. Signals from sensors are transmitted through the power cable to surface remote terminal units (RTUs), allowing continuous sampling of pressures, temperatures, vibration, current and voltage”(Bates, Cosad et al. 2004).

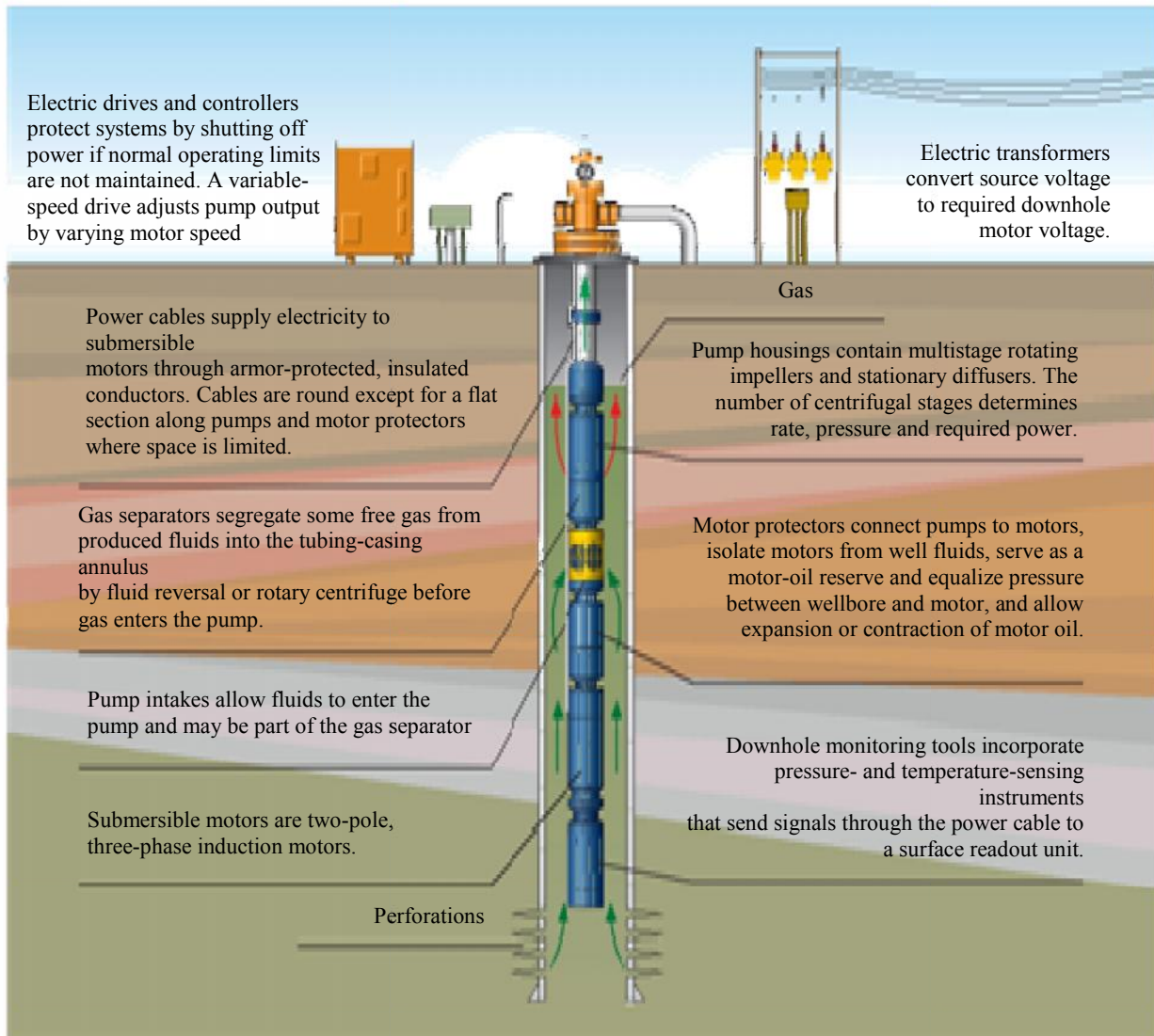


Figure 2. Typical ESP configuration(Bates, Cosad et al. 2004)

2.1.1 SURVEILLANCE AND MONITORING OF ESP

In this part, it is necessary to define terms. The monitoring and the surveillance of any phenomena may be described as the same activity, however there is a difference. “To **Monitor** is to observe or gather data. The **Surveillance** of any phenomena includes monitoring but goes further as a process in setting the boundaries, classifying, analyzing, and interpreting collected data to provide information on the phenomena of interest. Armed with information, one can make decisions to try and realize an objective or achieve a target relative to the phenomena of interest”(Camilleri and Macdonald 2010).

According to (Camilleri and Macdonald 2010) ,the activities which are required in a field with ESPs, could be depicted in a block diagram as is shown in Figure 3. As you can see from diagram, there are two loops called *slow* and *fast* loops, which these feedback loops have steps listed below (Camilleri and Macdonald 2010):

- Monitoring
 - Data management
 - Data Transmission
 - Data Storage
- Surveillance
 - Setting alarms
 - Recording alarms
 - Analysis of alarmed events, also known as diagnostics
 - Recording events
 - Decision making or recommendation
 - Implementation

In the fast loop the time frame is fast and involves monitoring and decision making for start and stop of ESP, changing the settings for ESP and Choke and etc. In slow loop, time frame is identified by the mean time between pulls and could be between few months to 6 years or in some exceptional cases even 10 years. “At each workover, the key decision is whether to rerun an identical ESP or whether to take the opportunity to change the specification with a view to improving the run life and/or production “ (Camilleri and Macdonald 2010) .

However, in the fast loop, by monitoring, we can be ensured of correct operation, but it is not sufficient data to explain the failure. To do such, we need to analyze and review the ESPs production history and inspection supplied by surveillance service in the slow loop. Thus, inspection of the ESP hardware on its own is not sufficient and the ESP history is required to do an inclusive root-cause analysis (Camilleri and Macdonald 2010) .

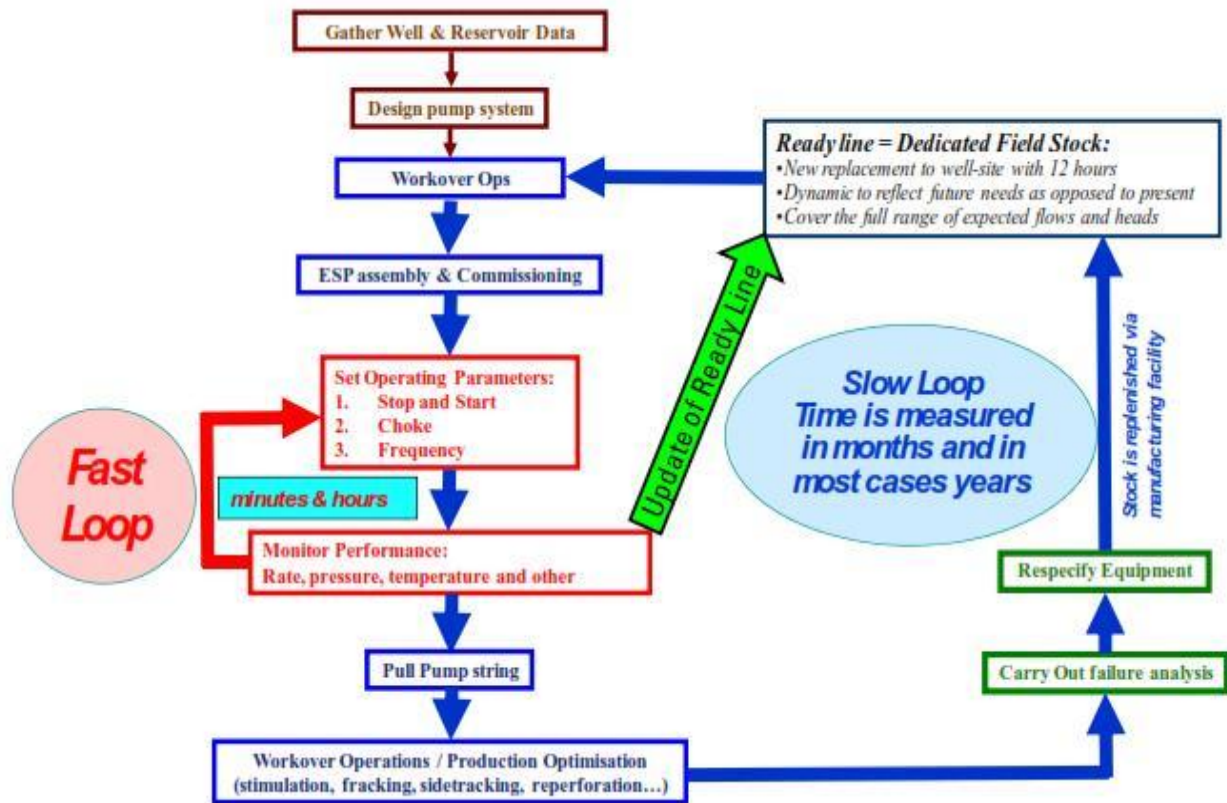


Figure 3. ESP operations block diagram illustrating the slow and fast feedback loops(Camilleri and Macdonald 2010)

2.1.2 THE REASONS AND IMPORTANCE OF MONITORING AND SURVEILLANCE

Generally, the reasons to do the monitoring for systems are, in one hand, to avoid the failures of system and costs related to that, on the other hand to increase the run life of the system. In the case of this project, one of the reasons we do the ESP monitoring is to prevent the stop-start and shut down of the pump and also to prevent the damages accurse because of misusing the pump. In the following section the causes of such failures and the consequences will be shortly described.

2.1.3 START UP AND SHUTDOWN THE ESP

Starting up is one of the most harrowing times in the ESP life. It is because, starting up the ESP will make lots of stress on ESP, caused by the high in-rush current, which as a consequence will impose thermal and mechanical stress, and motor heating(Camilleri and Macdonald 2010).

However, by using some soft starting techniques we can reduce these stresses, but still in-rush current remains considerable. Therefore it is important to reduce the number of starts

and stops to have less stress and also improve the ESP life time(Camilleri and Macdonald 2010).

Also, shut downing or stopping the pump for any reason will make a drop in production rate and will have economical effects on the project too; because the well will not be productive.

2.1.4 TEMPERATURE

One of the parameters that should be under control in a pump is the temperature. As the nature of motor hardware implies, running the motor will have effects in motor temperature; the higher the frequency becomes, the more increase in the temperature will accrue(Camilleri and Macdonald 2010)

To diminish such an increase in the temperature and make it cooler, one of the solutions could be changes in the flow rate. Since the temperature of fluid would be less than motor temperature, so the difference will result in cooling the motor. Therefore, when the flow rate is low, the motor winding temperature will increase exponentially, which could lead to motor burn. Even if the temperature is not too high, it could cause stress on motor insulation, which will have negative influences on ESP life span(Camilleri and Macdonald 2010).

2.1.5 GAS LOCK

The gas which is generated in oil wells can significantly affect pump performance and pump prolonged existence (Bates, Cosad et al. 2004) .

“Free gas passing through the pump stages not only reduce the volumetric efficiency of an ESP pump, but can lead to a gas-locked condition and even system failure (i.e., it can burn the motor or cause extreme vibrations to break the pump shaft).once the free-gas volume exceeds 10% in a given pimp stage, the energy normally used to add head (i.e., lift) is instead spent compressing the free gas back into solution. Once the free-gas volume reaches a certain percentage, the pump will become gas locked”(Noonan, Kendrick et al. 2005).

2.1.6 ALARM CONDITIONS

Since it in not reasonable and economically practical to dedicate one engineer per well, for investigative purposes, so we should use Alarm systems. This will provide an engineer the ability to manage and diagnose several wells at the same time (Camilleri and Macdonald 2010)

When it comes to the monitoring and control of the pump, we will be faced with two different groups of conditions: One group, are the conditions which will lead to shut down of ESP and the other will result in sending a warning signal, and leading to actions such as slowing down the pump (Camilleri and Macdonald 2010).

In this section, the conditions which will be monitored in the project and the corresponding alarm signals and warnings are provided. Below are the abbreviations are used in the formulas:

PIP: ESP intake Pressure
PDP: ESP Discharge Pressure
PDT: Discharge Temperature
AMP: Motor Current
WHP: Well Head Pressure
WHT: Well Head Temperature
FLP: Flow Line Pressure

The different alarms which will be made are listed below:

Gas lock alarm if all three conditions are met:

- 1) $\text{Avg24hPDP} - \text{PDP} > 10\% * \text{Avg24hPDP}$
- 2) $\text{PIP} - \text{Avg24hPIP} > 10\% * \text{Avg24hPIP}$
- 3) $\text{Avg24hMotorCurrent} - \text{MotorCurrent} > 10\% * \text{Avg24hMotorCurrent}$

If both of conditions are met the systems generates an alarm indicating a **Tubing Choke Restrictions:**

- 1) $\text{WHP} - \text{FLP} > 150\% * (\text{Avg24hWHP} - \text{Avg24hFLP})$
- 2) $\text{AMP} - \text{Avg24hAMP} > 10\% * \text{Avg24hAMP}$

An **Unstable or Noflow** conditions alarm if this condition met:

- 1) $\text{WHP} < \text{FLP}$

Analog motor current equals high limit then **Discrete Alarm** is generated :

- 1) $\text{AMP} = \text{Hi limit}$

Analog motor current equals high high limit then **Discrete Alarm** is generated:

- 1) $\text{AMP} = \text{HiHi limit}$

If all these conditions are met then the system generates an **Outage Block** alarm:

- 1) $\text{Avg24hWHP} - \text{WHP} > 10\% * \text{Avg24hWHP}$
- 2) $\text{PDP} - \text{Avg24hPDP} > 10\% * \text{Avg24hPDP}$
- 3) $\text{DP} - \text{Avg24hDP} > 10\% * \text{Avg24hDP}$

Pump Intake Plug alarm if all 3 conditions are met:

- 1) $PIP - Avg24hPIP > 10\% * Avg24hPIP$
- 2) $Avg24hPDP - PDP > 10\% * Avg24hPDP$
- 3) $DP - Avg24hDP > 10\% * Avg24hDP$

If condition 1, 2 and 3 are met then the system generates a **Reduced Flow** alarm:

- 1) $Avg24hPIP - PIP > 10\% * Avg24hPIP$
- 2) $Avg24hPDP - PDP > 10\% * Avg24hPDP$
- 3) $Avg24hWHP - WHP > 10\% * Avg24hWHP$

2.2 PI SYSTEM

The PI System collects, stores, and manages data from plant or process. Data sources, which are the instruments that generate data, are connected to one or more PI Interface Nodes. These interfaces get the data from data sources and send them to the PI Server, where the users can get the data from and display it with some client tools like: ProcessBook, DataLink, or RtWebParts. Each different data source need a PI interface which could interpret it.(OSIsoft 2006)

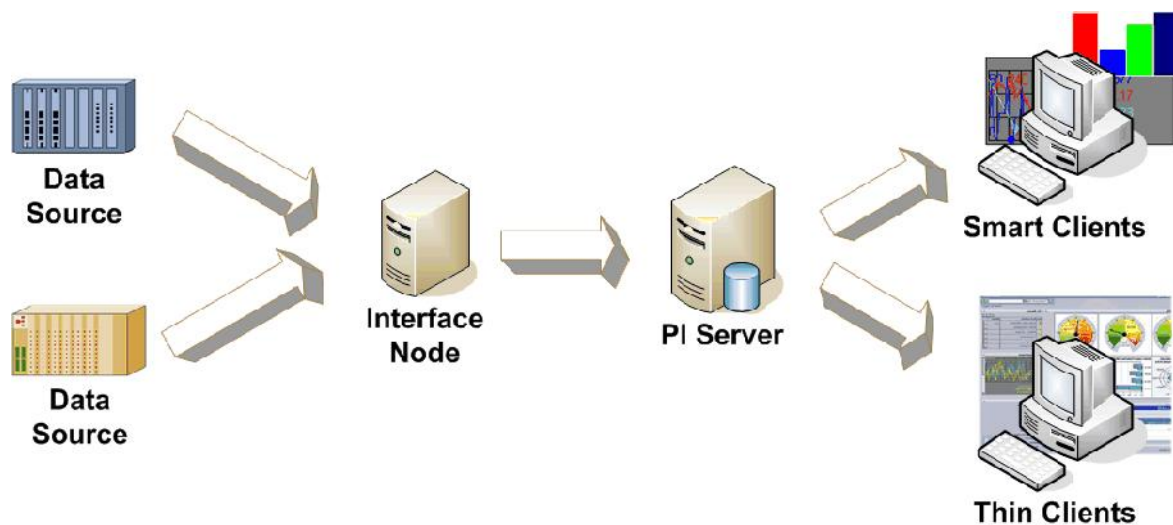


Figure 4. PI System data flow (OSIsoft 2006)

2.2.1 PI INTERFACE NODE

For each data sources, OSIsoft has afforded a specialize interface. Usually these interfaces run on a granted system, called an Interface Node. The interface gathers data from data source. The data sources can be almost anything, including Distributed Control Systems (DCSs), Programmable Logic Controllers (PLCs), lab systems, Supervisory Control and Data Acquisition systems (SCADA), process models, and other business information systems. PI Performance Equations, ACE, and Totalizer are also all data sources(OSIsoft 2006).

“Interface Nodes can run multiple interfaces to multiple PI Servers. The Interface Node might be a machine that is a part of the foreign data system, or a stand-alone dedicated interface machine, or even a PI Server itself”(OSIsoft 2006)

2.2.1.1 Data Flow on the Interface Nodes

The events that are stored in the PI sever, are consist of: a value and a time stamp which tells the time that value was collected. Interfaces do not pass all of events directly to the Server; only the noteworthy data will be passed and the rest will be discarded. This is done by exception reporting in Interface Node(OSIsoft 2006).

Interface Nodes are also benefited by the buffering service. If it is conFIGured on the node, in the case of losing the connection with server, the data will be buffered and stored until the connection to server is repaired(OSIsoft 2006).

2.2.1.2 Exception Reporting

The reasons for doing exception reporting are sending the data that is interesting for the system, and reduce the unnecessary usage of network connection.

Data will be sent to server in the certain conditions. If the value is outside the boundary or range which is defined by ExcDev, that value and the previous value will be passed to server; so considering that role, in the case which is depicted in Figure 5. D and C will be the data that server will receive.

However, when the ExcMax time reaches, if there is no value outside the range, the last value will be sent in the case that such value exists. (OSIsoft 2006)

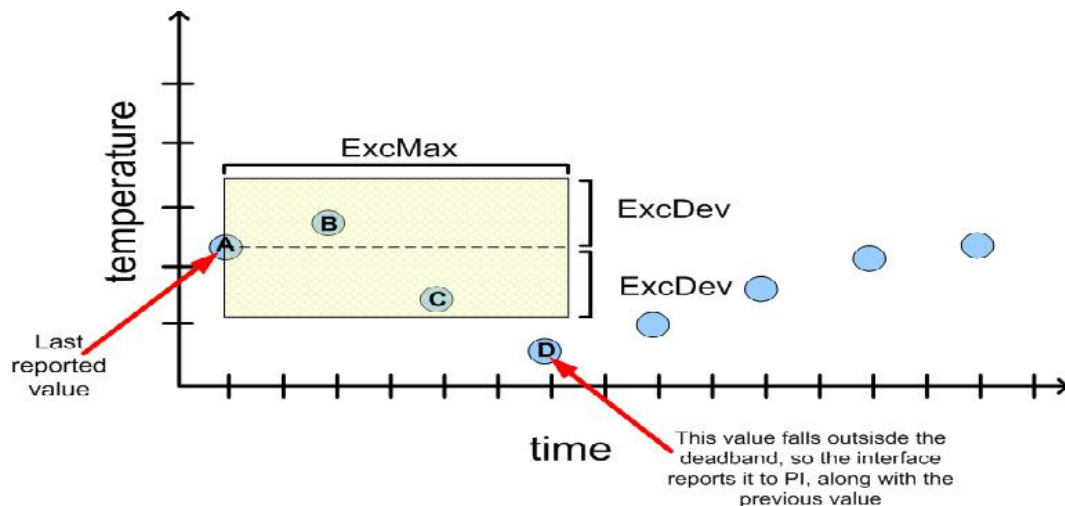


Figure 5. Exception reporting(OSIsoft 2006)

2.2.2 PI SERVER

PI server is considered as the heart of PI system. It acquires the data and directs it all over the PI system and entire information infrastructure, in real-time. Operators, engineers, managers, and other plant personnel can connect to the PI Server and view manufacturing data from PI Data Storage or from external data storage systems(OSIsoft 2006).

2.2.2.1 Data Flow in the PI Server

When a new event is received in PI Server from interface or manual input, server will send it to the Snapshot Subsystem. In the snapshot subsystem a single value for each PI point will be held in memory. Later, when a new value enters, the server will send the old value to archive subsystem, where a compression test will be performed to decide either discards it or sends it to Event Queue based on the result of test (OSIsoft 2006).

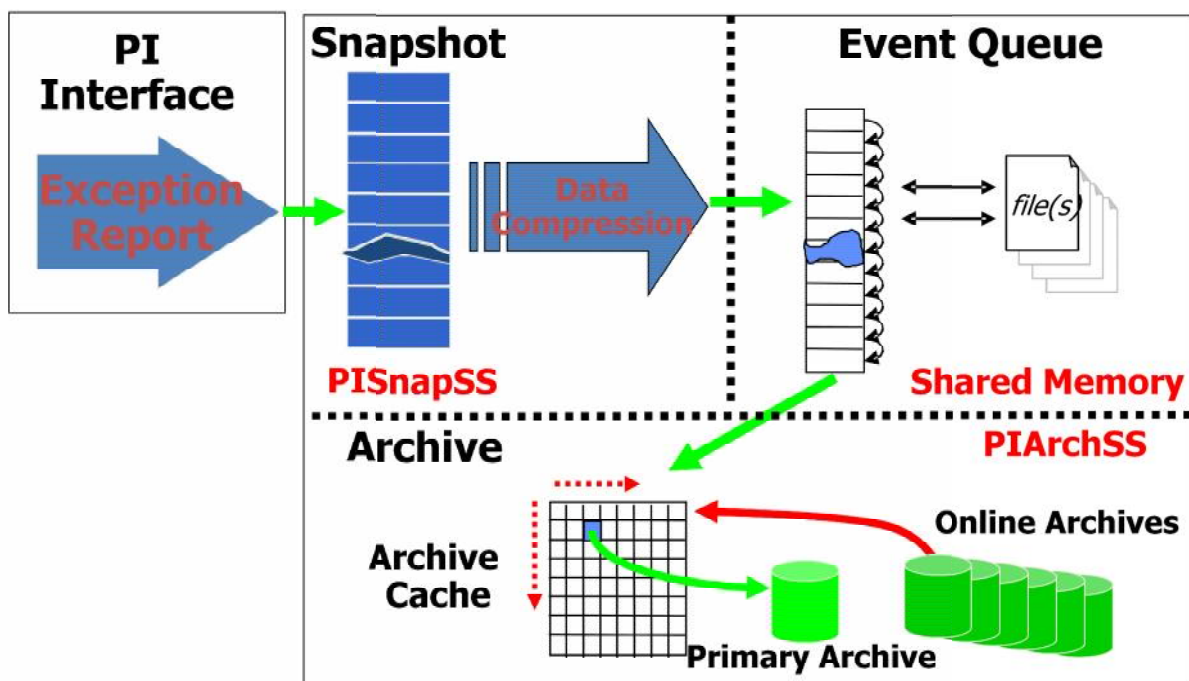


Figure 6. Data flow in PI(OSIsoft 2006)

2.2.2.2 Snapshot Subsystem

“The Snapshot Subsystem gets the new data from the Interface Node and holds the most recent value for each point. This most recent value is called the Snapshot for that point” (OSIsoft 2006). If an event that comes in to the Snapshot Subsystem has a timestamp that is older than the current Snapshot value, the PI Server will send it directly to the Event Queue for archiving, without compression testing. This kind of events are called out of order events (OSIsoft 2006).

2.2.2.3 Compression Testing

“The point of compression testing is to store just enough data to accurately reproduce the original signal”(OSIsoft 2006). By using the compression test, only values which worth to store will be archived, instead of archiving all the values. This will lead to higher speed in retrieving the data.

PI uses a compression method called **swinging door compression**. “Swinging door compression discards values that fall on a line connecting values that are recorded in the Archive. When a new value is received by the Snapshot Subsystem, the previous value is recorded only if any of the values since the last recorded value do not fall within the compression deviation blanket. The deviation blanket is a parallelogram extending between the last recorded value and the new value with a width equal to twice the compression deviation specification”(OSIsoft 2006).

The first value received by snapshot will be stored in archive, later, every time that snapshot receives a new value a parallelogram will be created between the last archived value and the current value. The width of parallelogram is equal to twice the compression deviation specification. When creating the parallelogram, if there is a value falling outside it, the previous value will be archived and the process will continue this time with a new archived value (OSIsoft 2006).

2.2.2.4 Event Queue

The PI Event Queue as is between the Snapshot and Archive Subsystem and acts as a memory buffer. The data from snapshot will be added to the Queue and will be removed from queue by archive subsystem(OSIsoft 2006).

Usually the data that come to queue will be passed quickly to archive, but in some cases data could be hold in queue. This could be happen for example when the archives are unavailable because archive shift or archive backups are occurring, or when the archive subsystem is busy for the reason that the incoming events are out of order(OSIsoft 2006)

2.2.2.5 Archives

PI archive is the place that PI stores the data. As the data mounts up, new archives are needed to hold data. “Typically, archives are files of a fixed size that can hold PI data. Fixed archives allocate the full amount of space upfront, meaning that an empty archive and a full archive take the same amount of disk space”(OSIsoft 2006).

The archive which receives current data is called the **Primary Archive**. When the Primary Archive becomes full, an **Archive Shift** will be occurred and the next available archive will become the new Primary Archive. “For an archive file to be eligible to be the new Primary Archive, it must be registered, writeable, shiftable, and large enough to handle the current size of the Point Database”(OSIsoft 2006). In the case of not having an eligible archive

available, PI will use the oldest available filled archive as the new Primary Archive and overwrite the data(OSIsoft 2006).

Also, during the time which archive shifting is accruing, it is not allowed to add, edit, or delete points. Incoming data will be stored in Event Queue until the shift is done and then PI will write the data to new primary archive(OSIsoft 2006).

2.2.3 PI POINTS AND PI TAGS

“Points, sometimes also called tags are the basic building blocks of a PI system, because they are how you track the events that comprise your data history”(OSIsoft 2006).It is a unique storage place in the PI System for a specific stream of data like a flow rate from a flow meter or the pressure of a well and etc.

Although the terms point and tag are often used interchangeably, there is a difference; a tag is a label or name for a point. A point is any measurement or calculation that is stored in the data archive(OSIsoft 2006).

Below are the data types PI can store:

Digital:	Discrete value (On/Off, Red/Black/Green)
Int16:	Integer value, 16 bits (0 to 32767, acc: 1/32767)
Int32:	Integer value, 32 bits (-2147450880 to 2147483647)
Float16:	Scaled Floating Point number, 16 bits (1/32767 times range)
Float32:	Floating Point number, 32 bits (single precision)
Float64:	Floating Point number, 64 bits (double precision)
String:	Text value up to 976 characters
Blob:	Binary large object up to 976 bytes

2.2.4 POINT ATTRIBUTES

By using the point attributes we can identify how and when the data should be collected for that point. “Point attributes specify the data source location, how often PI should get new values from the data source, which values PI can ignore and which represent valid data, and much more”(OSIsoft 2006).

Every PI tag belongs to a point class. A point class identifies how many attribute that particular class supports. All PI tags support base class. Figure 7 shows the point classes.

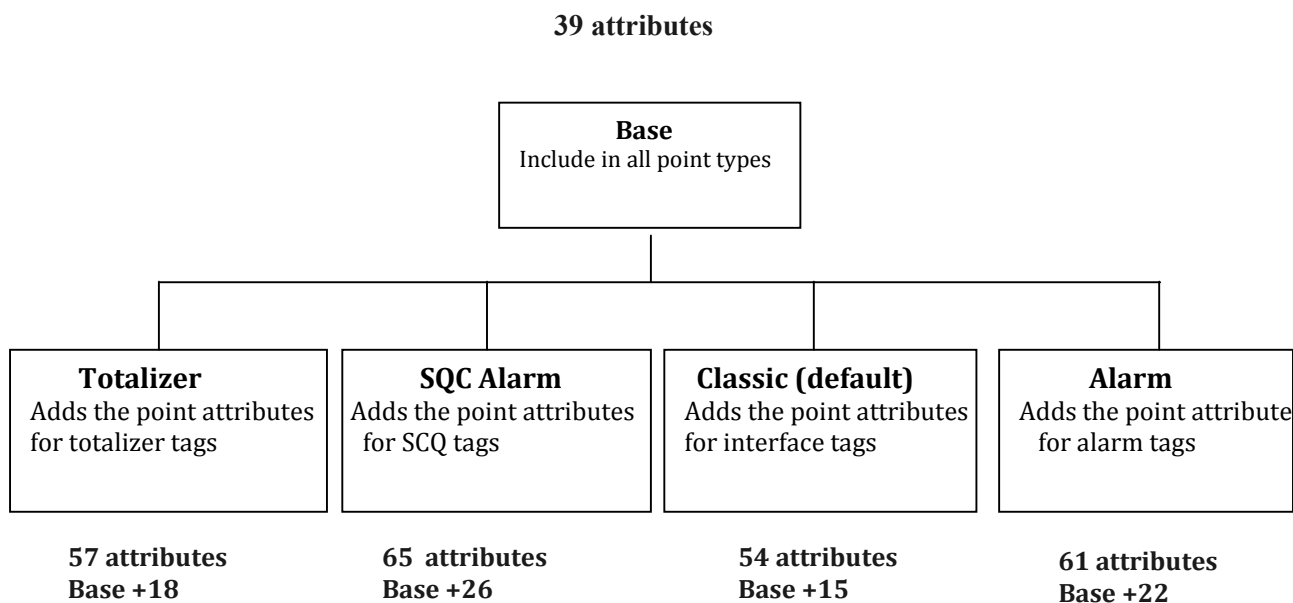


Figure 7. Attribute classes

The list of point attributes and a short description is provided in Appendix A.

2.2.5 TIME IN PI

PI stores timestamps as the number of seconds expired since Jan 1st of 1970 GMT. For specifying time, PI support two categories: Absolute and Relative or a combination of these two(OSIsoft 2006).

Absolute time would reference to a specific time not necessarily relevant to the current time. The format is **dd-mmm-yy HH:mm:ss**

Date field is default to current date and time field default to 00. If date field be omitted, the corresponding current date field will be assumed and in the case of time 0 will be assigned. For example: 14 9 means 09:00:00 on the 14th of current month.

There are also symbols can be used:

Symbol	Meaning
*	Current time
T	00:00:00 on the current day (TODAY)
Y	00:00:00 on the previous day (YESTERDAY)
Monday, Tuesday, Wednesday, Thursday, Friday, Saturday. Sunday	00:00:00 on the most recent of that day of the week

In Relative time, time is offset from another time and is relative to other times like start time or current time.

There is no default time unit in relative time and you should specify the unit which could: be d, h, m, s, w, mo, and y respectively for days, hours, minutes, seconds, weeks, months and years. And you can use fractions just for hours, minute and seconds for example +3.5 h or -2m(OSIsoft 2006).

2.2.6 PI PROCESS BOOK

“PI ProcessBook is a PC application for displaying plant information stored in the PI Data Archive or in relational databases”(OSIsoft 2009).

Each ProcessBook is a collection of display entries. The display entries show process data from one or more PI Systems as well as other static and dynamic information from outside sources. Most displays contain a number of tags from one or more PI Systems. By connection to a PI Server, users will be able to view process data at the current time or at other, discrete points in time. Displays update dynamically whenever values on the PI Server change(OSIsoft 2009).

Process Books can be shared among users, this removes the need to build duplicate displays, but on the other hand, only one user at a time can open individual display files. On networks, unrestricted number of users may access the same ProcessBook at the same time. Additionally, it is possible to have multiple sessions of the application concurrently active on a computer (OSIsoft 2009).

2.2.7 PI PERFORMANCE EQUATION

Performance Equation is an expression that allows a user to implement an arbitrary and potentially sophisticated calculation without formal programming. A performance equation has an intuitive syntax and may consist of standard mathematical and logical operators as well as a wide variety of built-in functions. The result of a performance equation can be archived for a PE point just like data for any other point. Performance equations are also available programmatically via the SDK for archive calculations and other data filtering operations(OSIsoft 2006).

2.2.8 PI ADVANCED COMPUTING ENGINE

“The PI Advanced Computing Engine is a computation tool that can be used to create many calculations. PI ACE is designed to develop standalone VB applications that make use of PI data. It has been used to perform flow compensation, control loop performance and tuning, downtime analysis, tanks movements and Inventories, alarming (including sending emails &

pages), complex batch triggering, read from and write to flat files and data filtering”(OSIsoft 2006).

Since, the development environment for ACE is Visual Basic and because of the major differences between Visual Basic 6 and Visual Basic.NET, PI ACE is available in two versions: ACE 1.x (based on Visual Basic 6) and ACE 2.x (based on Visual Basic.Net).

ACE consists of three components:

- ACE Wizard: A visual basic Add-In which helps users to build and test ACE Modules.
- ACE Manager: Allows users to monitor and change different properties of ACE Modules.
- ACE Scheduler: Executes ACE Modules in a timely manner and handles updates and unusual behavior.

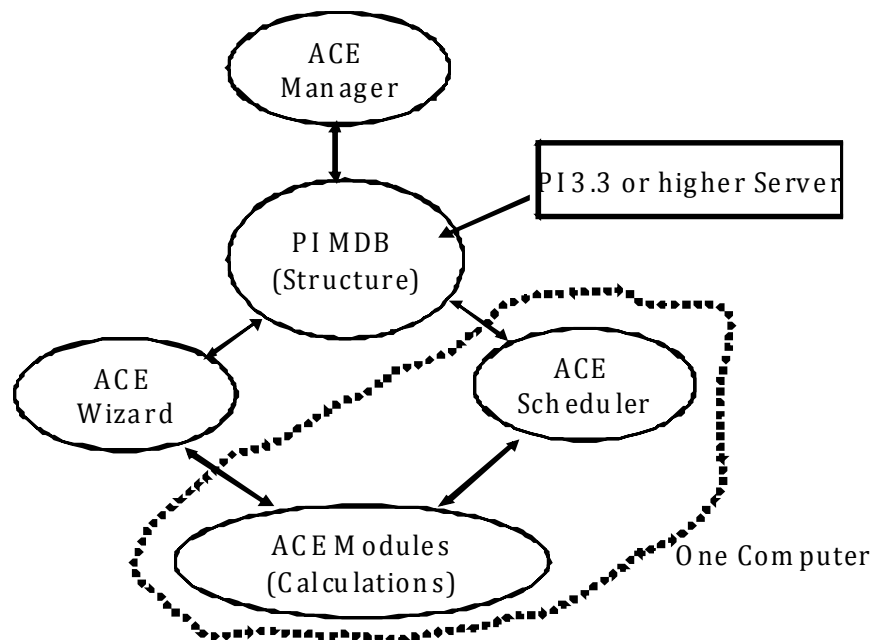


Figure 8. ACE components and their common data(OSIsoft 2006)

“The structural information for ACE Modules is stored in the PI Module Database. While the ACE Scheduler and the .exe (developed with ACE 1.x) or .dll (developed with ACE 2.x) files for all ACE Modules need to be on the same computer, the ACE Wizard, the ACE Manager, the ACE Scheduler, and the PI Server that stores the ACE structural information may be on different computers.”(OSIsoft 2006)

2.2.8.1 ACE Module and ACE Executables

Each ACE Module developed with ACE 1.x or 2.x match up to a class in a Visual Basic 6 ActiveX Exe project and in a Visual Basic.NET Class Library project, correspondingly. The project is referred to as ACE Executable.

Each ACE Executable can include multiple ACE Modules. Each ACE Module can be run in different contexts (*i.e.*, an instance of ACE Module). For example, if the same calculations need to be carried out for well #1 and well #2, then only one set of equations needs to be developed and maintained but it may be run in two different contexts. This would significantly facilitate the development and maintenance of calculations for similar units or processes.(OSIsoft 2006)

One advantage of putting multiple ACE Modules in one ACE Executable is that only one set of .dll's needs to be loaded which will result in reduction the overall memory usage(OSIsoft 2006).

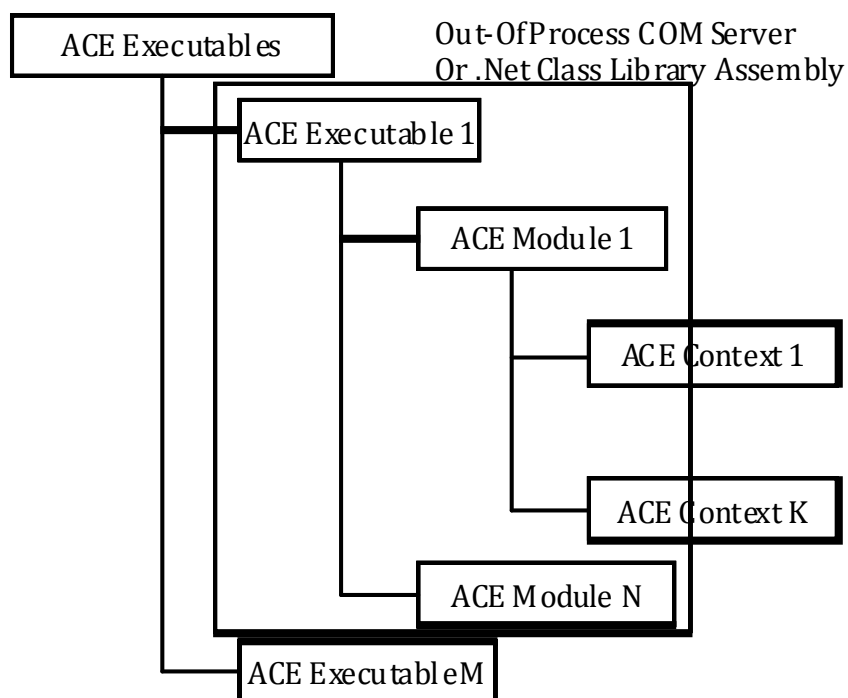


Figure 9. Structure of ACE Modules(OSIsoft 2006)

2.2.8.2 PI Module Database

“The PI Module Database is a hierarchical database hosted by the PI Base subsystem. It can be used as a scalable and secure database to store plethora of information related to the PI System. The PI Module Database can provide structure at the server level for organizing plant information by grouping tags together based on the asset that they are retrieve data from”(OSIsoft 2006).

PI Modules can be built in the PI Module Database in any hierarchical structure. They can be built based on organization, geography, product line, or any criteria. The best hierarchy In the case of PI ACE is the one that could be easily explored by the user who has created the calculation(OSIsoft 2006).

2.2.8.3 PI Aliases

“Aliases are common names for tags. This is especially useful when there are many units that have the same functions (aliases)”(OSIsoft 2006). All PI Aliases will contain:

- A name.
- A link to a PI tag for a given PI server.

2.2.8.4 PI Properties

“Properties are aspects that do not change often. The PI Properties are stored in a similar manner to the PI Aliases”(OSIsoft 2006). Each PI Property is an object that contains:

- A name.
- A value.
- A type.

When creating PI Properties, the data should change occasionally (no more than once a week).

2.2.8.5 Scheduling Types

The type of scheduling defines when a calculation should be carried out. ACE has two types of scheduling: clock and natural. The scheduling information applies to an ACE Context.

“With **clock scheduling**, an ACE Context is evaluated at fixed intervals. Two attributes, period and offset, determine when to evaluate the ACE Context. The period specifies the interval between calculations and the offset specifies the time since the midnight to start the calculation. While the period can have a fraction of a second, the offset should be an integer between 0 and 86399 seconds”(OSIsoft 2006).

“**Natural scheduling** means that an ACE Context is evaluated whenever one of its trigger PI tags has received a new snapshot event. The trigger tags must be a subset of input tags/aliases used in the ACE Module. Tags/aliases used as both input and output cannot be used as trigger tags as this may easily result in an infinite loop if not handled properly”(OSIsoft 2006).

2.2.9 ADVANTAGES OF ACE OVER PERFORMANCE EQUATION

Performance equation is a tool that we can use for doing the calculations but ACE has some features that cover the limitations of Performance equation. For example in Performance equation you are just restricted to use the equation syntax and if then else statements, but in ACE since you are doing your calculation with high level programming language you can be benefited of its advantages. As a simple example you can use the switch cases, loops, etc. Also it provides the ability to use external libraries. Another limitation for Performance Equation is that when testing the equation with the **pipetest** utility it is allowed to use up to 4095 characters , but in ACE there in no such limitation(OSIsoft 2006; OSIsoft 2006)

Other features of ACE can be listed as below (OSIsoft 2006):

- To implement complex calculations (e.g., iterative solutions, data and time manipulation, and numerically solving ordinary or partial differential equations).
- To provide a fault tolerant/redundant architecture.
- To retrieve data from and send results to multiple PI tags or other systems.
- To apply one set of equations to multiple units or processes.
- To provide various scheduling features: clock, natural, event, equation ordering, and graceful degradation under resource limitations (CPU loading).
- To use multiple PI tags from multiple PI Servers in calculations.
- To provide the ability to use the previous snapshot value for a PI tag (instead of the previous archive value) in calculations.
- To allow documentation/comments with a set of equations.
- To allow clamping and bad value substitution of inputs and outputs.
- To provide the ability to call COM and .NET objects and a library of user-written functions.
- To provide the ability to test and debug equations.
- To provide the ability to have different users responsible for different groups of equations.
- To provide the ability to stop and restart any of the calculations in production, set priorities, manage the data fed to ACE, even assign ACE to other data, – without recoding.
- To provide the ability to transfer PI ACE calculations from one PI Server to another.
- To provide the ability to monitor performance of individual equations.
- To provide robustness, including trapping floating point errors and avoiding repetitive message logs.
- To expose calculations via a web service.
- To allow automatic recalculation when “past” data are changed.
- To manually recalculate an ACE Context.

2.3 FUZZY LOGIC

In 1965 Zadeh, a professor at the University of California at Berkley conceived the foundations of infinite-valued logic with his mathematics of fuzzy set theory (Engelbrecht 2008). “Fuzzy Logic is a problem-solving control system methodology that lends itself to implementation in systems ranging from simple, small, embedded micro-controllers to large, networked, multi-channel PC or workstation-based data acquisition and control systems. It can be implemented in hardware, software, or a combination of both”(Kaehler 2003)

Dissimilar to Boolean logic, which is a two-valued logic and everything is translated into the absolute terms of 0 and 1, Fuzzy logic is an infinite-valued approach to compute based on degrees of truth and uncertain information

Fuzzy Logic is a method for modeling and reasoning about vague or estimated concepts. It allows the user to represent or model the problem linguistically. For example, in concept of temperature, by using Boolean logic, we can only say if the temperature is high or not by comparing its value with a number; but fuzzy logic provides us the ability to reason about if “*Temperature is normal*” or “*Temperature is high*” or “*Temperature is too high*” with using the exact natural linguistic variables.

2.3.1 FUZZY SET AND MEMBERSHIP FUNCTION

The difference between two-valued sets and fuzzy logic sets is that, in fuzzy logic set, each element has a membership degree which defines the certainty that the element belong to that set(Engelbrecht 2008).

Assume X is the domain of discourse and x a specific element of domain X , so for fuzzy set A we can define a membership mapping function such that :

$$\mu_A: X \rightarrow [0, 1]$$

where $\mu_A(x)$ defines the certainty that the element belong to that set(Engelbrecht 2008).

Membership function is the soul of fuzzy set. “A membership function, also referred to as the characteristic function of the fuzzy set, defines the fuzzy set. The function is used to associate a degree of membership of each of the elements of domain to the corresponding fuzzy set”(Engelbrecht 2008).

The membership function, which is determined by experts in the domain, could have different shape and type but when designing, it should be granted that the function must be bounded from below by 0 and from above by 1 (i.e. the range should be $[0, 1]$) and for each element $x \in X$ in the domain $\mu_A(x)$ must be unique(Engelbrecht 2008).

2.3.2 FUZZY OPERATORS

Relations and operators are defined for fuzzy sets similar to crisp sets. Below is a short description of each operator(Engelbrecht 2008):

Equality: Two fuzzy sets are equal if they have exactly the same elements and the degrees of membership of elements to the sets are also equal.

Containment: Fuzzy set A is a subset of fuzzy set B if all the elements of A are also elements of B and if $\mu_A(x) \leq \mu_B(x)$ for all $x \in X$.

Complement (NOT): For fuzzy sets, the complement of the set A consists of all the elements of set A , but the membership degrees differ. Let \bar{A} represent the complement of set A . Then, for all $x \in X$, $\mu_{\bar{A}}(x) = 1 - \mu_A(x)$. It also follows that $A \cap \bar{A} \neq \emptyset$ and $A \cap \bar{A} \neq X$.

Interaction (AND): “The intersection of two-valued sets is the set of elements occurring in both sets. Operators that implement intersection are referred to as t-norms. The result of a t-norm is a set that contain all the elements of the two fuzzy sets, but with degree of membership that depends on the specific t-norm”(Engelbrecht 2008). Below are two popular t-norms:

$$\text{Min-operator: } \mu_{A \cap B}(x) = \min\{\mu_A(x), \mu_B(x)\}, \quad \forall x \in X$$

$$\text{Product operator: } \mu_{A \cap B}(x) = \mu_A(x) \mu_B(x), \quad \forall x \in X$$

Union: The union of fuzzy sets includes the elements of all of the sets, but with membership degrees that depend on the specific Union referred to as s-norms. Followings are most commonly used s-norms:

$$\text{Max-operator: } \mu_{A \cup B}(x) = \max\{\mu_A(x), \mu_B(x)\}, \quad \forall x \in X$$

$$\text{Summation operator: } \mu_{A \cup B}(x) = \mu_A(x) + \mu_B(x) - \mu_A(x)\mu_B(x), \quad \forall x \in X$$

2.3.3 LINGUISTIC VARIABLES AND FUZZY RULES

In 1973, Lotfi Zade presented linguistic variables which enable computing with words or sentences from natural language, rather than numbers. Linguistic variables are essential for fuzzy logic strategies. They contain values that are uniformly distributed between 0 and 1, depending on the significance of a context-dependent linguistic term (Banks and Hayward 2001; Engelbrecht 2008).

In natural language we combine nouns with adjectives to quantify nouns, for example in context of weather we can use adjectives like *very* for quantification, and say weather is “*very cold*”. In fuzzy system theory these adjectives are called **hedges**. A hedge is used as a modifier of fuzzy values and changes the membership of elements(Engelbrecht 2008).

By using a set of linguistic rules, we can portray the dynamic behavior of a fuzzy system. Generally, fuzzy rules are in the form of:

If antecedent(s) then consequent(s)

Where antecedent and consequent should consist of linguistic variable; for example one of the common forms of rules is

If A is a and B is b then C is c

where A, B and c are fuzzy sets(Engelbrecht 2008).

The collaboration of the fuzzy sets and fuzzy rules will form the knowledge base of a fuzzy rule-based reasoning system.

2.3.4 FUZZIFICATION

The input space and output space of fuzzy are generated by antecedent and consequent respectively. The input space is defined by the combination of input fuzzy sets, while the output space is defined by combination of output sets. “The **fuzzification** process is concerned with finding a fuzzy representation of non-fuzzy input values. This is achieved through application of the membership functions associated with each fuzzy set in the rule input space. That is, input values from the universe of discourse are assigned membership values to fuzzy sets”(Engelbrecht 2008).

As an example, consider a fuzzy set A with membership function μ_A and universe of discourse X. when receiving a $a \in X$, fuzzification process will produce $\mu_A(a)$.

2.3.5 INFERENCE

The inferencing process is for mapping the fuzzified input to the rule base, and to produce a fuzzified output for each rule. “That is, for the consequents in the rule output space, a degree of membership to the output sets is determined based on the degrees of membership in the input sets and the relationships between the input sets. The relationships between input sets are defined by the logic operators that combine the sets in the antecedent. The output fuzzy sets in the consequent are then combined to form one overall membership function for the output of the rule”(Engelbrecht 2008).

As an example consider input fuzzy sets A and B with universe of discourse X1 and the output fuzzy set C with X2 as universe of discourse and assume the rule:

If A is a and B is b then C is c

After the fuzzification process is finished, the inference engine receives $\mu_A(a)$ and $\mu_B(b)$. Then in the first step, the inferencing process calculates the firing strength of each rule in the rule base. Thus, for each rule k, the firing strength α_k is computed(Engelbrecht 2008).

Later all activated outcomes will be accumulated. During this step, one single fuzzy value is determined for each $c_i \in C$ (usually max-operator is used) (Engelbrecht 2008).

Finally, the end result of the inferencing process is a series of fuzzified output values. Rules that are not activated have a zero firing strength(Engelbrecht 2008).

2.3.6 DEFUZZIFICATION

Defuzzification produces a crisp value from output of the inferencing process There are several defuzzifiers proposed literature, but when we are interested in engineering application of fuzzy logic one of the criterions to opt a difuzzifier is its simplicity(Mendel 1995).

Several inference methods exist to find an approximate scalar value to represent the action to be taken. Below is a description of some popular functions from(Engelbrecht 2008):

- The **max-min method**: The rule with the largest firing strength is selected, and it is determined which consequent membership function is activated. The centroid of the area under that function is calculated and the horizontal coordinate of that centroid is taken as the output of the controller. For our example, the largest firing strength is 0.8, which corresponds to the *large increase* membership function.
- The **averaging method**: For this approach, the average rule firing strength is calculated, and each membership function is clipped at the average. The centroid of the composite area is calculated and its horizontal coordinate is used as output of the controller. All rules therefore play a role in determining the action of the controller.
- The **root-sum-square method**: Each membership function is scaled such that the peak of the function is equal to the maximum firing strength that corresponds to that function. The centroid of the composite area under the scaled functions is computed and its horizontal coordinate is taken as output.
- The **clipped center of gravity method**: For this approach, each membership function is clipped at the corresponding rule firing strengths. The centroid of the composite area is calculated and the horizontal coordinate is used as the output of the controller.

2.3.7 FUZZY CONTROLLER

Fuzzy control uses the principles of fuzzy logic based decision making to achieve the control tasks(Karakose and Akin 2010). “Where fuzzy logic is frequently described as computing with words rather than numbers, fuzzy control is described as control with sentences rather than equations”(Engelbrecht 2008).

A fuzzy controller can be considered as a nonlinear static function that maps controller inputs onto controller outputs. A controller controls some systems or plants by taking corrective action and response, according to the any set of inputs that are provided. A fuzzy controller consists of four main components, which are essential to the operation of the controller(Engelbrecht 2008):

Fuzzy rule base: The rule base or knowledge base, includes the fuzzy rules that show the knowledge and experience of a human expert of the system.

Condition interface (fuzzifier): The fuzzifier gets the actual outputs of the system, and turns these non-fuzzy values into membership degrees to the corresponding fuzzy sets.

Action interface (defuzzifier): The action interface defuzzifies the result of the inference engine to generate a non-fuzzy value to represent the real control function to be applied to the system.

Inference engine: The inference engine executes inferencing on fuzzified inputs to generate a fuzzy output.

2.4 COMPLEX EVENT PROCESSING AND STREAM INSIGHT

Complex event processing (CEP) is a set of techniques and tools helping us to control and be aware of event-driven information systems(Luckham 2001). “In the mid-1990s when the database community realized that databases were too slow to do real-time data analysis, they started researching the idea of running continuous queries on streams of incoming data. They used sliding time windows to speed up the queries. An answer to a query would be valid only over the events in the current time window, but as the window slid forward with time so also the answer was updated to include the new events and exclude the old ones. This research was called Data Streams Management (DSM) and led to the event streams processing world of today. The emphasis was on processing the data in lots of events in real-time”(Luckham 2006)

In April 2009, Microsoft published first version of StreamInsight which is a platform for developing and deploying complex event processing (CEP) applications. Its stream processing architecture and .NET-based development experience enable developers to implement event processing applications. StreamInsight analyzes and associates data incrementally and in-memory while the data is in flight.(Torsten Grabs, Roman Schindlauer et al. 2009)

The StreamInsight server is made up of the core engine and the adapter framework. Incoming events are incessantly streamed into standing queries in the CEP server, which processes and transforms the data according to the logic defined in each query. The query result at the output can then be used to trigger specific actions(Torsten Grabs, Roman Schindlauer et al. 2009).Figure 10 illustrates StreamInsight architectural overview.

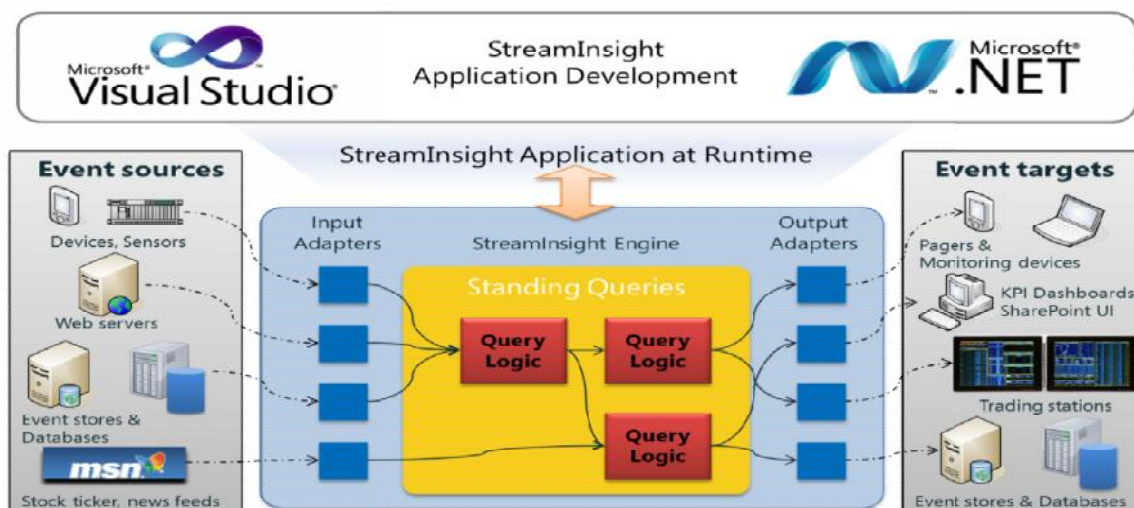


Figure 10. StreamInsight architectural overview(Torsten Grabs, Roman Schindlauer et al. 2009)

2.4.1 EVENTS

All data in StreamInsight are structured as streams and the fundamental data represented in the stream is packaged into events. Each event consists of two parts (Microsoft 2011):

- **Payload:** A .NET data structure that holds the data associated with the event. The fields defined in the payload are user-defined. Their types are based on the .NET type system.
- **Header:** An event header contains metadata that defines the event kind and one or more timestamps that define the time interval for the event. The timestamps are application-based and supplied by the data source rather than a system time supplied by the StreamInsight server.

Based on the temporal characteristics of events, StreamInsight has three event shapes(OSISoft 2011):

- **Point:** “represents an event at a single point in time. Point events have a single timestamp representing the start time of the event but no timestamp for the end time of the event”.
- **Interval:** “represents events that span a time period. These events have a start and an end time that represent a time span during which the event payload is considered to be valid”.
- **Edge:** “an alternative to interval events for use if the end time of the event is not known at the time the event is started. Initially, the end time is set to a value in the far distant future, and this is modified to the actual end time when it becomes known”.

2.4.2 ADAPTERS

Adapters interpret events coming to StreamInsight serve and send outgoing event streams from StreamInsight serve. “Adapters are implemented in the C# programming language and stored as assemblies. The adapter classes are created as templates during design time, registered in the StreamInsight server, and instantiated in the server during run time as adapter instances”(Microsoft 2011).

Input and output adapters are characterized into two groups(OSISoft 2011):

- **Typed Adapters:** “Typed adapters are those for which payload type is determined when the adapter code is compiled. This generally means that the number of fields, their names, and their types are fixed. This knowledge allows the adapters to build and populate StreamInsight events more efficiently so the throughput of a typed adapter is usually better than that of an untyped adapter using the same payload”.
- **Untyped Adapters:** “Untyped adapters support payload types that are not known at the time the adapters are compiled, so an application developer using an untyped adapter has more freedom to design a payload type. StreamInsight must do extra work to analyze the payload structure and work out how to populate the individual fields in

the payload so there is a performance penalty every time a new event created and submitted to the StreamInsight engine”.

2.4.3 QUERY

To define the business logic which needs to continuously analyze and process events submitted to the StreamInsight server from the input adapter and correspondingly generate an event stream that is consumed by the output adapter a query template is used. Query templates are written in LINQ combined with the C# language(Microsoft 2011).

LINQ is an acronym for the Language Integrated Query which was released first in 2007 as a part of .NET framework3.5. It makes it possible to query data in .NET supported languages. .NET Language-Integrated Query defines a set of general purpose standard query operators that allow traversal, filter, and projection operations to be expressed in a direct yet declarative way in any .NET-based programming language(Microsoft 2007) .

3 IMPLEMENTATION

3.1 ALARM SYSTEM USING ACE

In this section the implementation details of alarm system will be described. The screen dumps are provided to show how the system works and how it is developed. Also PI random interface has been used during the development to get the simulated data and make the testing more productive and fast.

Since access to components and executing the modules require PI server and PI ACE to be installed and running, and due to the industrial license of all OSIsoft PI products, it will not be possible to execute modules on the other machine in the absence of any of those components.

The implementation details and program outputs are provided in appendix C.

3.1.1 ENVIRONMENT AND SYSTEM PREREQUISITES

The development environment for ACE is the Visual Basic programming software, the latest release of ACE (2.1.50) that is used in the project, supports Visual Basic 6, Visual Basic.NET 2005, 2008, 2010. Among those, Visual Basic.NET 2005 has been selected. The three components of ACE (i.e. ACE Manager, ACE 2.x Wizard, and ACE 2.x scheduler) have been installed on same the computer. Prerequisites for these three components are:

- Microsoft .NET Framework 2.0 or above
- PI SDK 1.3.8.387 or higher for 64-bit and PI SDK 1.3.8.388 or higher for 32-bit
- PI API 1.6.2.4 or higher

In order to develop and run ACE Modules, a PI Server version 3.3 or higher is required to store the structural information of ACE Modules in the PI Module Database. This server is referred to as the PI ACE Data Server. The actual PI tags used in calculations or for archiving results in an ACE Module, come from PI Server. Both PI server and PI ACE components have been installed in the same machine.

3.1.2 DEVELOPMENT OF ACE MODULES

The first step in developing the modules was to create the tags; each would represent one of the parameters described in section 2.1.6. Later, a module in Module database was created containing the application parameters shown in Figure 11.

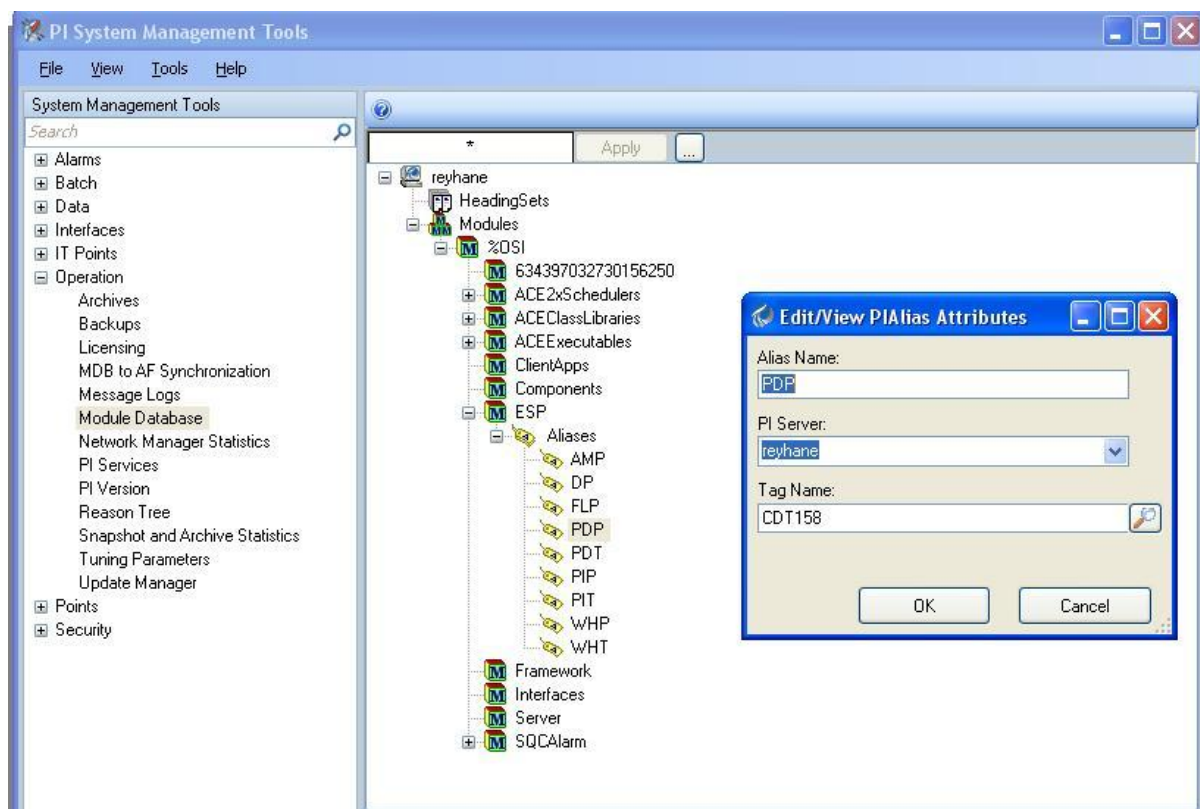


Figure 11. Module database and aliases

Using the PI ACE Wizard Add-In for .NET framework we can create the modules. For each alarm a separate module has been created. Since the modules were implemented and tested in test environment all output tags were strings with the value indicating the conditions which are met. But in the production level, the types of output tags are different; also writing a value to an output tag will trigger the PI Notification system and automatically will result in sending an email to the responsible persons with the relevant message. These all are configurable.

Each ACE Module developed with ACE corresponds to a class in a VB.NET Class Library project. The project is referred to as ACE Executable (even though the .NET Class Library project is actually a .dll assembly). The ESP executable runs in its own process and thus is independent of other ACE Executables. All the modules for alarm system have been putted in one ACE Executable, so only one set of .dll files needs to be loaded which will reduce the overall memory usage

Figure 12 demonstrates the code for Reduced Inflow alarm. All the codes for the other alarm conditions and the screen dumps related to the different steps for development are provided in appendix C.


```

ESP - Microsoft Visual Studio
File Edit View Project Build Debug Data Tools Window Community Help
Release Any CPU

ReducedInflow.vb
ReducedInflow ACECalculations()

Inherits PIACENetClassModule
Private ReducedInflow As PIACEPoint
Private WHP As PIACEPoint
Private PIP As PIACEPoint
Private PDP As PIACEPoint

'
' Tag Name/VE Variable Name Correspondence Table
' Tag Name VB Variable Name
'-----
' %OSI\ESP\PDP PDP
' %OSI\ESP\PIP PIP
' %OSI\ESP\WHP WHP
' ReducedInflow ReducedInflow
'

Public Overrides Sub ACECalculations()

    If PIP.Avg("*-24h", "*", 0) - PIP.Value > 0.1 * PIP.Avg("*-24h", "*", 0) AndAlso _
        PDP.Avg("*-24h", "*", 0) - PDP.Value > 0.1 * PDP.Avg("*-24h", "*", 0) AndAlso _
        WHP.Avg("*-24h", "*", 0) - WHP.Value > 0.1 * WHP.Avg("*-24h", "*", 0) Then

        ReducedInflow.Value = "Cond1Cond2Cond3"

    Else : ReducedInflow.Value = "OK"
    End If
End Sub

Protected Overrides Sub InitializePIACEPoints()
    PDP = GetPIACEPoint("PDP")
    PIP = GetPIACEPoint("PIP")
    WHP = GetPIACEPoint("WHP")
    ReducedInflow = GetPIACEPoint("ReducedInflow")
End Sub

```

Figure 12. Code for Reduced Inflow Alarm

After registering the context, it will be added to the tree view in ACE manager. Once the module is registered, the scheduler which is a Windows service application automatically executes the calculations according to their schedule. After the ACE Scheduler starts the calculations, it periodically checks for calculation updates.

One of the functionalities added by implementing these modules in ACE is that, it provides the ability to apply the modules to different contexts (i.e. reference to PI server or PI module). Using the same aliases for different tags made it possible to develop generic modules that could be applied to different contexts. For example in a plant with two different pumps however two different tags are used for measuring the well head pressure, but by assigning the same alias for tags (i.e. WHP) the code which is illustrated in Figure.12 can be reused. This approach was not possible with Performance Equation.

Another advantage of using ACE is the ability to import different libraries and also make the use of .NET framework. In the case of this thesis an open source library for fuzzy logic has been used which will be described more in the next section.

Figure 13 shows the output values for Reduced Inflow alarm. As you can see, for the last 24 hours just 6 events has been written to the archive; this is due to the compression functionality of PI archive which prevents storing unnecessary or duplicated data.

PI System Management Tools

System Management Tools

Search

Alarms

Batch

Data

Archive Editor

Current Values

Stale and Bad Points

Interfaces

IT Points

Operation

Points

Security

ReducedInflow

Event Count: 6 Retrieved: 6 Row:

Server: reyhane Tagname: ReducedInflow

Start Time: *-24h End Time: *-3h

Merge Type: Replace Duplicates Boundary Type: Inside

Show Filtered: Remove Filtered Values Use String Annotations?

Filter Expression:

Value	Event Time	Questionable	Annotated	Substituted
OK	6/25/2011 11:31:30 AM	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Cond1Cond2Cond3	6/25/2011 6:31:30 PM	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
OK	6/25/2011 8:50:00 PM	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Cond1Cond2Cond3	6/25/2011 8:50:30 PM	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
OK	6/25/2011 9:27:30 PM	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
OK	6/26/2011 1:40:30 AM	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
*		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Session Record

REYHANE\Reyhaneh | piadmin, piadmins, PIWorld

Figure 13. Reduced Inflow Alarm output

The reason that compression is used here is that it brings significant savings. When comparing a properly tuned Swinging Door Algorithm in a process historian with a relational database we see between 1:1000 and 1:5000 differences, thus a 1 gigabyte process historian database is a 1 to 5 terabyte database in a relational database, which is significant (Winslow 2010).

3.2 APPLYING FUZZY LOGIC

Among different conditions had to be checked there were some, where the value of an input tag was checked against a fix value. One of those conditions was motor current; if the current becomes above 185 then the alarm will be on. According to the conversation we had with Bodil Sømme, sr Process Engineer from Talisman, one of the actions which would be taken in such a case will be a decrease in frequency of motor and bring it to the slow down phase. Also when the motor current exceeds the value over 195, then another alarm will trigger and the reaction should be the shutdown of pump. Consequences of taking those reactions have been described in section 2.1.3; but what should be the reaction when the motor current is close to 185? With a Boolean logic, as it is applied now, it will not consider as a triggering point, however it is the case for some extend. Considering this situation this could be the place to apply fuzzy logic.

Also one of the contributions to this scenario was to suggest the proper frequency of the motor for the corresponding incoming value of motor current. With the existing alarm system just an alarm will be generated when the motor current passes the limit value and the responsible persons in the control or monitoring team will discuss the condition and take the action (i.e. reducing the frequency to a certain level or shutting down the pump).

3.2.1 ENVIRONMENT AND SYSTEM PREREQUISITES

To apply the fuzzy logic the same environments and systems like the ones for ACE are required because it will be an ACE module. In addition, an open source fuzzy logic library is imported to the program called “DotFuzzy”.

3.2.2 DEVELOPMENT AND CHALLENGES

During the development, one of the challenges was determination of membership functions. Since the time scope for this thesis was restricted and there were not enough technical resources (both human and documentation) available, the membership functions are created based on some assumptions. Moreover, the entire test has been applied on simulated data.

As described in section 2.3 we first need to define the membership functions for Motor Current and Motor Frequency. Based on the information we got, when the motor current raises to 186 then an alarm will be generated and corresponding action would be a decrease in the frequency of motor to make the pump slowdown. But if the motor current value reaches to 196 then the motor should be shot down. Considering this scenario, for Motor current, three different states assumed:

- 1) Normal
- 2) High
- 3) HighHigh

And for frequency this three states assumed:

- 1) Normal
- 2) Decreased
- 3) Dropped
- 4)

Figure 14 and Figure 15 demonstrate the membership functions have been used for Motor Current and Motor Frequency.

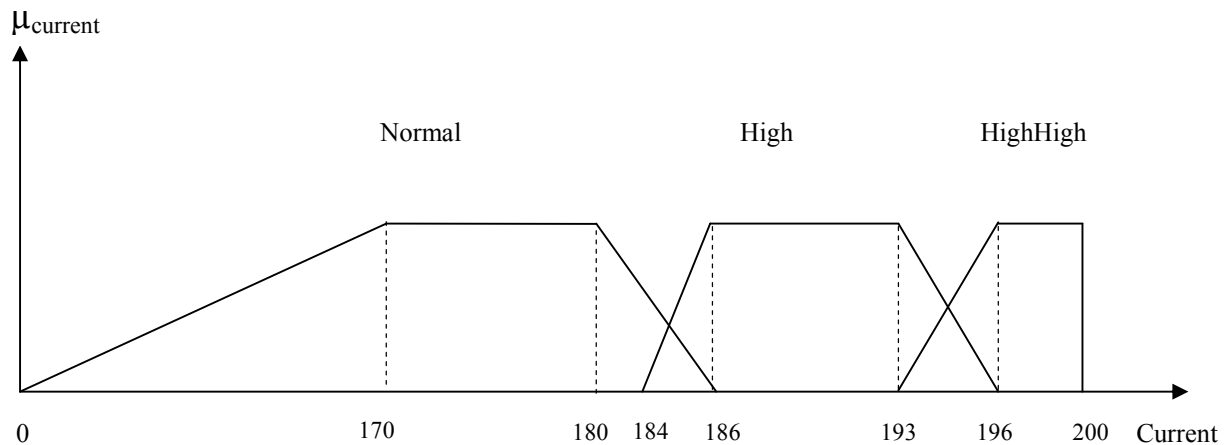


Figure 14. Motor Current membership function

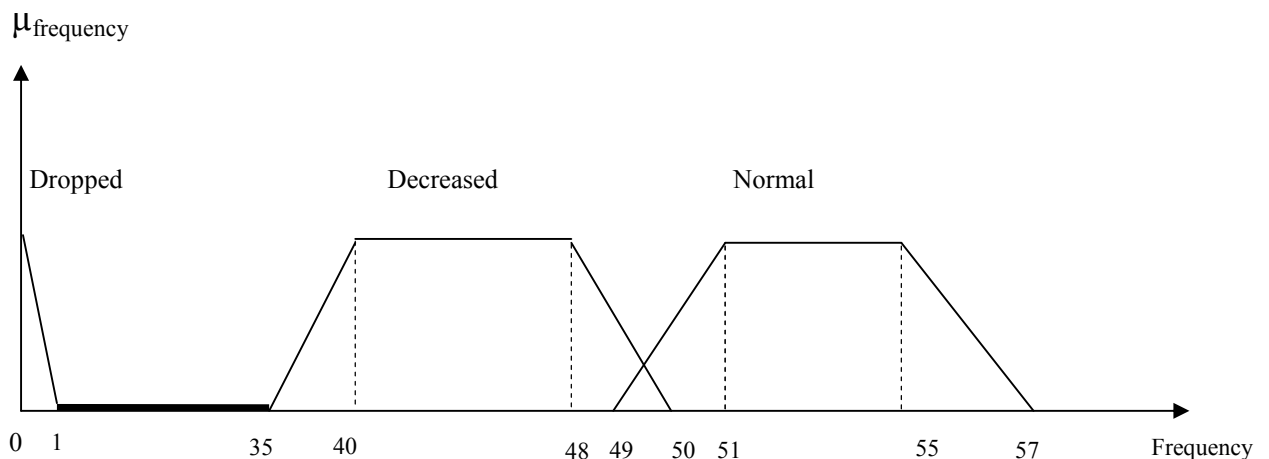


Figure 15. Motor Frequency membership function

The ranges for motor current divided to three areas representing the motor state. After 180 the membership degree related to normal state decrease until it reaches to 186 since 186 is not a normal current. The High state starts from 184 and the degree increase till 186, after that, all

the current values are consider as high but from 193, the membership degree start to decrease. The same is applied for HighHigh state with ranges from 193 to 200.

For Frequency, the membership function is different. Since the action of having motor current value above 195 is to shutdown the pump, so the frequency range should be below 1 which will result to turn off the pump motor. Hence, the range of [0, 1] considered as dropped state and the frequency between [1, 35] would have zero degree.

Based on the membership functions the rules were created; for this scenario three rules have been defined as follow:

```
"IF (MotorCurrent IS HiighHigh) THEN MotorFrequency IS Dropped"
```

```
"IF (MotorCurrent IS High) THEN MotorFrequency IS Decreased"
```

```
"IF (MotorCurrent IS Normal) THEN MotorFrequency IS Normal"
```

To implement this fuzzy logic an open source fuzzy logic library called “DotFuzzy” have been used which uses the trapezoidal membership function with the Centroid method; the code are demonstrated in appendix C.

The scheduling mode for this module is natural, meaning that every time that AMP tag gets a value it will be first fuzzified and then based on the rules a frequency related to this current will be calculate and the defuzzified. As described in section 3.1.2 after registering the module the .dll will be added to the modules list in Manager and executed by ACE scheduler.

The results of applying this logic are shown in Figure 16. As one can see when the current is more than 185 a decrease in the frequency will lead to reduction in motor revolution.

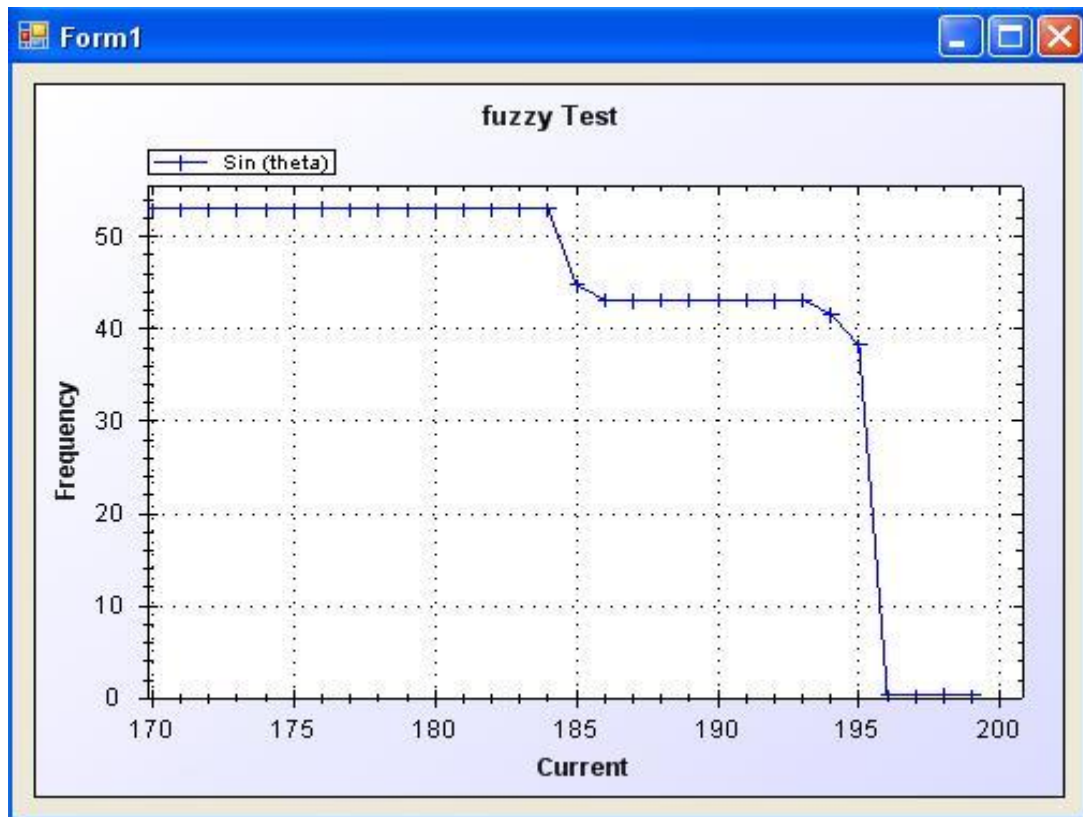


Figure 16. The Frequency results after applying fuzzy logic

As you it is illustrated in the diagram, instead of having sharp edges, the frequency decreases smoother. This will result in less stress on the pump and it is expected that in long term this might increase the pump life time and production rate.

In addition to the smooth output that is received for the frequency, using fuzzy logic makes coding easier and more understandable and readable since we are dealing with linguistic variables. This will omit the need to know and implement some complex mathematical models and formulas. The code is provided in appendix C.

3.3 ALARM SYSTEM USING STREAMINSIGHT AND PI ADAPTERS

In this part, the development of an application using Microsoft StreamInsight and OSIsoft PI adapters for StreamInsight is described. The aim of this approach was to examine the functionality and performance of this new product and see if we can get the same result as the one implemented by ACE.

The application tested and run using data from PI random interface. The part of code that is important to explain and is the contribution is presented in this section and the rest is presented in appendix C.

3.3.1 ENVIRONMENT AND SYSTEM PREREQUISITES

To run the Microsoft StreamInsight application that uses PI for StreamInsight adapters to access a PI System, the followings needed to be installed in the machine:

- Microsoft StreamInsight 1.1: Microsoft StreamInsight is included with SQL Server 2008 R2. However, SQL Server does not need to be installed to use Microsoft StreamInsight.
- .NET Framework 3.5.
- PI SDK 1.3.8.387 or later; PI SDK is installed by the PI for StreamInsight installation program
- Visual Studio 2008 or 2010. Visual Studio 2010 is preferred as there are additional StreamInsight tools available

Also a PI server should be up and running in order to get data from and write data back to server.

3.3.2 DEVELOPMENT AND CHALLENGES

Microsoft StreamInsight was first published in April 2009 and the first version of PI adapters for StreamInsight was published in 1th June 2011(The prerelease version was available in OSIsoft virtual campus website since March 2011)(Schindlauer 2011).

Hence, being one of the first developers using these products, faced with some challenges and difficulties during the development phase. There were few resources especially for this scenario, also as the nature of new technology implies, there were some restrictions which are supposed to be removed in later version that as is discussed in the following part.

The StreamInsight server is the processing engine that runs user-defined queries across event streams. In this project, the StreamInsight server runs as an in-process embedded service in the application. As an embedded service, the queries are included in the application. The application is developed using C#.Net 2010. To run the application there has to be a PI server up and running for the reason that the PI input adapter receives PI snapshot events from the PI Update Manager and converts those events into a form that can be read and queued into the StreamInsight server and the PI output adapter receives events from the StreamInsight server and sends the data to the PI archive.

The .NET code that implements the adapters and interfaces to PI is provided in three assemblies:

```
OSIsoft.ComplexEventProcessing.Adapters.dll
```

```
OSIsoft.ComplexEventProcessing.PINet.dll
```

```
OSIsoft.ComplexEventProcessing.Archive.dll
```

To test the adapters and StreamInsight engine results, following LINQ queries were written to monitor the same Reduced Inflow conditions shown in Figure 12:

```
// compute the 24h average for each Id separately
var avgStream = from e in dataStream
                group e by e.Id into g
                from win in g
                    .AlterEventDuration(e => TimeSpan.FromHours(24))
                    .SnapshotWindow(SnapshotWindowOutputPolicy.Clip)
                select new { Id = g.Key, Avg = win.Avg(e => e.Value) };

// for each Id, compare the average with the real-time value
var check = from a in avgStream
            join b in dataStream
            on a.Id equals b.Id
            select new { a.Id, Check = (a.Avg - b.Value) > a.Avg * 0.1 };

// turn the result into a continuous signal for each Id:
var checkSignal = check.AlterEventDuration(e => TimeSpan.MaxValue)
    .ClipEventDuration(check, (e1, e2) => e1.Id == e2.Id);
```



```
// compute the periods where condition is met over all Ids:
var result = from conditions in (from win in checkSignal
    .Where(e => e.Check)
    .SnapshotWindow(SnapshotWindowOutputPolicy.Clip)
    select new{
        Count = win.Count()
    })
    where (conditions.Count >2)
    select new PIDataSingle()
    {
        Path = "SI.ReducedInflow",
        Status = 0,
        Value = conditions.Count
    };
```

One of the restrictions here was using nested queries. Current version of StreamInsight does not support nested queries. So first we need to calculate the average and then using those results and create another query to subtract the average from current value.

After testing the two (ACE and StreamInsight) of the same condition, the results were not the same. The number of times where the conditions were met in ACE was more than in StreamInsight.

In order to find the reason for differences, some changes were applied to the conditions and code. Since the logic was correct, the average method seemed to be the only reason of differences. To test that, an ACE module with following code had implemented:

```
AvrgACE.Value = AlarmTest_Input_Float32_1.Avg("*-24h", "*", 0)
```

Below is the query that used to generate the average over last 24 hours:

```
var result = from win in input
    .AlterEventDuration(e => TimeSpan.FromMinutes(24))
    .SnapshotWindow(SnapshotWindowOutputPolicy.Clip)
    select new PIDataSingle()
    {
        Path = "AvrgSI",
        Status = 0,
        Value = win.Avg(e => e.Value)
    };
```

As presented in Figure 14 the results was totally different, however, the compression and Exception was turned off for all the tags.

The figure shows two side-by-side screenshots of the 'Archive Editor - PI System Management Tools' application. The left window is titled 'AvgSI' and shows 1908 events. The right window is titled 'AvgACE' and shows 1909 events. Both windows have a similar interface with a menu bar (File, View, Tools, Help), a toolbar, and a table of event data. The table columns are 'Value', 'Event Time', 'Questionable', and 'Annotated'. The 'Event Time' column shows dates and times from 6/25/2011 12:31:27 PM to 6/25/2011 12:36:57 PM. The 'Value' column contains numerical values. The 'Questionable' and 'Annotated' columns have checkboxes.

Figure 17. Results of average function for ACE and StreamInsight

Based on the analyses of data and also the help of Roman Schindlauer and Erwin Gove from Microsoft and OSIsoft teams, finally we came to the conclusion that the reason of having such different values is that ACE engine uses an average method which is time-weighted but StreamInsight uses the normal average method (i.e. sum/count); this will cause a decrease in the number of alarms generated by StreamInsight comparing to ACE. Because the average should be subtracted from the current value and since the current value is same for both engines so the result of each equation would be different.

As the end result, using this application, we can generate alarm by reading data from PI and writing the result to PI. But if the data are time-weighted this application will not produce the proper alarms and the results will not be too accurate. So in such scenarios the modules implemented using ACE engine must be used.

If the normal average is sufficient this application, especially in the case that number of events is so high, will perform better because StreamInsight engine runs in-memory so we get very superior speed and performance. Using ACE the performance will be around 1000 events/second but using PI for StreamInsight adapters and StreamInsight engine it could increase to 10,000 because of in-memory calculations.

Although, running in-memory will have memory usage subsequently, average is an incremental aggregation, so no matter how long the time window is, we only need to maintain the current sum and count in StreamInsight. Considering a few bytes for this information per tag, we could import a million tags and use a few megabytes. Of course other parts of the query will also have a memory footprint, but the 24h sliding average should not be a problem by itself.¹

¹ Roman Shindlauer

One of the other differences between ACE module and StreamInsight application is that ACE is requesting data from archive while StreamInsight is receiving data from snapshot system. Since data archived in PI goes through compression using a modified swinging door algorithm, if the data mining strategy is sensitive to such compression and prefer to process snapshots instead of archived data, StreamInsight will be useful. But if the compression is needed, then a compression algorithm must be implemented in StreamInsight.

4 CONCLUSION

4.1 CONCLUSION

This thesis had three goals: Implementing an alarm system to monitor an ESP pump using PI ACE, approaching the same goal by using Microsoft StreamInsight, comparing the result and performance, and finally applying fuzzy logic to an alarm condition in order to add more functionality.

The alarm system which is already in use in Talisman Company was implemented using PI Performance Equation. Implementing the system with ACE brought advantages which we could not achieve using Performance Equation. These are: to be able to use fuzzy logic, generating generic modules which could apply to multiple contexts and doing recalculation. ACE also provides the developer the chance of using all functionalities of .Net framework, easily integrating the system, and making complex calculations which is not possible to do with Performance Equation.

As presented in last chapter, the results achieved from implementing the system using two technologies namely ACE and StreamInsight were different, StreamInsight generated less alarm than ACE because of the different average method which are in used. Also the other differences between two platforms could be listed as below:

- StreamInsight is an in-memory data processing engine.
- StreamInsight uses LINQ but programming in ACE is with Visual Basic or Visualbasic.Net, so the developers are required to know LINQ in addition to .NET Framework.
- ACE is requesting data from archive while StreamInsight is receiving data from snapshot system.
- Current version of StreamInsight does not support the nested queries.
- For some scenarios like the one is implemented for this thesis, using LINQ requires more coding and is complex comparing to ACE.
- Using the ACE will omitted the need to implement any scheduling service.
- PI ACE supports recalculation but StreamInsight does not have such option.

Moreover the environments and prerequisites are different which should be considered when it comes to financial part of the solution.

Finally, a new functionality has been added to motor current alarm. Using fuzzy logic we monitored the motor current and suggest the proper frequency related to this current. Doing this, results in a smoother reduction in the motor frequency which is expected to increase the pump life time and production in long term. Also representing or modeling the problem linguistically reduced development time and increased the comprehension and understanding of the solution made.

4.2 FURTHER WORK

Since all the tests were applied on simulated data, the solutions should be also tested on real data before application be used in production environment. Moreover membership functions which were created under some assumptions should be further analyzed to ensure that they demonstrate the expected behavior and relations.

Also a user-defined time-weighted average method could be developed in order to make the StreamInsight application act the same as corresponding ACE module.

5 REFERENCES

Alimonti, C. and G. Falcone (2002). Knowledge Discovery in Databases and Multiphase Flow Metering: The Integration of Statistics, Data Mining, Neural Networks, Fuzzy Logic, and Ad Hoc Flow Measurements Towards Well Monitoring and Diagnosis. SPE Annual Technical Conference and Exhibition. San Antonio, Texas.

Banks, W. and G. Hayward (2001). "Fuzzy logic in embedded microcomputers and control systems."

Bates, R., C. Cosad, et al. (2004). "Taking the pulse of producing wells—ESP surveillance."

Camilleri, L. A. P. and J. Macdonald (2010). How 24/7 Real-Time Surveillance Increases ESP Run Life and Uptime. SPE Annual Technical Conference and Exhibition. Florence, Italy.

Cohen, D. J. (1997). CAPTAIN FIELD ELECTRIC SUBMERSIBLE PUMP, CONDITION MONITORING AND COMPLETION SYSTEMS.

Danquigny, J. A., R. Daian, et al. (2007). Production Optimization by Real-Time Modeling and Alarming: The Sendji Field Case. SPE Annual Technical Conference and Exhibition. Anaheim, California, U.S.A.

Engelbrecht, A. P. (2008). Computational Intelligence an introduction, John Wiley & Sons Ltd.

Gray, C. L. (1994). "Electrical Submersible Pumps: How to Aachive Longer Lives Throught Simple Computer Data Monitoring ".

Haapanen, B. E. and M. G. Gagner (2010). Remote Monitoring and Optimization of Electrical Submersible Pumps Utilizing Control Algorithms. Canadian Unconventional Resources and International Petroleum Conference. Calgary, Alberta, Canada.

Kaehler, S. D. (2003). "Fuzzy Logic Tutorial." Retrieved 16.05, 2011, from http://www.seattlerobotics.org/encoder/mar98/fuz/fl_part1.html

Karakose, M. and E. Akin (2010). "BLOCK BASED FUZZY CONTROLLERS." International Journal of Research and Reviews in Applied Sciences.

Luckham, D. (2006). "What's the Difference Between ESP and CEP." Retrieved 01.06, 2011, from <http://www.complexevents.com/2008/08/28/a-short-history-of-complex-event-processing-part-2-the-rise-of-cep/>.

Luckham, D. C. (2001). The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems, Addison-Wesley Longman

Mendel, J. M. (1995). "Fuzzy logic systems for engineering: a tutorial." Proceedings of the IEEE **83**(Compendex): 345-377.

Microsoft (2007). "LINQ: .NET Language-Integrated Query." Retrieved 07.06, 2011, from <http://msdn.microsoft.com/en-us/library/bb308959.aspx>.

Microsoft (2011). "StreamInsight Server Concepts." Retrieved 11.02, 2011, from <http://msdn.microsoft.com/en-us/library/ee391434.aspx>.

References

Moffatt, T. and D. Craig (2001). "Innovative Real-Time Well-Monitoring Systems (ESP Applications)." SPE Production & Operations **16**(3).

Noonan, S. G., M. Kendrick, et al. (2005). "Impact of Transient Flow Conditions on Electric Submersible Pumps in Sinusoidal Well Profiles: A Case Study." SPE Production & Operations **20**(3).

O.C Olmstead, J. L. D. (2009). "Electric Submersible Pump Monitoring and Alarms-establishing a philosophy document." SPE-GULF COAST.

OSIsoft (2006). "ACE User Manual." Retrieved 01.05, 2011, from <http://elearn.viewcentral.com/content/ositraining/669eac132653ondemandmenu/ondemandmenu/index.html>.

OSIsoft (2006). "PI Advanced Computing Engine (ACE) version 3.0.1." Retrieved 11.05, 2010, from http://vcampus.osisoft.com/training_center/p/piace.aspx.

OSIsoft (2006). "Introduction to PI System Management." Retrieved 01.05, 2011, from <http://elearn.viewcentral.com/content/ositraining/669eac132653ondemandmenu/ondemandmenu/index.html>.

OSIsoft (2009). "PI ProcessBook User Guide." Retrieved 01.05, 2011, from <http://elearn.viewcentral.com/content/ositraining/669eac132653ondemandmenu/ondemandmenu/index.html>.

OSIsoft (2011). "PI for StreamInsight User Guid." Retrieved 02.006, 2011, from <http://vcampus.osisoft.com/downloadcenter/download.aspx>.

Schindlauer, R. (2011). "OSIsoft PI Adapter Release Candidate ". Retrieved 14.04, 2011, from http://sqlblog.com/blogs/stream_insight/archive/2011/03/21/osisoft-pi-adapter-release-candidate.aspx.

Thornhill, D. G. and D. Zhu (2009). Fuzzy Analysis of ESP System Performance. SPE Annual Technical Conference and Exhibition. New Orleans, Louisiana.

Torsten Grabs, Roman Schindlauer, et al. (2009). Introducing Microsoft StreamInsight, Microsoft.

Winslow, R. (2010). "Relational Database vs. Process Historian for process data; use BOTH!". Retrieved 18.05, 2011, from <http://osipi.wordpress.com/>.

APPENDIX A: PI POINT ATTRIBUTES

All information in this appendix is retrieved from (OSIsoft 2006).

Point Name: Tag Attribute

Many PI users use the terms tag and point interchangeably, which is fine. Technically though, the tag is actually just the name of the point. Follow these rules for naming PI points:

- The name must be unique on the PI Server
- The first character must be alphanumeric, the underscore (_), or the percent sign (%)
- No control characters are allowed; such as linefeeds or tabs
- The following characters are not allowed:
* ' ? ; { } [] | \ ' ' “

Class of Point: PtClass Attribute

The attributes that you need to conFIGure for a particular point depend on what the point is for. PI provides several different classes of points, each of which provides a slightly different set of attributes to work with. You can also build your own point classes.

Points that represent data from a PI interface are always in the Classic point class. The list of available PI point classes is as follows:

- **Classic:** The Classic point class includes attributes used by interfaces.
- **Base:** The Base class is a common set of attributes that all point classes include. The Base class includes both system-assigned and user-assigned attributes. This is the minimum set of attributes that a PI point needs in order to function.
- **Alarm:** The Alarm class is used for alarm points.
- **Totalizer:** The Totalizer class is for a type of point that represents a running total of data. There are many different kinds of Totalizer points.
- **SQC_Alarm:** The SQC_Alarm class is for SQC_Alarm points

Data Type of Point: PointType Attribute

Use the **Type** attribute to specify the data type of the point values.

Point Type	When to Use It
Digital	Use the Digital point type for points whose value can only be one of several discrete states, such as ON/OFF or Red/Green/Yellow.
Int16	Use the Int16 point type for points whose values are integers between 0 and 32767 (15-bit unsigned integers).
Int32	Use the Int32 point type for points whose values are integers between -2147450880 and 2147483647 (32-bit signed integers).
Float16	Use the Float16 point type for floating point values, scaled. The accuracy is one part in 32767.
Float32	Use the Float32 point type for single-precision floating-point values, not scaled (IEEE floating points).
Float64	Use the Float64 point type for double-precision floating-point values, not scaled (IEEE floating points).
String	Use the String point type for strings of up to 976 characters.
Blob	Blob stands for Binary Large Object. Use the Blob point type to store any type of binary data up to 976 bytes.
Timestamp	Use the Timestamp point type for any time/date in the range 1-jan-1970 to 1-Jan-2038 Universal Time (UTC).

Data Source: PointSource Attribute

The **PointSource** attribute specifies which interface is the data source for this point. Set the **PointSource** attribute to match the point source character for the interface.

The default point source is **L**, which stands for “Lab”. Depending on your installation, the default point source is either **L** or **Lab**. Use **L** or **Lab** for points that are not associated with any interface to specify lab-input points.

Interface ID Number: Location1 Attribute

The **Location1** attribute is only for interface points, meaning points that get their data from a PI interface, as opposed to some other source. Most interfaces use the Location1 attribute to specify the interface ID number.

Setting Scan Class: Location4 Attribute

The **Location4** attribute is only for interface points, meaning points that get their data from a PI interface, as opposed to some other source. Each PI interface has one or more scan classes for scheduling data collection. You set the Location4 attribute for a point to specify which of the interface's scan classes you want to use.

Exception Specifications

Exception reporting specifications determine which events the interface sends to PI and which it discards.

Each point can set the following three attributes to configure the exception reporting specifications:

Specification	Attribute	How and When to Use it
Exception Deviation	ExcDev	Use this attribute to specify how much a point value must change before the interface reports the new value to PI. Use ExcDev to specify the exception deviation in the point's engineering units. As a general rule, you should set the exception slightly smaller than the precision of the instrument system.
	ExcDevPercent	You can use ExcDevPercent instead of ExcDev. ExcDevPercent sets the exception deviation as a percentage of the Span attribute, but be careful. If your Span attribute is not set correctly, your exception reporting will be wrong too. A typical exception deviation value is about 1% of Span.
Exception Minimum	ExcMin	Use ExcMin to limit how often (in seconds) the interface reports a new event to PI. For example, if you set ExcMin to five, then the interface discards any values collected within five seconds of the last reported value. ExcMin is typically set to zero.
Exception Maximum	ExcMax	Set ExcMax to the maximum length of time (in seconds) you want the interface to go without reporting a new event to PI. After this time, the interface will report the new event to PI without applying the exception deviation test.

Compression Specifications

The compression specifications include a flag that allows you to turn compression on or off. It is recommended to turn compression on for all real-time points in the system. You usually turn compression off for points with manually-entered data, production targets, control limits, and so on.

For each point, you can set four attributes to conFIGure the compression specifications.

Specification	Attribute	What it Does
Compression Flag	Compressing	Turns compression on or off (set to 1 to turn compression on or 0 to turn compression off).
Compression deviation	CompDev	As a rule of thumb, set CompDev to the precision of the instrument. Set it a little “loose” to err on the side of collecting, rather than losing data. After collecting data for a while, go back and check the data for your most important tags and adjust CompDev if necessary. Use CompDev to specify the compression deviation in the point's engineering units.
	CompDevPercent	Use CompDevPercent to specify the compression deviation as a percent of the point's Span attribute.
Compression minimum time	CompMin	Sets a minimum limit on the time between events in the Archive. Set the CompMin attribute to zero for any point coming from an interface that does exception reporting. You typically use CompMin to prevent an extremely noisy point from using a large amount of archive space
Compression maximum	CompMax	CompMax sets a maximum limit on the time between events in the Archive. If the time since the last recorded event is greater than or equal to CompMax, then PI automatically stores the next value in the Archive, regardless of the CompDev setting. You typically set CompMax to the same value for all points in the system. It's common practice to choose a CompMax setting of one work shift (for example, 8 hours).

Configuring Shutdown Events: Shutdown

The **Shutdown** attribute has two possible values: **1 (On)** and **0 (Off)**. If the PI Server shuts down, PI writes a shutdown event to all points that have the shutdown flag set to **1 (On)**. Set **Shutdown** to **Off** for points on interfaces that are buffered. The buffering service restores the data for these tags as soon as it connects to the PI Server again.

Point Value Range: Zero, Span and Typical Value

The **Zero**, **Span**, and **Typical Value** attributes specify the range of values for a point.

Zero	Indicates a point's lowest possible value. Zero does not have to be the same as the instrument zero, but that is usually a logical choice. This attribute is required for all numeric data type points and is critically important for float16 points.
Span	The difference between the top of the range and the bottom of the range. This attribute is required for all numeric data type points.
Typical Value	Documents an example of a reasonable value for this point. For a numeric tag, it must be greater than or equal to the zero, and less than or equal to the zero plus the span.

Point Security: PtOwner, PtGroup, PtAccess, DataOwner, DataGroup, DataAccess

Each point has different, configurable, access privileges for its data and its point configuration. To control who has access to what, you assign an owner and a group for each point's data and attributes, respectively. Then you set owner, group and world privileges.

PI point security is divided into two separate pieces, Data Access and Point Access.

Data Access	Specifies who has access to a point's data values (Snapshot and Archive data).
Point Access	Specifies who has access to the point's attributes (Zero, Span, Descriptor, and so on).

You can have different owners and different group access for a point's attributes than for the point's data. So, for example, one user might be allowed to edit the data for a point, but not be allowed to edit the attributes of that point.

APPENDIX B: MEMBERSHIP FUNCTIONS

Below are some of the common membership functions retrieved from (Engelbrecht 2008)

Triangular functions (refer to Figure10 (a)), defined as

$$\mu_A(x) = \begin{cases} 0 & \text{if } x \leq \alpha_{min} \\ \frac{x - \alpha_{min}}{\beta - \alpha_{min}} & \text{if } x \in (\alpha_{min}, \beta] \\ \frac{\alpha_{max} - x}{\alpha_{max} - \beta} & \text{if } x \in (\beta, \alpha_{max}) \\ 0 & \text{if } x \geq \alpha_{max} \end{cases}$$

Trapezoidal functions (refer to Figure 10(b)), defined as

$$\mu_A(x) = \begin{cases} 0 & \text{if } x < \alpha_{min} \\ \frac{x - \alpha_{min}}{\beta_1 - \alpha_{min}} & \text{if } x \in (\alpha_{min}, \beta_1] \\ \frac{\alpha_{max} - x}{\alpha_{max} - \beta_2} & \text{if } x \in (\beta_2, \alpha_{max}) \\ 0 & \text{if } x > \alpha_{max} \end{cases}$$

S-membership functions, defined as

$$\mu_A(x) = \begin{cases} 0 & \text{if } x < \alpha_{min} \\ 2 \left(\frac{x - \alpha_{min}}{\beta - \alpha_{min}} \right)^2 & \text{if } x \in (\alpha_{min}, \beta] \\ 1 - 2 \left(\frac{x - \alpha_{max}}{\alpha_{max} - \alpha_{min}} \right)^2 & \text{if } x \in (\beta, \alpha_{max}) \\ 1 & \text{if } x > \alpha_{max} \end{cases}$$

Γ -membership functions, defined as

$$\mu_A(x) = \begin{cases} 0 & \text{if } x \leq \alpha \\ 1 - e^{-\gamma(x-\alpha)^2} & \text{if } x > \alpha \end{cases}$$

Logistic function (refer to Figure 10(c)), defined as

$$\mu_A(x) = \frac{1}{1 + e^{-\gamma x}}$$

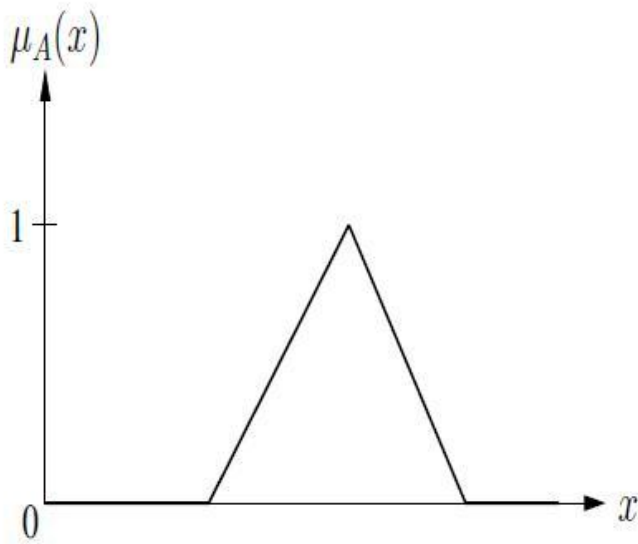
Exponential-like function, defined as

$$\mu_A(x) = \frac{1}{1 + \gamma(x - \beta)^2}$$

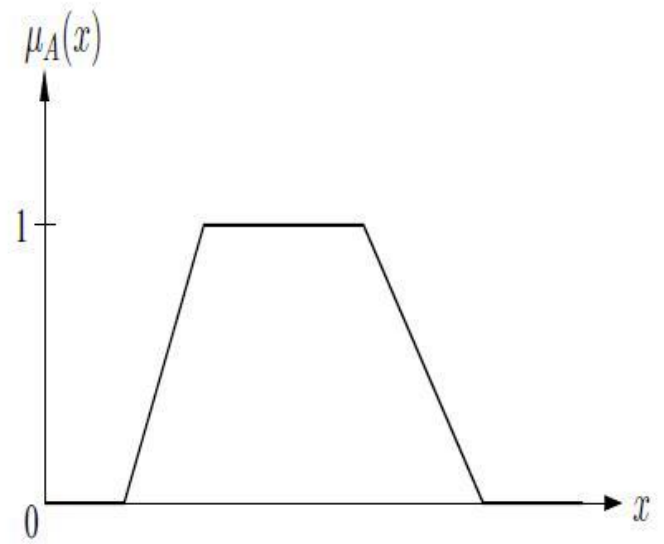
With $\gamma > 1$.

Gaussian function (refer to Figure 10.3(d)), defined as

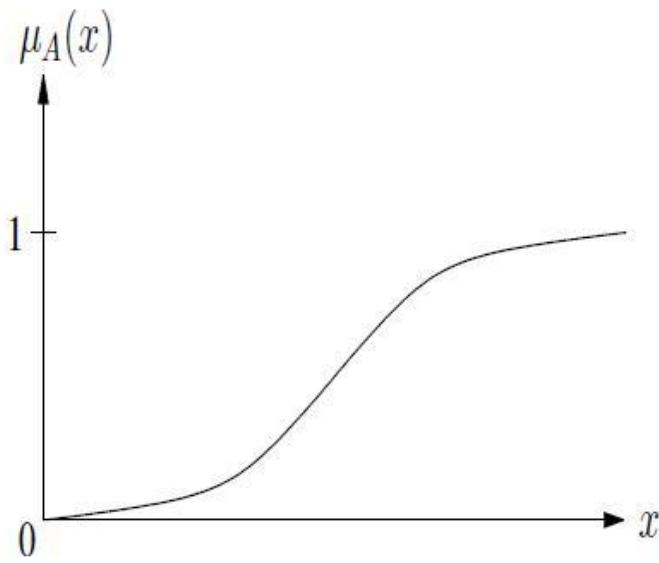
$$\mu_A(x) = e^{-\gamma(x-\beta)^2}$$



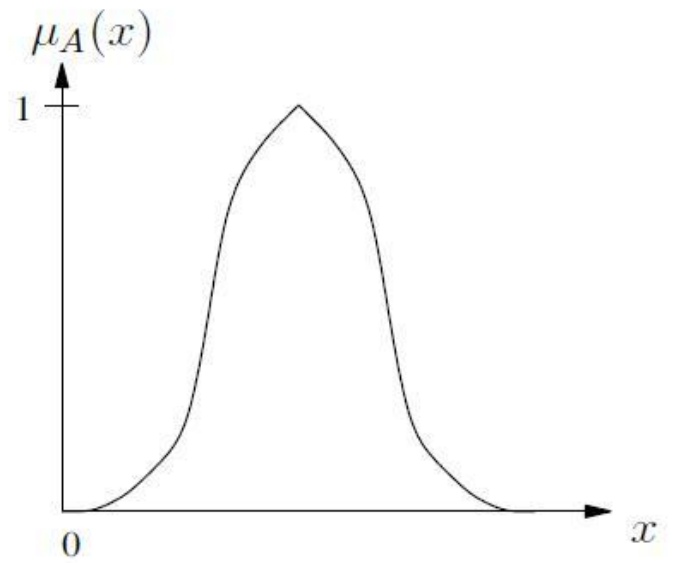
(a) Triangular Function



(b) Trapezoidal Function



(c) Logistic Function



(d) Gaussian Function

Figure 18. Example membership functions for fuzzy Sets(Engelbrecht 2008)

APPENDIX C: CODE AND SCREEN DUMPS

In this section the screen dumps and code for the applications are provided. Also the development steps for ACE modules have been presented by screen dumps.

Alarm System Using ACE

The ACE modules were created using the Add-In for .NET framework called ACE wizard.

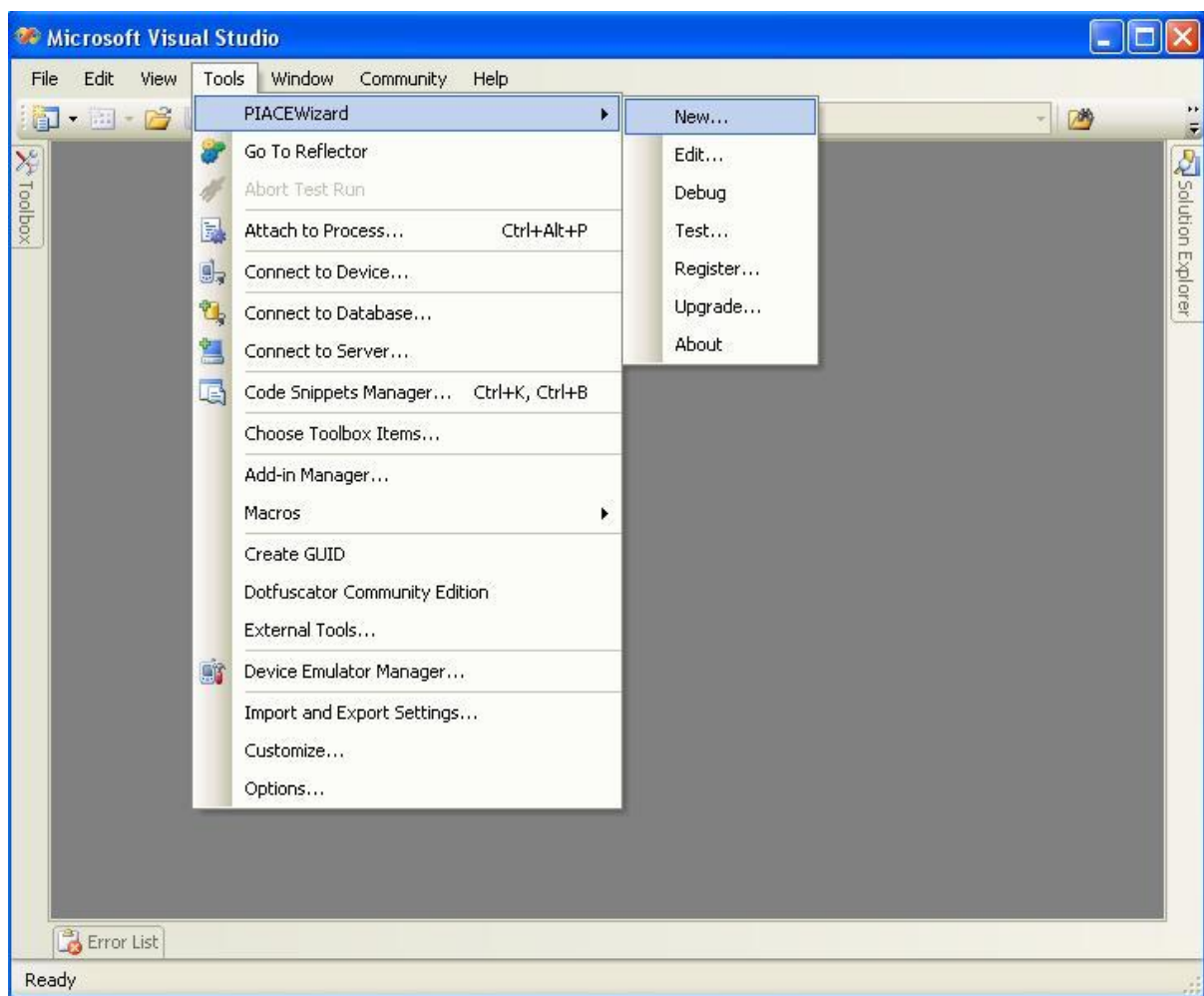


Figure 19. ACE Wizard

To make a module, first we should provide the executable name and module name and also select the server.

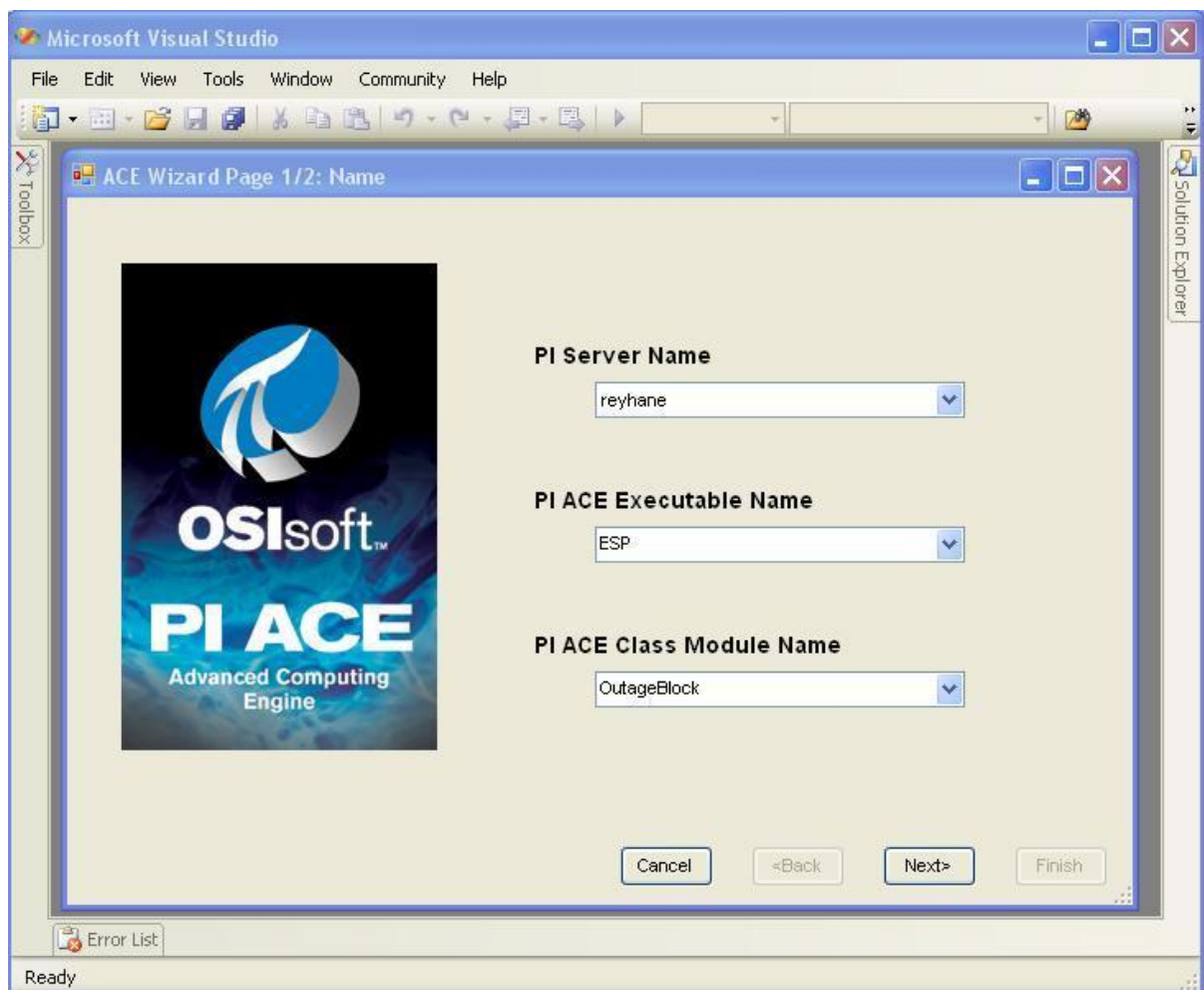


Figure 20. Define module and context

Now the next step is to select the input and output tags, here we can do the search both by tag name or alias; there is a possibility to create a tag too.

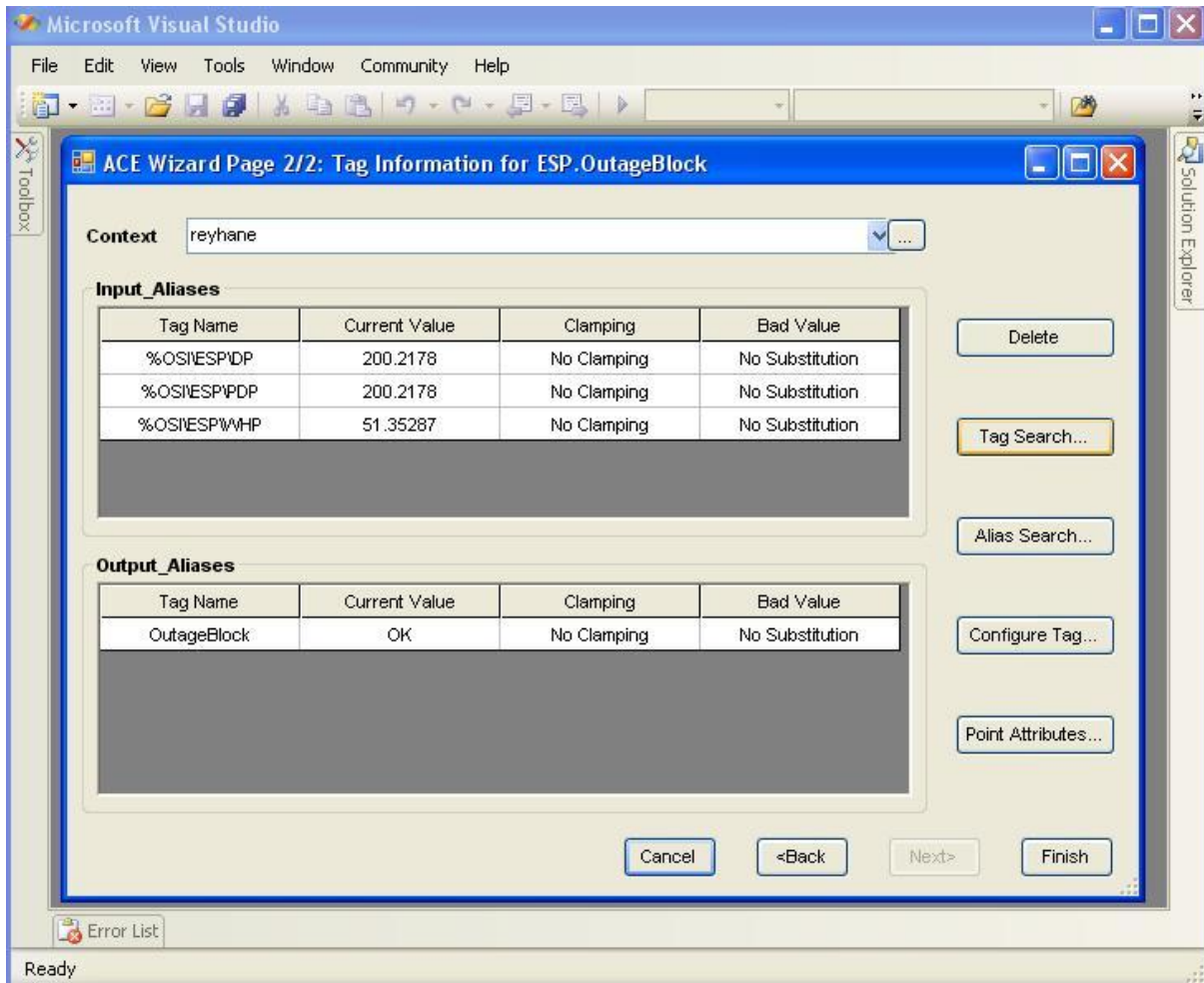
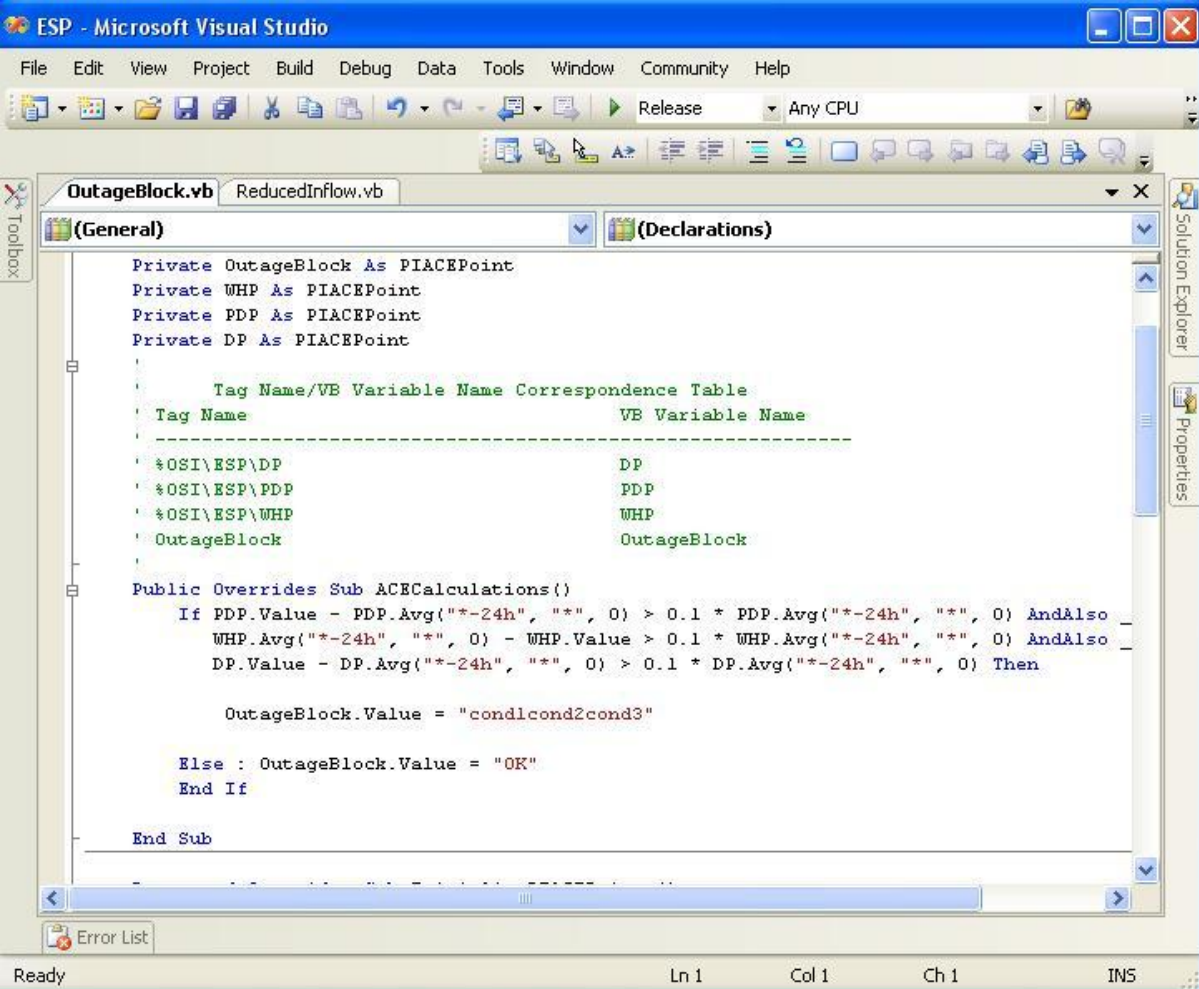


Figure 21. Selecting the input and output tags

After selecting the tags we can apply the logic and calculation using Visual Basic.NET environment. Following is the code for OutageBlock Alarm.



```
ESP - Microsoft Visual Studio
File Edit View Project Build Debug Data Tools Window Community Help
Release Any CPU
OutageBlock.vb ReducedInflow.vb
(General) (Declarations)
Private OutageBlock As PIACEPoint
Private WHP As PIACEPoint
Private PDP As PIACEPoint
Private DP As PIACEPoint
'
' Tag Name/VE Variable Name Correspondence Table
' Tag Name VE Variable Name
'-----
' %OSI\ESP\DP DP
' %OSI\ESP\PDP PDP
' %OSI\ESP\WHP WHP
' OutageBlock OutageBlock
Public Overrides Sub ACBCalculations()
If PDP.Value - PDP.Avg("**-24h", "**", 0) > 0.1 * PDP.Avg("**-24h", "**", 0) AndAlso
WHP.Avg("**-24h", "**", 0) - WHP.Value > 0.1 * WHP.Avg("**-24h", "**", 0) AndAlso
DP.Value - DP.Avg("**-24h", "**", 0) > 0.1 * DP.Avg("**-24h", "**", 0) Then
OutageBlock.Value = "cond1cond2cond3"
Else : OutageBlock.Value = "OK"
End If
End Sub
Error List
Ready Ln 1 Col 1 Ch 1 INS
```

Figure 22. OutageBlock Alarm module

After writing the code, we should first debug and test it; then it will be possible to register the module. These steps should be done using the wizard Add-In.

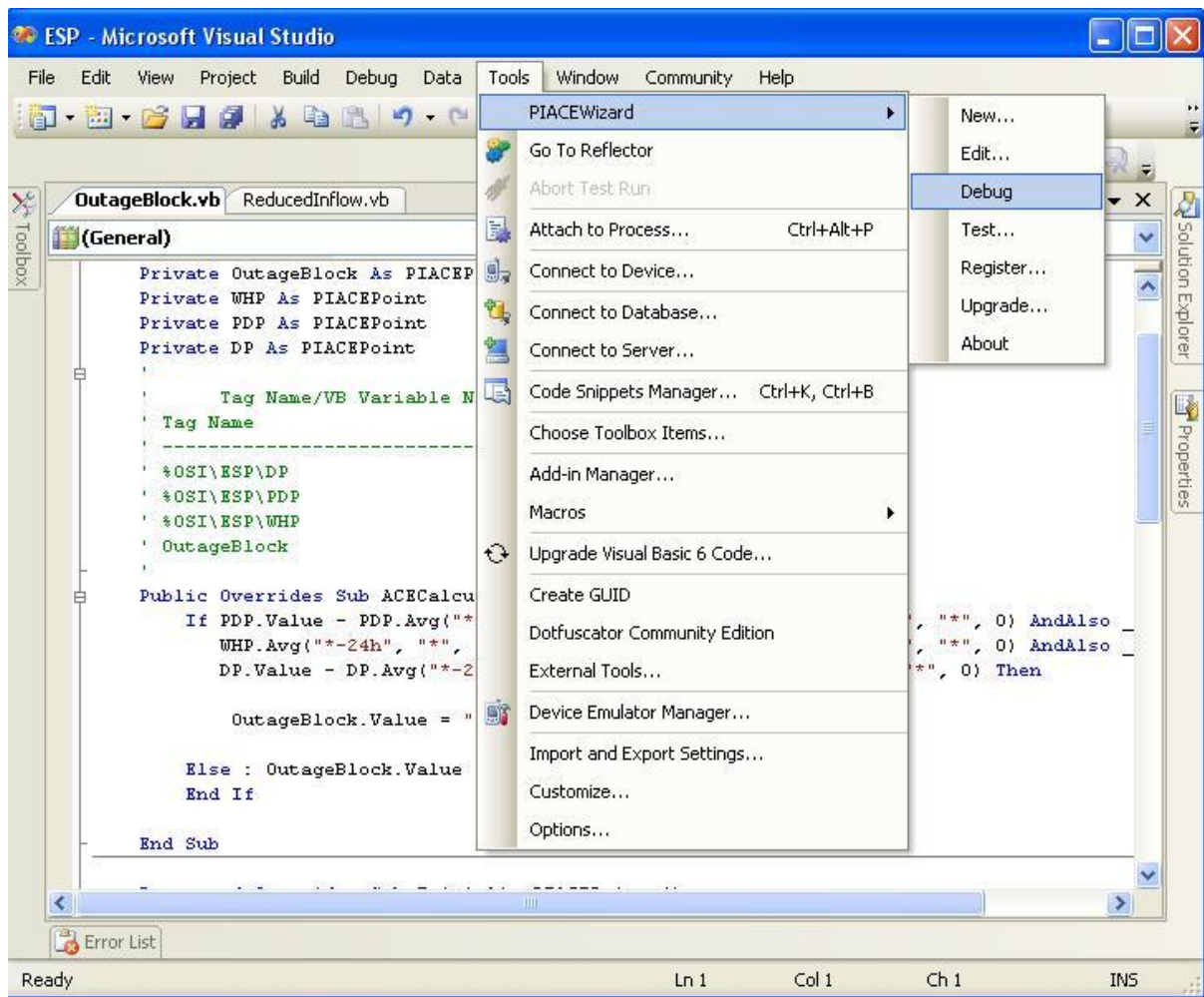


Figure 23. Debugging a module

Registration could be clock base or natural.

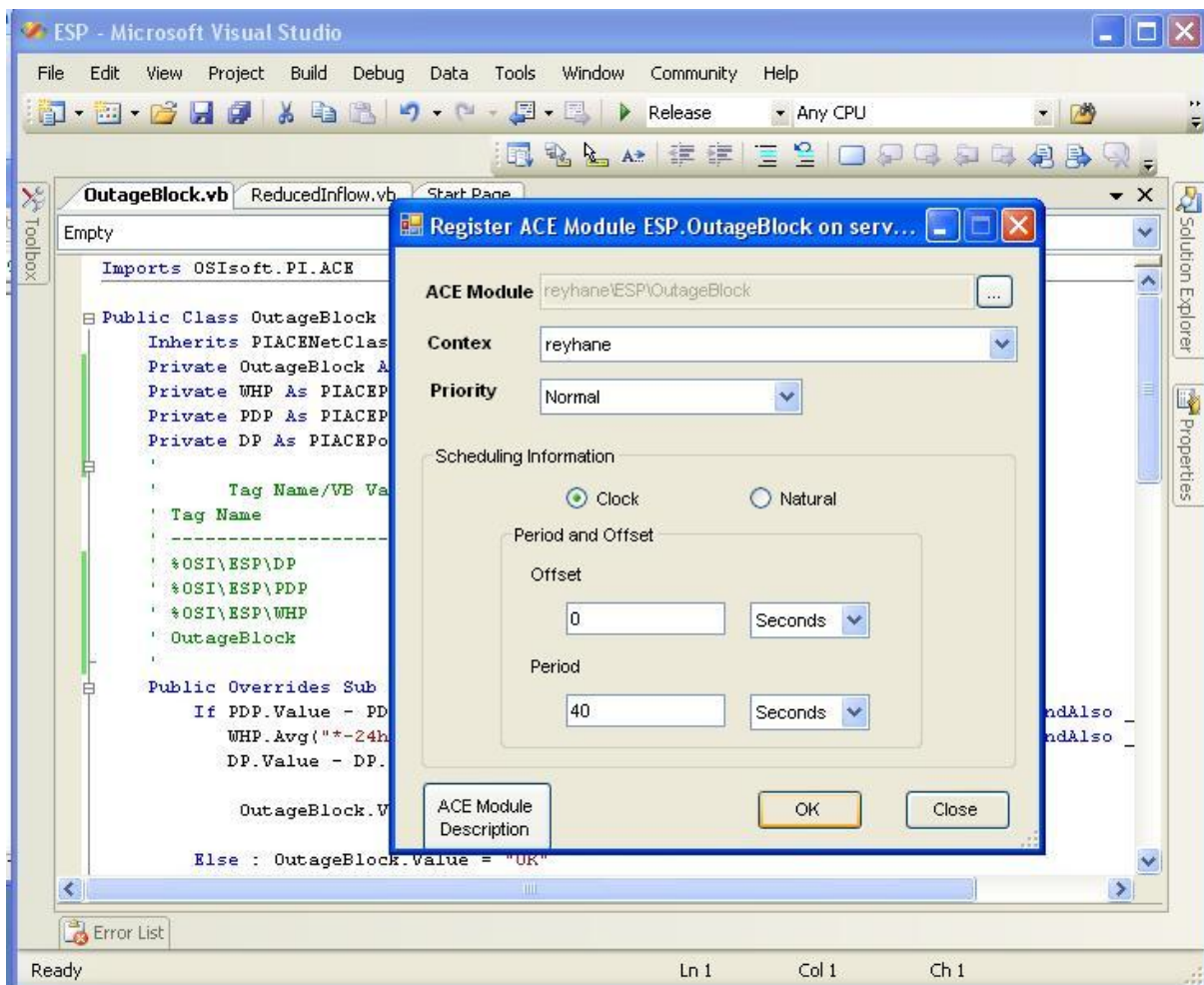


Figure 24. Registering the module

By registering a module it will be added to ACE Manager and the class library will be handled by Scheduler service.

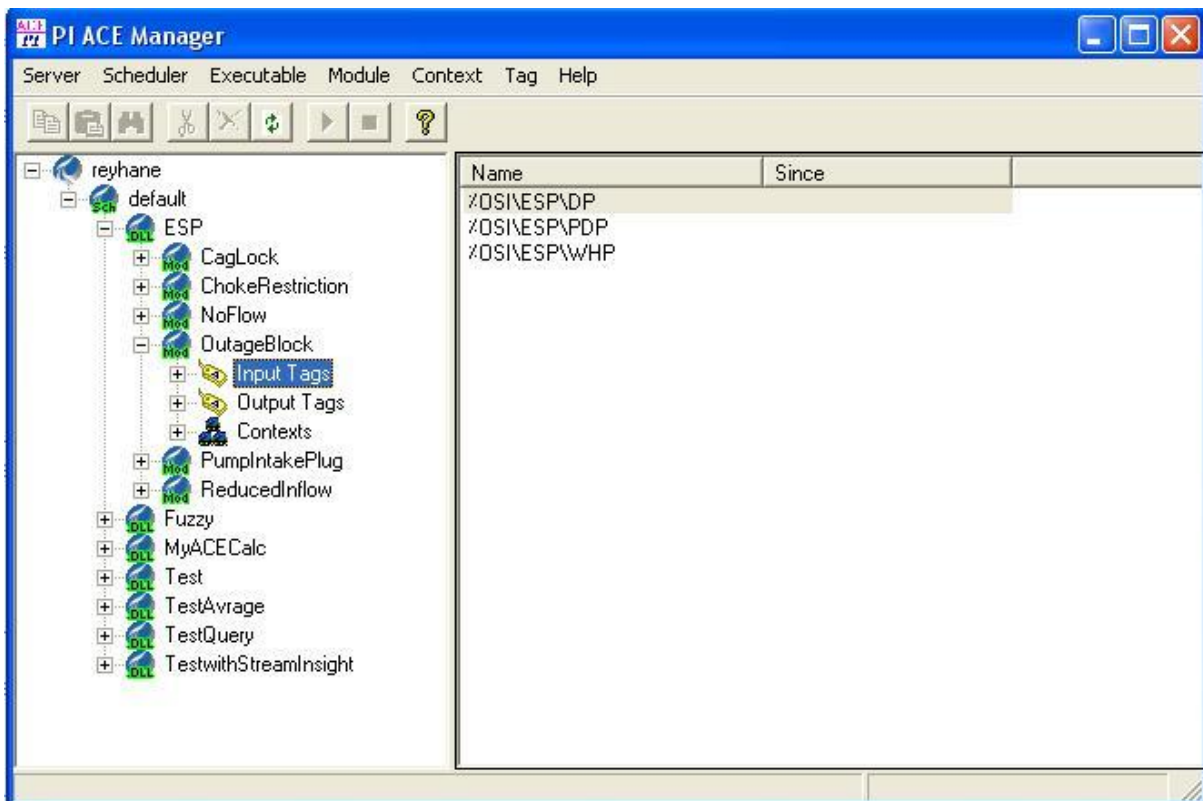


Figure 25. ACE Manager

All the previous steps repeated to make the modules. Below is the code for each module.

```
Imports OSISOFT.PI.ACE
Imports System.Net.Mail

Public Class CagLock
    Inherits PIACENetClassModule
    Private GasLockAlarm As PIACEPoint
    Private PIP As PIACEPoint
    Private PDP As PIACEPoint
    Private AMP As PIACEPoint
    '
    '      Tag Name/VB Variable Name Correspondence Table
    ' Tag Name                                VB Variable Name
    '-----
    ' %OSI\ESF\AMP                            AMP
    ' %OSI\ESF\PDP                            PDP
    ' %OSI\ESF\PIP                            PIP
    ' GasLockAlarm                            GasLockAlarm
    '
    Public Overrides Sub ACECalculations()
        If PDP.Avg("*-24h", "*", 0) - PDP.Value > 0.1 * PDP.Avg("*-24h", "*", 0) AndAlso _
            PIP.Avg("*-24h", "*", 0) - PIP.Value > 0.1 * PIP.Avg("*-24h", "*", 0) AndAlso _
            AMP.Avg("*-24h", "*", 0) - AMP.Value > 0.1 * AMP.Avg("*-24h", "*", 0) Then
            GasLockAlarm.Value = "cond1cond2cond3"

            Else : GasLockAlarm.Value = "OK"
            End If

            'Dim myMsg As MailMessage
            'Dim myMailClient As SmtplibClient
            'myMsg = New MailMessage("sender.unknown.com", "recipient@unknown.com", "Sub: Test
interface", "Body: Problem with test interface")
            'myMailClient = New SmtplibClient("mysmtplibclient.unknown.com")
            'myMailClient.Send(myMsg)
        End Sub

        Protected Overrides Sub InitializePIACEPoints()
            AMP = GetPIACEPoint("AMP")
            PDP = GetPIACEPoint("PDP")
            PIP = GetPIACEPoint("PIP")
            GasLockAlarm = GetPIACEPoint("GasLockAlarm")
        End Sub

        '
        ' User-written module dependent initialization code
        '
        Protected Overrides Sub ModuleDependentInitialization()
        End Sub

        '
        ' User-written module dependent termination code
        '
        Protected Overrides Sub ModuleDependentTermination()
        End Sub
    End Class
```


Choke Restriction Alarm

```
Imports OSIssoft.PI.ACE

Public Class ChokeRestriction
    Inherits PIACENetClassModule
    Private ChokeRestriction As PIACEPoint
    Private WHP As PIACEPoint
    Private FLP As PIACEPoint
    Private AMP As PIACEPoint
    '
    '      Tag Name/VB Variable Name Correspondence Table
    ' Tag Name                               VB Variable Name
    ' -----
    ' %OSI\ESP\AMP                           AMP
    ' %OSI\ESP\FLP                           FLP
    ' %OSI\ESP\WHP                           WHP
    ' ChokeRestriction                       ChokeRestriction
    '
    Public Overrides Sub ACECalculations()

        If WHP.Value - FLP.Value > 1.5 * WHP.Avg("*-24h", "*", 0) - FLP.Avg("*-24h", "*", 0)
        AndAlso AMP.Value - AMP.Avg("*-24h", "*", 0) > 0.1 * AMP.Avg("*-24h", "*", 0) Then

            ChokeRestriction.Value = "Cond1Cond2"

        Else : ChokeRestriction.Value = "OK"
        End If

    End Sub

    Protected Overrides Sub InitializePIACEPoints()
        AMP = GetPIACEPoint("AMP")
        FLP = GetPIACEPoint("FLP")
        WHP = GetPIACEPoint("WHP")
        ChokeRestriction = GetPIACEPoint("ChokeRestriction")
    End Sub

    '
    ' User-written module dependent initialization code
    '
    Protected Overrides Sub ModuleDependentInitialization()
    End Sub

    '
    ' User-written module dependent termination code
    '
    Protected Overrides Sub ModuleDependentTermination()
    End Sub
End Class
```

Pump Intake Plug

```
Imports OSIssoft.PI.ACE

Public Class PumpIntakePlug
    Inherits PIACENetClassModule
    Private PumpIntakePlug As PIACEPoint
    Private PIP As PIACEPoint
    Private PDP As PIACEPoint
    Private DP As PIACEPoint
    '
    '      Tag Name/VB Variable Name Correspondence Table
    ' Tag Name                                VB Variable Name
    '-----
    ' %OSI\ESF\DP                             DP
    ' %OSI\ESF\PDP                            PDP
    ' %OSI\ESF\PIP                            PIP
    ' PumpIntakePlug                          PumpIntakePlug
    '
    Public Overrides Sub ACECalculations()
        If PIP.Value - PIP.Avg("*-24h", "*", 0) > 0.1 * PIP.Avg("*-24h", "*", 0) AndAlso _
            PDP.Avg("*-24h", "*", 0) - PDP.Value > 0.1 * PDP.Avg("*-24h", "*", 0) AndAlso _
            DP.Value - DP.Avg("*-24h", "*", 0) > 0.1 * DP.Avg("*-24h", "*", 0) Then
            PumpIntakePlug.Value = "cond1cond2cond3"
        Else : PumpIntakePlug.Value = "OK"
        End If
    End Sub

    Protected Overrides Sub InitializePIACEPoints()
        DP = GetPIACEPoint("DP")
        PDP = GetPIACEPoint("PDP")
        PIP = GetPIACEPoint("PIP")
        PumpIntakePlug = GetPIACEPoint("PumpIntakePlug")
    End Sub

    '
    ' User-written module dependent initialization code
    '
    Protected Overrides Sub ModuleDependentInitialization()
    End Sub

    '
    ' User-written module dependent termination code
    '
    Protected Overrides Sub ModuleDependentTermination()
    End Sub
End Class
```

Redusec Inflow Alarm

```
Imports OSIssoft.PI.ACE

Public Class ReducedInflow
    Inherits PIACENetClassModule
    Private ReducedInflow As PIACEPoint
    Private WHP As PIACEPoint
    Private PIP As PIACEPoint
    Private PDP As PIACEPoint
    '
    '      Tag Name/VB Variable Name Correspondence Table
    ' Tag Name                               VB Variable Name
    ' -----
    ' %OSI\ESP\PDP                            PDP
    ' %OSI\ESP\PIP                            PIP
    ' %OSI\ESP\WHP                            WHP
    ' ReducedInflow                          ReducedInflow
    '
    Public Overrides Sub ACECalculations()

        If PIP.Avg("*-24h", "*", 0) - PIP.Value > 0.1 * PIP.Avg("*-24h", "*", 0) AndAlso _
            PDP.Avg("*-24h", "*", 0) - PDP.Value > 0.1 * PDP.Avg("*-24h", "*", 0) AndAlso _
            WHP.Avg("*-24h", "*", 0) - WHP.Value > 0.1 * WHP.Avg("*-24h", "*", 0) Then

            ReducedInflow.Value = "CondiCond2Cond3"

        Else : ReducedInflow.Value = "OK"
        End If
    End Sub

    Protected Overrides Sub InitializePIACEPoints()
        PDP = GetPIACEPoint("PDP")
        PIP = GetPIACEPoint("PIP")
        WHP = GetPIACEPoint("WHP")
        ReducedInflow = GetPIACEPoint("ReducedInflow")
    End Sub

    '
    ' User-written module dependent initialization code
    '
    Protected Overrides Sub ModuleDependentInitialization()
    End Sub

    '
    ' User-written module dependent termination code
    '
    Protected Overrides Sub ModuleDependentTermination()
    End Sub
End Class
```

Outage Block

```
Imports OSIssoft.PI.ACE

Public Class OutageBlock
    Inherits PIACENetClassModule
    Private OutageBlock As PIACEPoint
    Private WHP As PIACEPoint
    Private PDP As PIACEPoint
    Private DP As PIACEPoint
    '
    '      Tag Name/VB Variable Name Correspondence Table
    ' Tag Name                                VB Variable Name
    '-----
    ' %OSI\ESF\DP                             DP
    ' %OSI\ESF\PDP                            PDP
    ' %OSI\ESF\WHP                            WHP
    ' OutageBlock                             OutageBlock
    '
    Public Overrides Sub ACECalculations()
        If PDP.Value - PDP.Avg("*-24h", "*", 0) > 0.1 * PDP.Avg("*-24h", "*", 0) AndAlso _
            WHP.Avg("*-24h", "*", 0) - WHP.Value > 0.1 * WHP.Avg("*-24h", "*", 0) AndAlso _
            DP.Value - DP.Avg("*-24h", "*", 0) > 0.1 * DP.Avg("*-24h", "*", 0) Then
            OutageBlock.Value = "cond1cond2cond3"

            Else : OutageBlock.Value = "OK"
            End If

        End Sub

    Protected Overrides Sub InitializePIACEPoints()
        DP = GetPIACEPoint("DP")
        PDP = GetPIACEPoint("PDP")
        WHP = GetPIACEPoint("WHP")
        OutageBlock = GetPIACEPoint("OutageBlock")

    End Sub

    '
    ' User-written module dependent initialization code
    '
    Protected Overrides Sub ModuleDependentInitialization()
    End Sub

    '
    ' User-written module dependent termination code
    '
    Protected Overrides Sub ModuleDependentTermination()
    End Sub
End Class
```

No Flow Alarm

```
Imports OSIssoft.PI.ACE

Public Class NoFlow
    Inherits PIACENetClassModule
    Private NoFlow As PIACEPoint
    Private WHP As PIACEPoint
    Private FLP As PIACEPoint
    '
    '      Tag Name/VB Variable Name Correspondence Table
    ' Tag Name                                VB Variable Name
    '-----
    ' %OSI\ESP\FLP                            FLP
    ' %OSI\ESP\WHP                            WHP
    ' NoFlow                                  NoFlow
    '
    Public Overrides Sub ACECalculations()
        If WHP.Value < FLP.Value Then
            NoFlow.Value = "NoFlow"
        Else
            NoFlow.Value = "OK"
        End If
    End Sub

    Protected Overrides Sub InitializePIACEPoints()
        FLP = GetPIACEPoint("FLP")
        WHP = GetPIACEPoint("WHP")
        NoFlow = GetPIACEPoint("NoFlow")
    End Sub

    '
    ' User-written module dependent initialization code
    '
    Protected Overrides Sub ModuleDependentInitialization()
    End Sub

    '
    ' User-written module dependent termination code
    '
    Protected Overrides Sub ModuleDependentTermination()
    End Sub
End Class
```

Applying Fuzzy Logic

For developing this module same steps has been done like the previous modules. In addition an external library called “DotFuzzy” has been added to module. Below is the code.

```
Imports OSIssoft.PI.ACE
Imports DotFuzzy

Public Class test
    Inherits PIACENetClassModule
    Private FuzzyTest As PIACEPoint
    Private ExpMessage As PIACEPoint
    Private AMP As PIACEPoint
    '
    '      Tag Name/VB Variable Name Correspondence Table
    ' Tag Name                                VB Variable Name
    '-----
    ' %OSI\ESP\AMP                            AMP
    ' ExpMessage                               ExpMessage
    ' FuzzyTest                                FuzzyTest
    '
    Public Overrides Sub ACECalculations()
        Dim MotorCurrent As LinguisticVariable
        Dim MotorFrequency As LinguisticVariable
        Dim FuzzyEngine As FuzzyEngine
        MotorCurrent = New LinguisticVariable("MotorCurrent")
        MotorFrequency = New LinguisticVariable("MotorFrequency")
        FuzzyEngine = New FuzzyEngine

    Try
        'Defining Membership Function for Motor Current
        MotorCurrent.MembershipFunctionCollection.Add(New MembershipFunction("Normal", 0, 170, 180, 186))
        MotorCurrent.MembershipFunctionCollection.Add(New MembershipFunction("High", 184, 186, 193, 196))
        MotorCurrent.MembershipFunctionCollection.Add(New MembershipFunction("HighHigh", 193, 196, 200, 200))

        'Defining Membership Function for MotorFrequency
        MotorFrequency.MembershipFunctionCollection.Add(New MembershipFunction("Dcrease", 35, 40, 48, 50))
        MotorFrequency.MembershipFunctionCollection.Add(New MembershipFunction("Normal", 49, 51, 55, 57))
        MotorFrequency.MembershipFunctionCollection.Add(New MembershipFunction("Dropped", 0, 0, 0, 1))

        'Adding variables to engine
        FuzzyEngine.LinguisticVariableCollection.Add(MotorCurrent)
        FuzzyEngine.LinguisticVariableCollection.Add(MotorFrequency)
        FuzzyEngine.Consequent = "MotorFrequency"

        'Declaring fuzzy rules
        FuzzyEngine.FuzzyRuleCollection.Add(New FuzzyRule("IF (MotorCurrent IS HighHigh) THEN
                                                         MotorFrequency IS Dropped"))
        FuzzyEngine.FuzzyRuleCollection.Add(New FuzzyRule("IF (MotorCurrent IS High) THEN
                                                         MotorFrequency IS Dcrease"))
        FuzzyEngine.FuzzyRuleCollection.Add(New FuzzyRule("IF (MotorCurrent IS Normal) THEN
                                                         MotorFrequency IS Normal"))

        'Getting the Motor Current Value
        MotorCurrent.InputValue = AMP.Value

        MotorCurrent.Fuzzify("Normal")
        MotorCurrent.Fuzzify("High")
        MotorCurrent.Fuzzify("HighHigh")
        FuzzyTest.Value = FuzzyEngine.Defuzzify()

    Catch Ex As Exception

        ExpMessage.Value = Ex.Message.ToString()

    End Try

End Sub

    Protected Overrides Sub InitializePIACEPoints()
        AMP = GetPIACEPoint("AMP")
    End Sub
End Class
```

```
        ExpMessage = GetPIACEPoint("ExpMessage")
        FuzzyTest = GetPIACEPoint("FuzzyTest")
    End Sub

    '
    ' User-written module dependent initialization code
    '
    Protected Overrides Sub ModuleDependentInitialization()
    End Sub

    '
    ' User-written module dependent termination code
    '
    Protected Overrides Sub ModuleDependentTermination()
    End Sub
End Class
```

Alarm System Using StreamInsight

This section contains three different steps when developing the system, below is the code which developed to generate Reduce Inflow alarm.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.ServiceModel;
using Microsoft.ComplexEventProcessing;
using Microsoft.ComplexEventProcessing.ManagementService;
using OSIssoft.ComplexEventProcessing.Adapters;
using Microsoft.ComplexEventProcessing.Linq;
using OSIssoft.ComplexEventProcessing.DataTypes;

namespace OSIssoft.ComplexEventProcessing.Samples
{
    public class PIDataSingle
    {
        /// <summary>
        /// Tag name of the tag to write to
        /// </summary>
        ///
        public String Path { get; set; }

        /// <summary>
        /// Value to be written to the tag
        /// </summary>
        public Double Value { get; set; }

        /// <summary>
        /// Status to be written to the tag
        /// </summary>
        public Int32 Status { get; set; }
    }

    class ESPAlarm
    {
        /// <summary>
        /// Verifies that the command line contains enough information to
        /// launch the code sample, then launches it.
        /// </summary>
        /// <param name="args">Command line arguments. First should be the name
        /// of the StreamInsight instance, then optionally the name of the
        /// PIServer.</param>
        static void Main(string[] args)
        {
            string instanceName, piServer = "reyhane";
            try
            {
                switch (args.Length)
                {

                    case 2:
                        piServer = args[1];
                        goto case 1;

                    case 1:
                        instanceName = args[0];
                        break;
                }
            }
        }
    }
}
```



```

default:
Console.WriteLine("Usage: AlarmWithFixedLimit instanceName [piServerName]");
Console.WriteLine("instanceName is the name of a local StreamInsight
instance");
Console.WriteLine("piServerName is the name of the PIServer (defaults to
'localhost')");
return;
}
Run(instanceName, piServer);
}
catch (Exception e)
{
Console.WriteLine(e.Message + "\n" + e.StackTrace);
}
Console.ReadLine();
}

/// <summary>
/// Runs the sample.
/// </summary>
/// <param name="instanceName">The name of the StreamInsight
instance.</param>
/// <param name="PIServer">The name of the PIServer to use.</param>
static void Run(string instanceName, string PIServer)
{
Server server = Server.Create(instanceName);
using (server)
{
Query query = BuildQuery(server, PIServer);

ServiceHost h = new
ServiceHost(query.Application.Server.CreateManagementService);

h.AddServiceEndpoint(typeof(IManagementService),
new WSHttpBinding(SecurityMode.Message),
"http://localhost/OSISoft/PIEventProcessingHost");
h.Open();

// Run Query
query.Start();

// Wait
Console.ReadLine();

// Stop Query
query.Stop();
}
}

/// <summary>
/// Returns a StreamInsight query, ready to run.
/// </summary>
/// <param name="numStreams">The number of input PI point data streams
to include in the query.</param>
/// <returns>The alarm query.</returns>
///
public static Query BuildQuery(Server server, string PIServer)
{
Application alarmApp = server.CreateApplication("PI Adapters
Samples");

// Input Stream Configuration
SnapshotInputConfig config = new SnapshotInputConfig();

```

```
config.Server = PIServer;
config.PointsQuery = new string[] { "AlarmTest.Input.Float32.1",
    "AlarmTest.Input.Float32.2" , "AlarmTest.Input.Float32.3" };

CepStream<PIEvent<Double>> dataStream;

// Process data input Stream
dataStream = CepStream<PIEvent<Double>>.Create(
    "PI Input Main Stream",
    typeof(SnapshotInputFactory),
    config,
    EventShape.Point,
    AdvanceTimeSettings.IncreasingStartTime
);

// compute the 24h average for each Id separately
var avgStream = from e in dataStream
                group e by e.Id into g
                from win in g
                .AlterEventDuration(e => TimeSpan.FromHours(24))
                .SnapshotWindow(SnapshotWindowOutputPolicy.Clip)
                select new
                { Id = g.Key, Avg = win.Avg(e => e.Value) };

// for each Id, compare the average with the real-time value
var check = from a in avgStream
            join b in dataStream
            on a.Id equals b.Id
            select new{ a.Id, Check = (a.Avg - b.Value) > a.Avg * 0.1 };

// turn the result into a continuous signal for each Id:
var checkSignal = check.AlterEventDuration(e => TimeSpan.MaxValue)
    .ClipEventDuration(check, (e1, e2) => e1.Id == e2.Id);

// compute the periods where condition is met over all Ids:
var result = from conditions in (from win in checkSignal
                                .Where(e => e.Check)
                                .SnapshotWindow(SnapshotWindowOutputPolicy.Clip)
                                select new{
                                    Count = win.Count()
                                })
            where (conditions.Count >2)
            select new PIDataSingle()
            {
                Path = "SI.ReducedInflow",
                Status = 0,
                Value = conditions.Count
            };

var outputFactory = typeof(SnapshotOutputFactory);
var outputConfig = new SnapshotOutputConfig();
```

```

        outputConfig.Server = PIServer;

        var query = result.ToQuery(alarmApp, "alarm sample", string.Empty,
            outputFactory,
            outputConfig,
            EventShape.Point,
            StreamEventOrder.FullyOrdered);

        return query;
    }
}

```

Since the previous query did not have the same result following query tested in StreamInsight to calculate the average over last 24 hours.

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.ServiceModel;
using Microsoft.ComplexEventProcessing.ManagementService;
using Microsoft.ComplexEventProcessing;
using OSIssoft.ComplexEventProcessing.Adapters;
using Microsoft.ComplexEventProcessing.Linq;
using OSIssoft.ComplexEventProcessing.DataTypes;

namespace OSIssoft.ComplexEventProcessing.Samples
{
    /// <summary>
    /// Declaration of a simple output stream payload that can be understood by
    the SnapshotOutputAdapter
    /// </summary>
    public class PIDataSingle
    {
        /// <summary>
        /// Tag name of the tag to write to
        /// </summary>
        ///
        public String Path { get; set; }

        /// <summary>
        /// Value to be written to the tag
        /// </summary>
        public Double Value { get; set; }

        /// <summary>
        /// Status to be written to the tag
        /// </summary>
        public Int32 Status { get; set; }
    }

    /// <summary>
    /// Sample code that illustrates a very simple query that reacts to values
    being written to one tag ("sinusoid") and
    /// writes identical values to another tag ("MovingTarget")
    /// </summary>
    public class Passthrough
    {

```

```
// The name of the StreamInsight instance, determined when
StreamInsight was installed
const string instanceName = "reyhane";
// The name of the PI Server to use for input. A null value will use
the local PI Server
const string inputPIServer = "reyhane";
// The name of the tag to read input values from
const string inputTag = "AlarmTest.Input.Float32.1";
// The name of the PI Server to use for output. A null value will use
the local PI Server
const string outputPIServer = "reyhane";

static void Main(string[] args)
{
    using (Server cepServer = Server.Create(instanceName))
    {
        Application cepApplication =
            cepServer.CreateApplication("Average24hours");

        SnapshotInputConfig inputConfig = new SnapshotInputConfig();
        inputConfig.Server = inputPIServer;
        inputConfig.PointsQuery = new string[] { inputTag };

        SnapshotOutputConfig outputConfig = new SnapshotOutputConfig();
        outputConfig.Server = outputPIServer;

        var input = CepStream<PIEvent<Double>>.Create("PI Input tream",
                                                    typeof(SnapshotInputFactory),
                                                    inputConfig,
                                                    EventShape.Point,
                                                    AdvanceTimeSettings.StrictlyIncreasingStartTime);

        var result = from win in input
                    .AlterEventDuration(e => TimeSpan.FromMinutes(24))
                    .SnapshotWindow(SnapshotWindowOutputPolicy.Clip)
                    select new PIDataSingle()
                    {
                        Path = "AvrgSI",
                        Status = 0,
                        Value = win.Avg(e => e.Value)
                    };

        var outputToPI = result.ToQuery(cepApplication,
                                        "simpleOutput",
                                        "A simple Output",
                                        typeof(SnapshotOutputFactory),
                                        outputConfig,
                                        EventShape.Point,
                                        StreamEventOrder.FullyOrdered);

        outputToPI.Start();
        Console.ReadLine();
        outputToPI.Stop();
    }
}
}
```

To compare the result with ACE the same logic implemented using following ACE module:

```
Imports OSIssoft.PI.ACE

Public Class avrg24

    Inherits PIACENetClassModule

    Private AvrgACE As PIACEPoint

    Private AlarmTest_Input_Float32_1 As PIACEPoint
    '
    ' Tag Name/VB Variable Name Correspondence Table
    ' Tag Name VB Variable Name
    '-----
    ' AlarmTest.Input.Float32.1 AlarmTest_Input_Float32_1
    ' AvrgACE AvrgACE
    '
    Public Overrides Sub ACECalculations()

        AvrgACE.Value = AlarmTest_Input_Float32_1.Avg("*-24h", "*", 0)

    End Sub

    Protected Overrides Sub InitializePIACEPoints()

        AlarmTest_Input_Float32_1 = GetPIACEPoint("AlarmTest_Input_Float32_1")

        AvrgACE = GetPIACEPoint("AvrgACE")

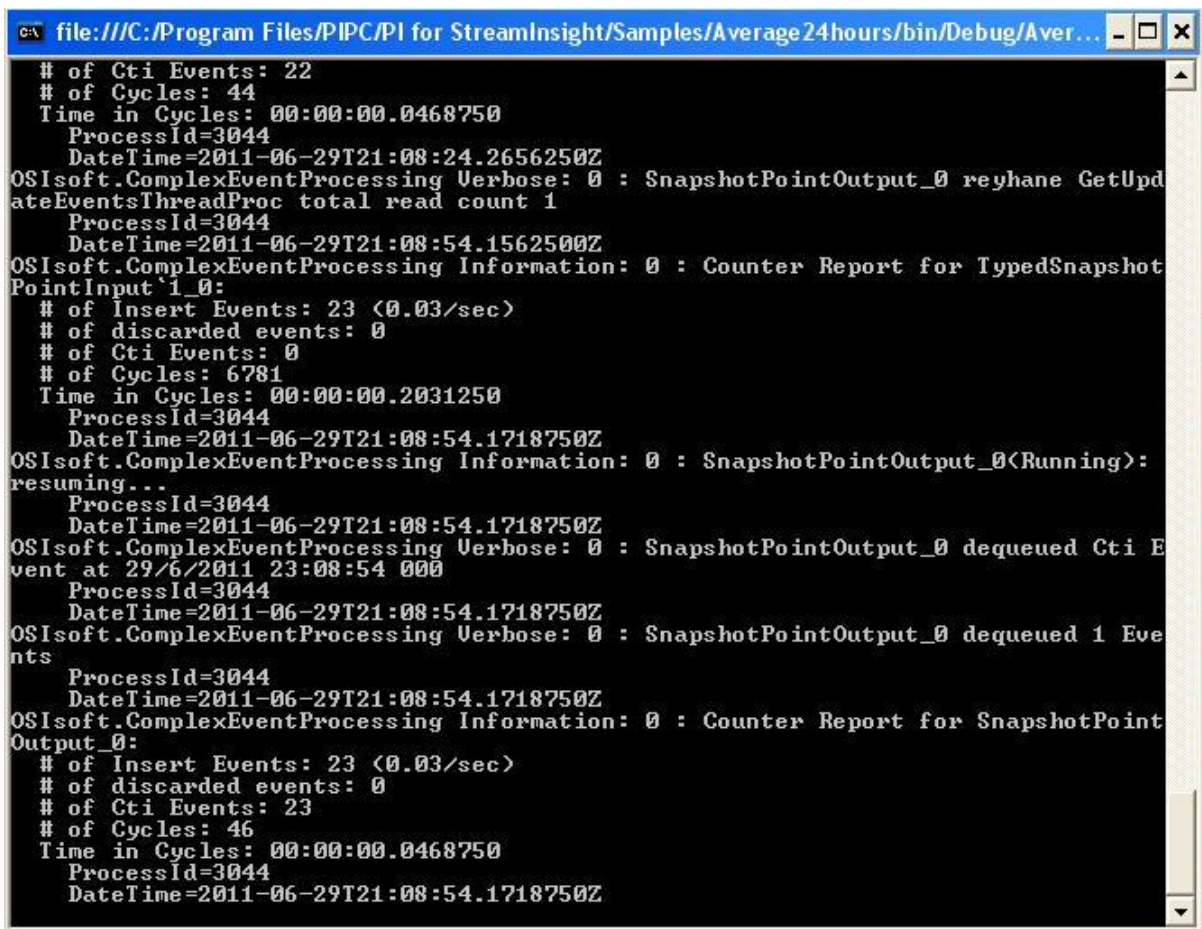
    End Sub

    '
    ' User-written module dependent initialization code
    '
    Protected Overrides Sub ModuleDependentInitialization()
    End Sub

    '
    ' User-written module dependent termination code
    '
    Protected Overrides Sub ModuleDependentTermination()
    End Sub

End Class
```

The application output illustrated in Figure 26 and the results are depicted in Figure 26:



```
file:///C:/Program Files/PIPC/PI for StreamInsight/Samples/Average 24hours/bin/Debug/Aver...
# of Cti Events: 22
# of Cycles: 44
Time in Cycles: 00:00:00.0468750
ProcessId=3044
DateTime=2011-06-29T21:08:24.2656250Z
OSISoft.ComplexEventProcessing Verbose: 0 : SnapshotPointOutput_0 reyhane GetUpdateEventsThreadProc total read count 1
ProcessId=3044
DateTime=2011-06-29T21:08:54.1562500Z
OSISoft.ComplexEventProcessing Information: 0 : Counter Report for TypedSnapshotPointInput`1_0:
# of Insert Events: 23 <0.03/sec>
# of discarded events: 0
# of Cti Events: 0
# of Cycles: 6781
Time in Cycles: 00:00:00.2031250
ProcessId=3044
DateTime=2011-06-29T21:08:54.1718750Z
OSISoft.ComplexEventProcessing Information: 0 : SnapshotPointOutput_0(Running): resuming...
ProcessId=3044
DateTime=2011-06-29T21:08:54.1718750Z
OSISoft.ComplexEventProcessing Verbose: 0 : SnapshotPointOutput_0 dequeued Cti Event at 29/6/2011 23:08:54 000
ProcessId=3044
DateTime=2011-06-29T21:08:54.1718750Z
OSISoft.ComplexEventProcessing Verbose: 0 : SnapshotPointOutput_0 dequeued 1 Events
ProcessId=3044
DateTime=2011-06-29T21:08:54.1718750Z
OSISoft.ComplexEventProcessing Information: 0 : Counter Report for SnapshotPointOutput_0:
# of Insert Events: 23 <0.03/sec>
# of discarded events: 0
# of Cti Events: 23
# of Cycles: 46
Time in Cycles: 00:00:00.0468750
ProcessId=3044
DateTime=2011-06-29T21:08:54.1718750Z
```

Figure 26. StreamInsight application output

The figure displays two screenshots of the Archive Editor - PI System Management Tools interface, showing the results of average functions in ACE and StreamInsight.

Left Window (AvgACE):

- Server: reyhane
- Tagname: AvgACE
- Event Count: 240
- Retrieved: 240
- Start Time: ~2h
- Merge Type: Replace Duplicates
- Boundary Type: Inside
- Show Filtered: Remove Filtered Values
- Filter Expression: (empty)

Value	Event Time	Questionable	Annotated
166.3855	6/28/2011 6:05:24 PM	<input type="checkbox"/>	<input type="checkbox"/>
167.0119	6/28/2011 6:05:54 PM	<input type="checkbox"/>	<input type="checkbox"/>
167.0381	6/28/2011 6:06:24 PM	<input type="checkbox"/>	<input type="checkbox"/>
167.0636	6/28/2011 6:06:54 PM	<input type="checkbox"/>	<input type="checkbox"/>
167.1128	6/28/2011 6:07:54 PM	<input type="checkbox"/>	<input type="checkbox"/>
167.1383	6/28/2011 6:08:24 PM	<input type="checkbox"/>	<input type="checkbox"/>
167.1648	6/28/2011 6:08:54 PM	<input type="checkbox"/>	<input type="checkbox"/>
167.1911	6/28/2011 6:09:24 PM	<input type="checkbox"/>	<input type="checkbox"/>
167.2188	6/28/2011 6:10:54 PM	<input type="checkbox"/>	<input type="checkbox"/>
167.2419	6/28/2011 6:10:24 PM	<input type="checkbox"/>	<input type="checkbox"/>
167.267	6/28/2011 6:10:54 PM	<input type="checkbox"/>	<input type="checkbox"/>
167.2923	6/28/2011 6:11:24 PM	<input type="checkbox"/>	<input type="checkbox"/>
167.3176	6/28/2011 6:11:54 PM	<input type="checkbox"/>	<input type="checkbox"/>
167.343	6/28/2011 6:12:24 PM	<input type="checkbox"/>	<input type="checkbox"/>

Right Window (AvgSI):

- Server: rcyhan0
- Tagname: AvgSI
- Event Count: 241
- Retrieved: 241
- Start Time: ~2h
- Merge Type: Replace Duplicates
- Boundary Type: Inside
- Show Filtered: Remove Filtered Values
- Filter Expression: (empty)

Value	Event Time	Questionable	Annotated	Substitute
147.0725	6/23/2011 6:05:24 PM	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
136.3825	6/23/2011 6:05:54 PM	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
136.3825	6/23/2011 6:06:24 PM	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
137.0000	6/23/2011 6:06:54 PM	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
136.9761	6/23/2011 6:07:24 PM	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
137.0069	6/23/2011 6:07:54 PM	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
137.1561	6/23/2011 6:08:24 PM	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
137.3752	6/23/2011 6:08:54 PM	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
137.592	6/23/2011 6:09:24 PM	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
137.7817	6/23/2011 6:09:54 PM	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
138.0344	6/23/2011 6:10:24 PM	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
138.295	6/23/2011 6:10:54 PM	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
138.5302	6/23/2011 6:11:24 PM	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
138.7900	6/23/2011 6:11:54 PM	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
138.1165	6/23/2011 6:12:24 PM	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Figure 27. Result of average functions in ACE and StreamInsight