AGH University of Science and Technology

Faculty of Electrical Engineering, Automatics, Computer Science and Electronics

Master's Thesis

Control Systems Integration using OPC Standard

Author:
Marcin Bajer

Tutor:
prof. dr hab. inż. Wojciech Grega

Antwerp 2008

Abstract

A subject of the master thesis is a usage of OPC standard for dynamic process control purpose. The process is both simulated and realized in real time. The aim of this work will be to specify OPC server/client configuration methods for different control parameters. Additionally, various OPC specifications and their implementation in modern manufacturing process will be described.

I wish to express my gratitude:

To my mentor and this thesis supervisor - prof. Wojciech Grega for his patience, guidance and valuable advices

To ing. Dominique Daens for his openness and support during my presence in Antwerp

Most of all I wish to convey my deepest gratitude to my fiancé – Aniu thank you for your patience throughout last year. Without your full support, it would have not been possible to accomplish this work.

Table of contents

1. Introduction

1.1. Purpose and Motivations

Nowadays, software has become essential part of modern automation and control systems. Dedicated control systems have been replaced by more universal ones. Industrial PCs are currently widely used for data acquisition, visualization, process control and other automation purposes. Analog and mechanical controllers were almost completely replaced by digital ones based on microprocessors. Different elements of control system are not located in a one central location but are distributed throughout the system. Over the last few years the number of sensor systems is growing very rapidly. They are used for delivering important information for process control purpose, as well as to improve process performance.

The ability to easily integrate information from control systems and plant floor measurement with production and maintenance management is a critical issue. Manufacturers need to access data from the plant floor and integrate it into their existing business systems. Manufacturers must be also able to use off-the-shelf software like SCADA packages, databases systems, process modeling tools and others. It is very difficult or if not impossible to share data between various devices and software manufactured by different vendors without a common industrial standard to facilitate interoperability. The key is open and effective communication architecture concentrating on data access, and not the types of data. The OPC (OLE for Process Control) protocol was made as a solution which fulfills those requirements.

The goal of this paper has been to describe different possibilities to implement OPC technology in modern manufacturing process. The main focus has been the possibility of usage OPC in real-time dynamic process control.

1.2. Structure of the Thesis

This diploma thesis has been prepared and written during my residence in Antwerp, Belgium while I was studying at Karel de Grote-Hogeschool as a part of a Socrates/Erasmus student's exchange programme. The short outline of this thesis is described here.

In the beginning of this thesis, areas of control systems integration have been presented. The purpose of using OPC at the different factory floor has been described in detail.

Chapter three reviews the various OPC Standard releases. This chapter also introduces history of OPC Foundation. State of art before the OPC Standard appearance is also being exposed.

Chapter four provides information for software developers about implementing OPC in their applications. Additionally, possibility of using OPC with Windows CE has been considered.

Fifth chapter describes methods for control systems developing. The fast prototyping idea is described. Possibilities of use the OPC Standard for simulation and validation of manufacturing systems has been also considered. Matlab / Simulink, is described as a major tool for this purpose.

Chapter six describes an experimental part of this thesis. The main idea with that chapter is to desire a model of control system with OPC in control loop and verify accuracy of this model. In this chapter reliability, robustness and performance issues have been also considered. Additionally the impact of network traffic for control quality is measured. At the end, connecting OPC with process model running on QNX real time operating system has been described. A few experiments with controlling this model with real PLC have been performed.

The conclusions of this thesis are summarized in the discussion performed in chapter seven.

2. Control Systems Integration

2.1. Areas of control systems integration

Usually industrial plant consists of combination of many subsystems cooperating with each other so that the overall system works as it was demanded. Generally, the main purpose with control system integration is to bring together separate devices, software and technologies into a whole and ensure that the complete system is able to deliver the expected functionalities.

In this chapter the structure of a modern factory has been described, the special focus is placed on investigation of possibilities to use the OPC protocol to integrate the data flow. To describe the flow of data in the factory the so-called pyramid of automation has been used. The number of required components, the information distributed within the different levels of a system can be portrayed in the form of a pyramid *[Figure 2.1]*.

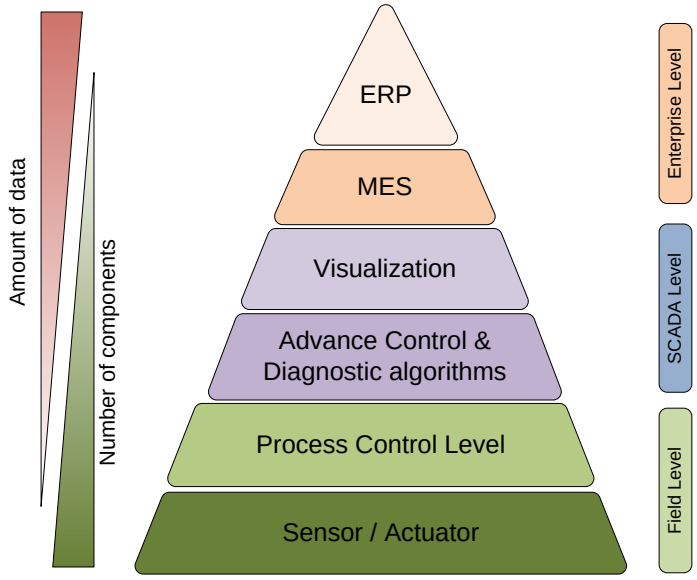


Figure 2.1. An Automation Pyramid *[2.1]*

As it can be seen, together with narrowing of the pyramid the amount of information to be transmitted is growing, but you must also remember that complexity of devices at a lower level is smaller and the number of devices increases. Generally, the following statement applies: the higher level is, the greater is the amount of information, but the speed of transmission is smaller and real-time requirements are weaker.

Due to the different data flow requirements, different fieldbuses must be used on different factory floors. Bus systems should provide the means for communication both within the specific floor and between the different factory floors. The main task for system integrators is to create this interconnection.

In today's word the market for control system integration is growing rapidly. The role of system integration is becoming more and more important because it provides big savings in factory day-to-day maintenance. Integration process is being performed both on the systems under construction, as well as on systems that are already deployed, but still requires some integration to be done. The aim is to achieve integration not only on the plant level, but to have a true enterprise level integration and tie the process control devices with the business systems of the enterprise.

Since 1970s, when the process of integrating industrial devices into larger installations has been started many things changed. At the first stage manufacturers and distributors of field instruments have been willing to do the odd integration projects. With the popularization of Programmable Logic Controllers (PLCs) in 1980s companies began to be asked to do the integration on a regular basis and it has become evident that the development of common standards was needed.

As personal computers grew in popularity and power, from the late 1980s, software-based Human-Machine-Interfaces (HMIs) were developed to permit supervisory control over sensors and control systems. Since then, integrators were looking for standardization not only in terms of hardware but also software.

In next step, PLCs with internal fieldbus interface and efficient analog to digital and digital to analog converters began to replace the previous generation of analog long-range data acquisition and telemetry systems. At this stage computers were used not only for visualization, but also for control purpose. Supervisory Control and Data Acquisition (SCADA) applications soon began to replace the backward architectures of earlier systems. Integrators wanted to use solutions provided by many manufacturers instead of having to choose a control system vendor and purchase all or most of the controls and sensors, as well as the control system itself from them. Architecture based on a PC has

allowed spreading off-the-shelf SCADA software. Many software companies found their market in producing software for industrial purpose. This was a trigger to begin the open systems movement.

The integration of control systems requires a broad set of skills and knowledge in various fields. Industrial systems integrators generally have to be familiar both with hardware and software. In this case, broad knowledge is required rather than a depth of knowledge. Besides of software and hardware engineering, interface protocols and general problem solving skills, knowledge about the industrial process is needed.

There are many possible areas of integration. Basically, it is possible to distinguish between horizontal and vertical integration.

Horizontal integration is a process which involves tying all devices on a factory floor together. Horizontal integration eliminates isolated cells of activity by merging the entire manufacturing process into a single coordinated system. For example, if a production in a plant is being performed on a production line, all machines on the line are connected together and exchange each other information about production flow *[Figure 2.2]*. Also in case of Enterprise Level, horizontal integration term means that all information systems on Enterprise level are linked together and exchange information for better production management and planning.

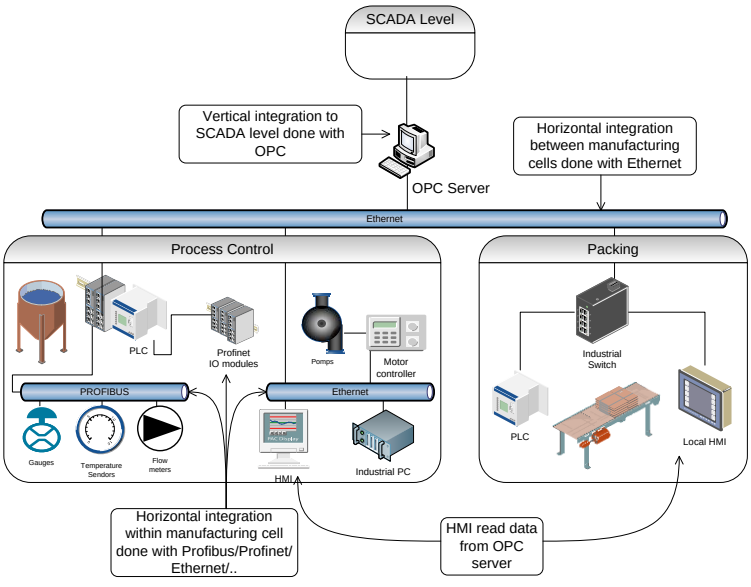


Figure 2.2. Example of Trends in control system integration.

Vertical integration refers to the need to integrate systems implemented at different administrative level of an organization. In case of production plant, it is crucial to provide communication between different factory floors. Vertical integration allows the transparent data transfer and execution of work instructions from the simplest sensor on the plant floor to the company's intranet.

The main problem associated with vertical integration is that it joins together different levels of the factory - that means usually gateway must be used to interconnect different fieldbuses. Even if both levels use the same communication layer firewall must be used to ensure the safety of lower levels.

In the case of vertical integration, it is important to provide adequate information filtering, so that left only those that are important. The main goals of vertical integration are help in managing, improving efficiency and minimizing downtime.

A combination of those two types of integration is called *star integration* or also known as *spaghetti integration [2.2].* It is a process of integration of the systems where each system is interconnected to each of the remaining subsystems. This architecture was typical for old systems. For small systems it can be quickly implemented, but time and costs of integration increase exponentially when adding additional subsystems. Before OPC has been released system integrators were forced to use this kind of integration.

In the further part of this chapter, the details of the integration of devices at different hierarchical levels of the plant will be described. A special focus was placed on the usage of OPC for integration.

2.2. Integration at the field level

The Field level is the lowest level of the factory. All actuators, sensors and measuring devices are located at the field level. Nowadays, the leading trend is to provide distributed control logic. Devices are more and more sophisticated and measured signal is being processed by sensors. Usually, measuring devices are also equipped with LCD panel to communicate with user. It is possible to configure safety levels, measured units and data analyze. It is recommended all field devices use digital data communication to enable data

transfer between distributed devices. Standard Ethernet and TCP/IP are use mainly for communication between the higher factory's levels, but due to increasing complexity of newest sensors, an industrial version of Ethernet and TCP/IP is becoming more popular also at the field level. Please note that the implementation of Ethernet in devices at the field level is usually limited to basic functionalities. It does not provide solutions for security, dynamic IP address configuration (DHCP) or device management (SNMP), but usually OPC server is available for those devices.

At the field level there is a wide number of other available fieldbus standards such as Profibus, ModBus, DeviceNet, CAN, AS-Interface. They are still very popular especially among measuring devices and actuators. The horizontal integration at the field level is mostly being done with them. The biggest problem with those protocols is that they should have Master – Slave architecture. It means only one device can read and write information form all other devices. The consequence of this is a fact the Master on the bus must be both process controller and gateway to access data to/from the actuators/sensors.

The Ethernet interface is the most widespread among complex devices which only capture data for visualization or diagnostic purpose (for example vibrations measurements for forecasting the life-time of bearings). The popularity of fieldbuses other than Ethernet makes necessary to use gateway to communicate with higher levels of the factory. The PLC devices are usually equipped with Ethernet interface to enable vertical integration with SCADA level and other interfaces for communicating with low level devices (sensors, actuators). In most of the solutions PLC devices are both low level process control devices and gateways for higher levels of the factory structure. All data which should be transferred to SCADA need to be specified while PLC is being programmed. Nowadays, almost all PLC devices have a dedicated OPC server.

2.3. Integration at the SCADA level

The SCADA term stands for ***S**upervisory **C**ontrol **A**nd **D**ata **A**cquisition*. It generally refers to a set of applications for computer based controlling and monitoring a process.

The OPC was mostly created to enable interoperability at the SCADA level. Currently, OPC is a widely accepted industrial communication standard that enables the exchange of data without any proprietary restrictions between multi-vendor devices and control software (horizontal integration) as well as between different software applications (vertical integration). Currently, most of SCADA systems can be configured to be the OPC Server for higher levels of the factory.

A SCADA System usually consists of the following subsystems:

- A Human-Machine Interface (HMI) – those are all means of communication between the system and the human operator. Not only pure data are presented to user, information from the field level is being processed. Data is compiled and formatted in such a way that the operators using the HMI can make supervisory decisions to control the process. Alarms are displayed to inform about dangerous system state and events to inform about current actions (OPC A&E).
- Database – it is important to save the past process data. It can be helpful for tracing the faults in the system, logistic information and future optimization of the process. HMI is connected to the database to provide the trending information (OPC HDA).
- Advanced control and diagnostic systems – even now the performance of most PLC devices is too small to perform advance control and diagnostic algorithms. In most cases SCADA level is a higher level of control, the basic control is being executed on a PLC. For example, PID algorithm is performed on the PLC and PID settings are optimized by the advance control algorithm at the SCADA level. Sometimes it is not possible to separate a complex control algorithm into two parts and control on the field level is required. In those cases it is necessary to use a dedicated device or more sophisticated PLC. Some PLCs have built-in a FPGA matrix (ABB AC800PEC) or a signal processor to meet the requirements of a process control.

There are many SCADA packages available on the market. They have many features in common. Each major supplier of industrial devices seeks to provide a full set of SCADA applications.

The border between the Field Level and SCADA Level devices is not as sharp as it was a few years ago. In the past, it was not possible to use standard PC computers at the field level because of the harsh industrial conditions. Nowadays, many things have changed. Modern PLC's performance is similar to old PC computers. The difference between old and new PLCs is so big that currently suppliers of automation systems define a new class of devices - Programmable Automation Controllers (PAC). Also, PCs went down to the field level. Industrial Personal Computers (IPC) are based on these same components, but offer features different from regular PCs in terms of reliability, compatibility, expansion modules and long-term supply.

Even, with those problems, clearly defining boundaries between the particular levels of the factory is crucial. The separation should be done by adding gateways and firewalls into system. This separation is important for two reasons. First, it provides the filtration of the data transferred between the levels, second it ensures the safety of the critical parts of factory infrastructure from a cyber attacks. If all traffic goes through a single connection point a firewall can be effectively monitored and secured. It is very important to use a secured firewall between the corporate network and the Internet. Enterprise level must be also separated from the SCADA level. It is recommended that the control system is also protected from attack within the SCADA network.

One of the most important issues for system integrators is to enable remote access to company's intranet. This is especially important in cases when the factory is placed in location where the access is limited (harsh weather conditions, places of military conflicts). Virtual Private Network (VPN) is a secured way for remote connecting into SCADA networks *[2.3], [2.4]*. VPN is a network created with using the public Internet network. A secured channel that allows exchange of information is established between the client and the network *[Figure 2.3]*. From the client site this connection is transparent – it means for client's software there is no difference between real connection to the network and virtual network. Transparency is important for configuring OPC connections, OPC clients will operate in the same way as if the computer was in the same network. Typically a VPN server is installed either as part of the

firewall or as a separate machine to which external users will authenticate before gaining access to the SCADA networks.

Separating the networks is the first step towards securing SCADA systems. It must be also assured that the access to the system will have only authorized personnel. Role-based access should be implemented into the SCADA software.

Complementary to those protection mechanisms writing policy should be employed. In case of OPC it is possible to define which variables can be modified by OPC clients.

Many of PLC devices have two Ethernet interfaces, one can be intended only for communication within the field level, second can be designed for vertical integration with SCADA level. It is the most common way of separating the levels in small and medium control systems.

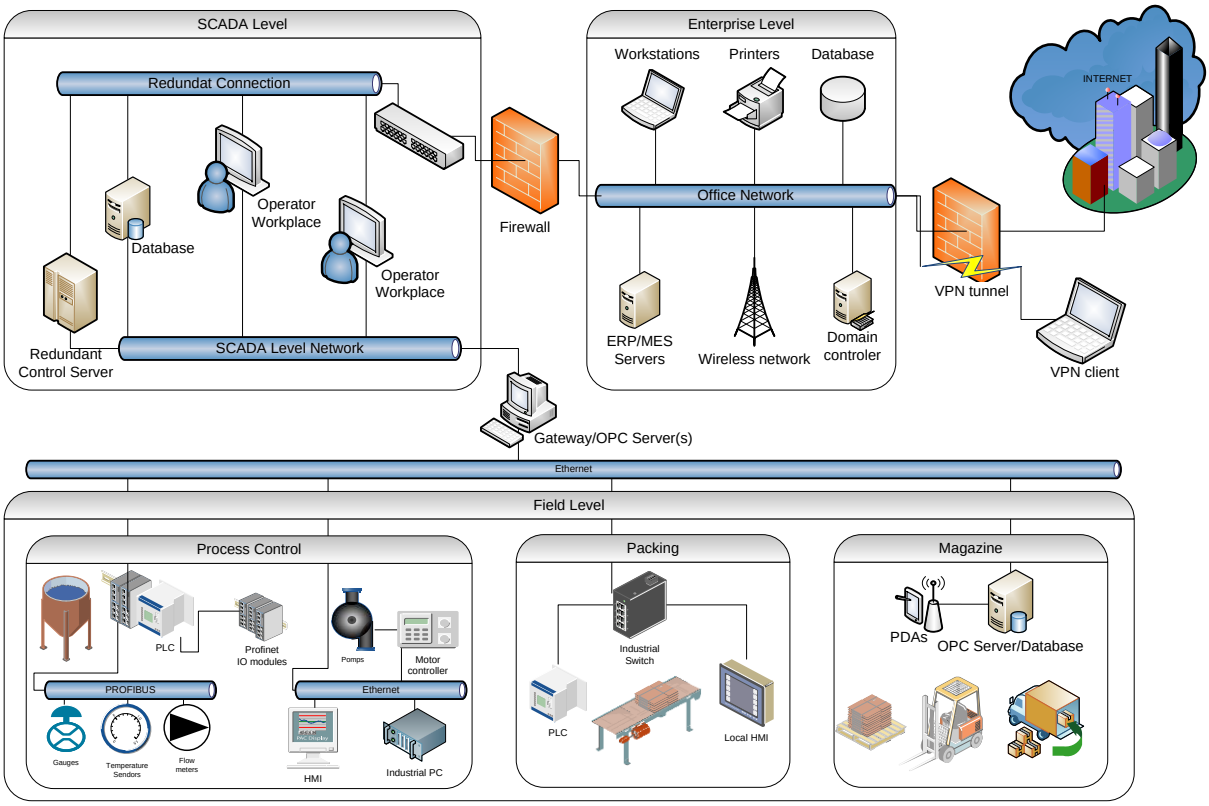


Figure 2.3. Horizontal and vertical integration using different fieldbuses

2.4. Enterprise Level integration

Controlling effectively the process is only a ‘tip of the iceberg’ what is going inside the factory. Efficiency in production is not only based on the effectiveness of the control. In the past, separate applications were responsible for supervising the actions necessary to run the factory. For example, separate applications were responsible for supplies management and production scheduling. There was not enough information exchange between them and due to the lack of integration it was not possible to effectively manage the production. That was a trigger to for development of Manufacturing Execution Systems (MES).

The main task of MES is to provide the automatic flow of information in a factory. With using it we can, carry out maintenance based on real time data. For example: an alarm on the plant floor automatically creates a work order for maintenance service.

MES helps people who are responsible for supplies chain, maintenance, production, capacity planning and other tasks. It calculates data and generates reports required for decision making. Managers are able to optimize the scheduling of short term production, based on all important information - the actual orders, materials available is a stock and many others. Control systems are responsible for automation of a production line, MES is more about automation of human activities.

In companies there was a need of a common system which also comprises functionalities no strictly connected to manufacturing, like Human Resources Management (HRM), Customer Relationship Management (CRM), Order Processing (OP) and many others. The solution was called Enterprise Resources Planning (ERP).

Originally ERP was developed from Material Requirements Planning (MRP). MRP is a set of methods which help in production management. It had been used far before computers became so popular. The biggest problem with using those tools was a lack of integration with a factory floor. With the popularization of MES type solutions, ERP systems started to use them as a source of information. Nowadays, MES is integrated as part of many ERP packets.

An ERP system is based on a common database to enable simple horizontal integration. For vertical integration with SCADA floor OPC is used. The additional OPC traffic generated by ERP systems is quite low and has no influence on overall system performance. Sometimes OPC is also being used for horizontal integration at the Enterprise Level – especially when software manufactured by different vendors need to interconnect. More common way of integrating the software at Enterprise Level is using B2MML (Business to Manufacturing Markup Language). The B2MML is a set of XML schemes which meant to be a common data definition to link ERP and MES *[2.10]*. It was introduced with ANSI/ISA-95 family of standards *[2.9]*

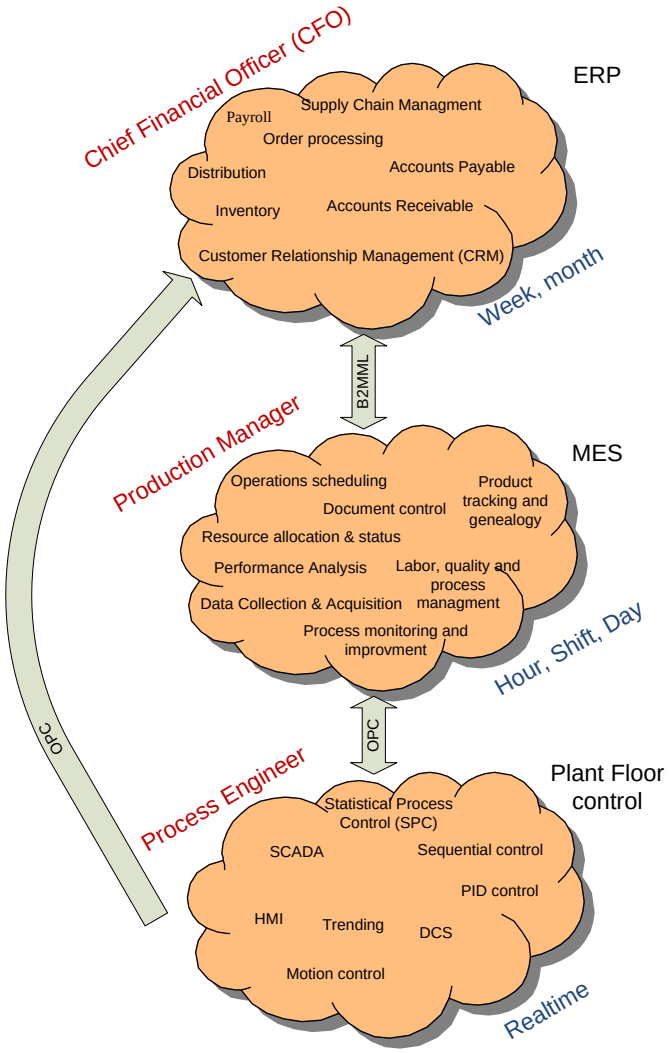


Figure 2.4. Tasks performed on different factory's floors.

References:

2.1. Ronald Dietrich (2005), *Industrial Ethernet.. from the Office to the Machine*

2.2. http://en.wikipedia.org/wiki/System_integration (2008.09.10), *System integration - Wikipedia free Encyclopedia*

2.3. http://en.wikipedia.org/wiki/Virtual_private_network (2008.09.10), Virtual Private Network – Wikipedia free Encyclopedia

2.4. http://en.wikipedia.org/wiki/SCADA (2008.09.10), Supervisory Control And Data Acquisition – Wikipedia free Encyclopedia

2.5. Hwaiyu Geng , *Manufacturing Engineering Handbook, chapter 11.3.5*

2.6. http://en.wikipedia.org/wiki/Enterprise_resource_planning (2008.09.13), *Enterprise Resource Planning - Wikipedia free Encyclopedia*

2.7. http://www.automation.com (2008.10.01), *SCADA/Business Network Separation: Securing an Integrated SCADA System*

2.8. Rafał Cupek (2007), *Akwizycja danych z sieci PROFINET CBA do systemów klasy MES*

2.9. http://www.isa-95.com (2008.12.01), *Technology ISA-95*

2.10. http://en.wikipedia.org/wiki/B2MML (2008.12.01), *Business To Manufacturing Markup Language - Wikipedia free Encyclopedia*

3. OPC Overview

3.1. State of art before OPC appearance

Before OPC, software developers had to develop specific communications drivers for each control system they wanted to interface with. There was no unified standard and different vendors developed their own technology. Human-Machine Interface (HMI) suppliers had to spend much time and money to develop hundreds of different low-level drivers for the various Distributed Control Systems (DCS) and Programmable Logic Controllers (PLCs). A change in the hardware's capabilities could break some drivers. Suppliers really never had enough resources to support so many different issues. Costs of control systems integration were huge. It was pretty obvious that common standardized interface was needed.

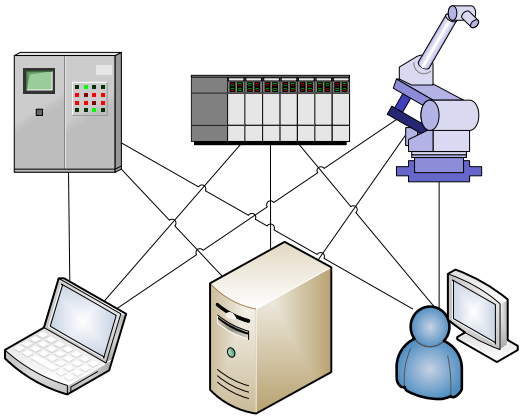


Figure 3.1. Information flow before OPC has been released

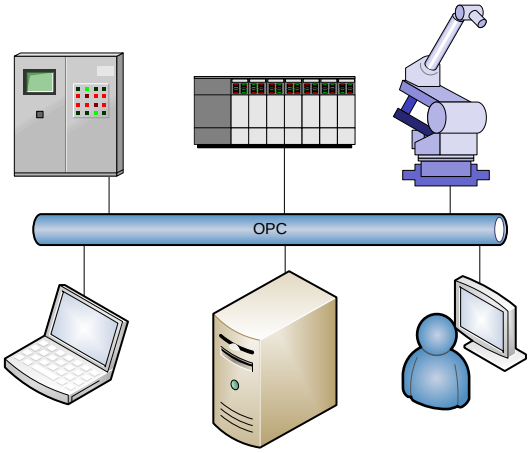


Figure 3.2. Information flow after OPC has been released

3.2. OPC History

The need for OPC probably can be dated back to the introduction of Windows 3.0 in 1990. With Windows 3 it became possible, on an inexpensive, mainstream computing platform, to run multiple applications simultaneously *[3.2]*. Dynamic Data Exchange (DDE) was presented in 1987 with release of Windows 2.0 as a simple mechanism to share data between those applications. It sends data between applications using shared memory. It was obvious that the same kind of interoperability was needed between plant floor and SCADA applications.

In 1993 Microsoft introduced Component Object Model (COM) to enable simple reuse of software components with no knowledge of their internal implementation. COM technology allows objects to be used in environments different from the one they were created in, even on another operating system. Distributed COM (DCOM) extends the Component Object Model (COM) to support network communication among software objects distributed across different computers. COM is the also foundation technology for Object Linking and Embedding (OLE).

Although, the DDE mechanism is still used by many applications, gradually it is being replaced by OLE, which provides greater control over shared data. OLE is much more powerful on access performance, security, reliability and flexibility. OLE technology allows objects embedding. For example, a CAD drawing can be integrated into a Word document or an Excel table. The aim with OPC is to achieve the same kind of interoperability with the connection of process control equipment to a computer based process monitoring or control systems.

In May 1995, in co-operation with Microsoft, OPC Task Force group was founded. The committee's aim was to fully define interfaces that are based on the OLE/COM standard. By the end of 1995 the first draw of the OPC standard was made and in August 1996 first standard was released. Originally it was called the OPC Specification and now commonly named the Data Access Specification. In September 1996 the OPC Foundation was established as a supervisory body for OPC development. Since then numerous different specifications were released. Nowadays the foundation associates all major

industrial equipment producers and many of those specifications have become an industrial standard.

3.3. OPC protocol releases

3.3.1. OPC Data Access

As it was mentioned before the first OPC standard was fully based on Microsoft's OLE/COM. OPC Data Access is a group of standards that provides specifications for data communication. You must remember OPC DA deals only with real-time data and not historical data or alarms. With each real-time data time stamp and signal quality value is associated. Till now the OPC Foundation has released second version of the specification, but the OPC Data Access 3.0 is now a Release Candidate. According to OPC Foundation it will leverage earlier versions of OPC while improving the reading capabilities and incorporating XML-DA Schema.

The OPC technology has a Client – Server architecture. An OPC Client can be connected to many OPC Servers provided by different vendors. Important feature of OPC is that the client is able to connect the OPC server which is located in other computer in the network. This is possible because of DCOM (Distributed COM) the COM standard extension which enables client running on one computer to create instances and invoke methods of servers on another computer within the network. It is important to configure properly DCOM from security point of view. Usually, it means that the firewall should enable DCOM traffic and the user creating connection to remote computer should have access rights granted to this computer. If two computers on two different domains need to communicate the domains where they are placed need to share at least one common username and password. Very often DCOM security configuration require a lot of work and causes loss of time.

Generally, there are two types of OPC Servers: in-process and out-process. In-process OPC servers are very rare. They are written in form of a Dynamic Link Library (DLL) loaded in the OPC Client's process space. For this kind of server only one client can be connected, because each OPC Client application manages its own Server instance. The communication between the client and

the server is being done inside one process, so it much faster. Generally in-process OPC do not offer remote OPC connections. Out-process OPC servers are those we will meet in most cases. They are separate applications. Multiple clients can be connected to one server. They offer both local and remote connections.

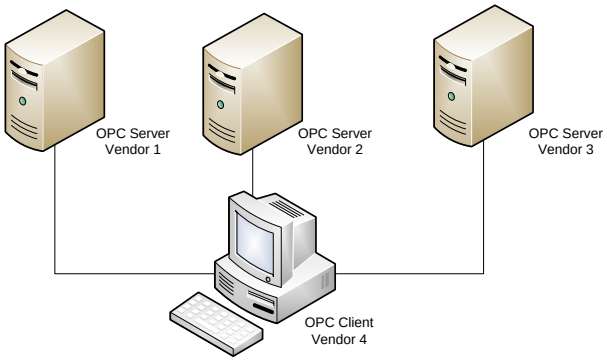


Figure 3.3. OPC client connected to several OPC Servers.

Many OPC clients can be connected to the OPC Server. Vendor supplied code determines the details about how server is connected to the devices and data to which each server has access.

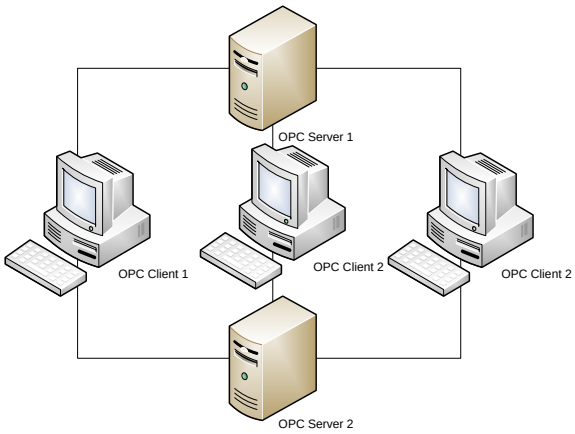


Figure 3.4. Multiple OPC clients connected to OPC server.

Every OPC server defines the namespace, which assigns the unique name to every data item, which can be communicated. This namespace can be flat or hierarchical. Flat namespace has only one level where all the items are located. Hierarchical structure is similar to the tree with branches and leafs - it is similar to the folders and files structure on hard drive. The OPC standard does not presumes any rules for data item names definition. The name can contain any characters, including spaces, special symbols etc.

Usually OPC server provides the unified interface for browsing the server. Such interface is described in detail in the OPC DA specification. Its implementation is not obligatory for server's vendors but it is strongly recommended, because it makes working with server much easier.

To simplify data management an OPC DataAccess software implementation is comprised of several objects: the server, the group, and the item. 'The server' object holds information about the OPC server and is a container for 'the group' objects. 'The group' object contains information about itself and provide mechanism for controlling the 'item' objects. Each 'item' object is linked with the item from server's namespace. An element from the namespace can be associated with many 'item' objects. The group is private variable of OPC Client, but the client who owns group can make it public – than it can be visible by other clients connected to the server.

The 'update rate' time is associated with each OPC group. This property defines the time between when the dead band limit is checked.

The client can specify an 'update rate' for each group. This determines the time between readings by the server the current item value from the device. If the change of value excesses specified percent (dead band) the server updates its internal memory (cache). Exception based connections can be created between the OPC client and the items. It means that the server can inform clients about changes in a value of the item. Items in the group and can be enabled and disabled. An OPC Client can configure the rate that an OPC server will provide data changes. In OPC DA 3.0 it is possible to specify update rate both for the group and for each item in the group separately, but the update rate associated with individual items does not effect the callback period for exception based connections. For the OPC DA 2.0 and earlier it was only possible to specify the update rate for all OPC items it the group.

Of course the client can request data with higher rate than it was specified by the group refresh time. The server should make a best effort to keep the client informed about data changes, but it should never send data to a client at a rate faster than the client requests. It is to be remember the OPC server has also its refresh rate. Usually this parameter is hidden in the server's configuration. OPC

users should be familiar with it and remember setting OPC group refresh rate faster than OPC server refresh rate causes that the real OPC group refresh rate will be equal to server's refresh rate.

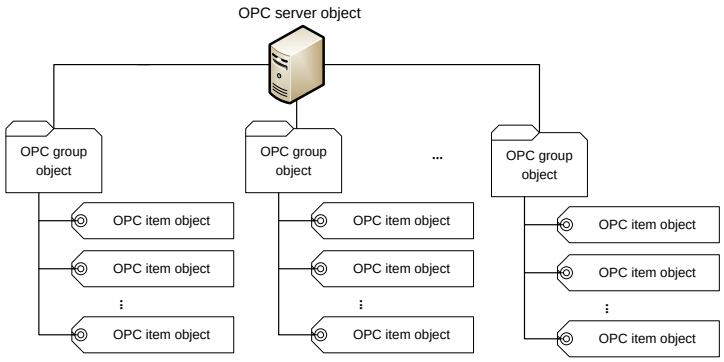


Figure 3.5. OPC Server logical structure.

The structure presented in *Figure 3.5* provides simple method for OPC clients to organize data. For example a particular group can comprise data for a particular Operator's Panel. Each OPC item represents connection to data in the server. Note that it is not a data source, but only a data connection. An OPC item can be only access via the OPC group that contains that item. A group can comprise items from different PLC, but among only one server *[Figure 3.6]*.

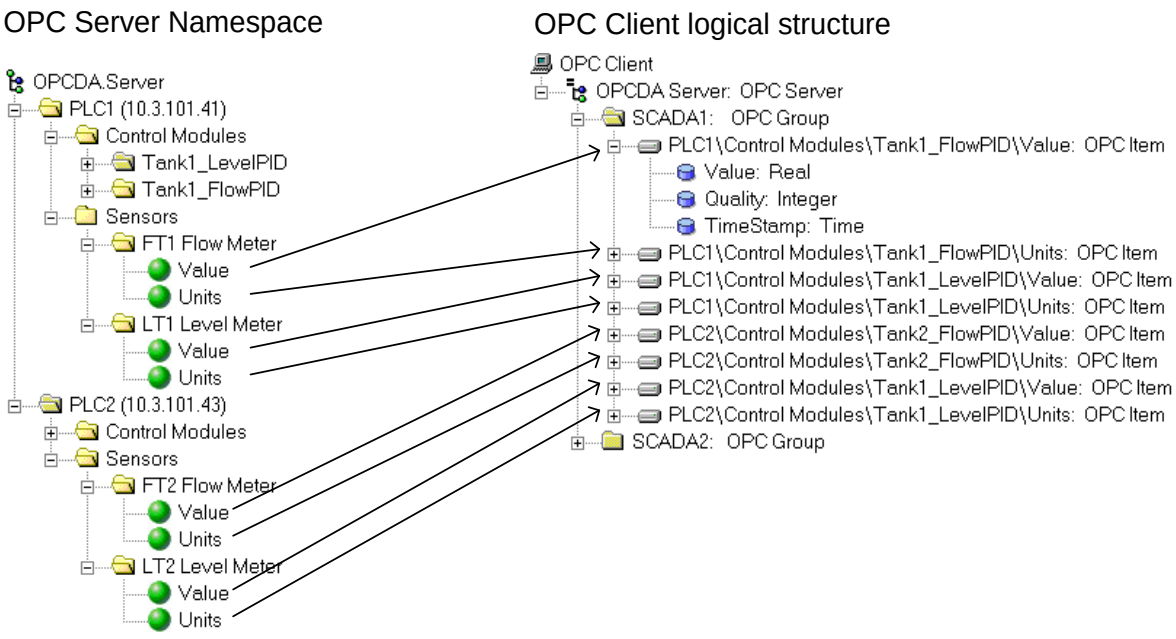


Figure 3.6. Dependences between server's namespace and client's logical structure

3.3.2. OPC Alarm & Events

Alarms are mainly useful to indicate improper system behavior which requires immediate reaction. An event is only a change of state and usually no operator reaction is needed, it can be used to inform about occurrence of particular, normal for system utilization situations.

In addition to human operators, also software applications may detect and record alarm and event information for later system audit or other purposes. In fact, nowadays Alarm & Events engines produce the vast stream of data which has to be distributed to users and software clients. Currently many vendors use its own alarming/event subsystems for this purpose.

The OPC Alarm & Events specification was written to establish common standard for communicating to the numerous alarm and event subsystems. With this standardization, interoperability is high and system integrators are able to use off the shelf solutions to easy integrate all levels of business.

The controllers generate alarms and events in response to changes in the plant variables, together with their precise time of occurrence, type, severity and associated message for the human operator. OPC Servers collect these data and make them available for multiple clients. According to the OPC Alarm & Events Specification, servers can be divided into following groups:

- Servers that can detect alarms and/or events and report them to one or more clients.
- Servers that can collect alarms and events from multiple sources. Those servers can detect alarm/event by them own or can obtain it from other OPC server and report such information to one or more clients.

The clients of Alarm and Events servers may include:

- Operator software
- Alarms & Events logging components
- Alarms & Events management subsystem

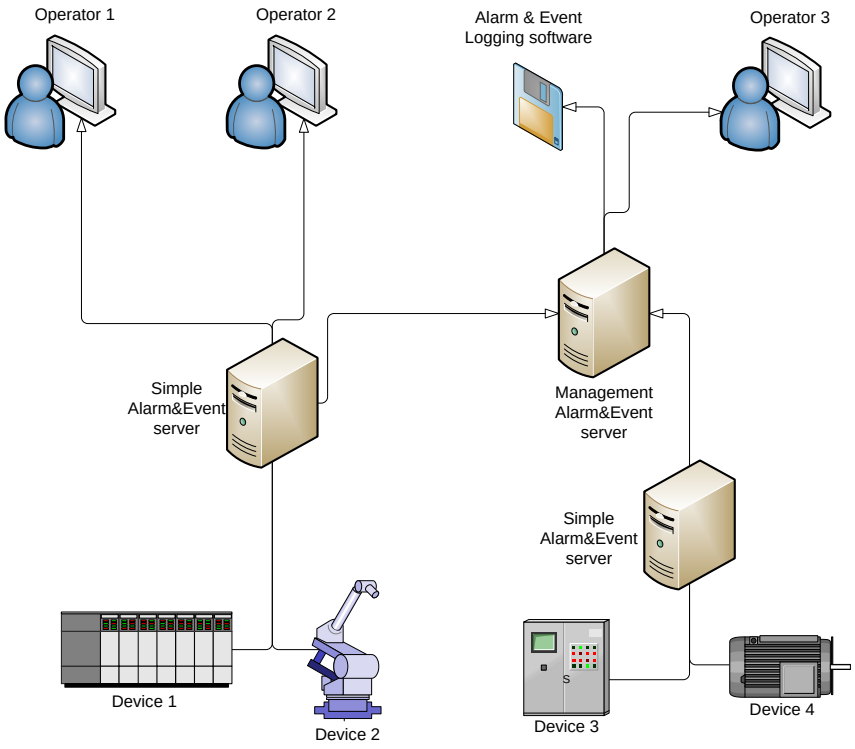


Figure 3.7. OPC Alarm & Event Client-Server structure.

The Alarm/Event Management server is also a client to more than one OPC Alarm and Event server. In this model, the Alarm/Event Management server is acting as kind of a collector or data concentrator, providing its clients with perhaps more organized information or a more advanced interface.

3.3.3. OPC Historical Data Access

Correspondingly to previously described OPC specifications, OPC Historical Data Access (OPC HDA) has been created to integrate software which need access to the past data. If present data are requested Data Access interface should be used.

The OPC HDA specification does not define the set of obligate data to which server should have access and how the data should be read. The source information for the OPC HDA server can be OPC DA server, as well as propriety historian server or database. Vendor supplied code determines the data to which each server has access, the data names, and the details about how the server physically accesses that data. An OPC HD*A* compliant client can read from OPC HDA servers regardless of its vendor.

OPC HDA servers are diversified in functionalities. Simple OPC HDA servers provide a bit more than simple raw data storage (Time, Value & Quality). Complex OPC servers usually offer data compression, interpolation and analyses. They can aggregate historical data to obtain such information like average, minimum, maximum value, variance, standard deviation and many others. Complex OPC HDA server features depends on vendor implementation.

Most vendors do not supply OPC HDA server for their devices because the real-time data can be obtained using OPC DA and then store into some sort of database.

3.3.4. OPC Batch

Batch production is a special manufacturing method used in some special areas where product is created in batches. Batch production is popular in bakeries, pharmaceutical industry, manufacturing of inks, paints, wallpapers.. etc. Batch process data can be divided into four basic types such as equipment capabilities, current and master recipe, historical and current batch execution.

A special OPC specification has been created to support batch production. In the case of batch control, there was a common accepted standard that defined batch production related terms and how the batch production process should be described (IEC 61512-1, ANSI/ISA S88.01). Those standards had to be taken into account when OPC Batch specification was created.

OPC batch server supports all interfaces required by OPC DA specification, as well as some additional ones dedicated for the batch production. The server collects data from an OPC DA server or other data source. The OPC batch server has a well-defined namespace that must be supported. The OPC Batch client must understand all OPC DA 2.0 interfaces, there are no additional interfaces, but the client must recognize the OPC batch namespace.

The OPC Batch server structure consists of a few well-known Items IDs *[Figure 3.8]*. The existence of those items comes from compliance with standards for batch production. All well-known items are required and their names start with “OPCB” prefix. This prefix is reserved and should not be used for vendor or user defined items.

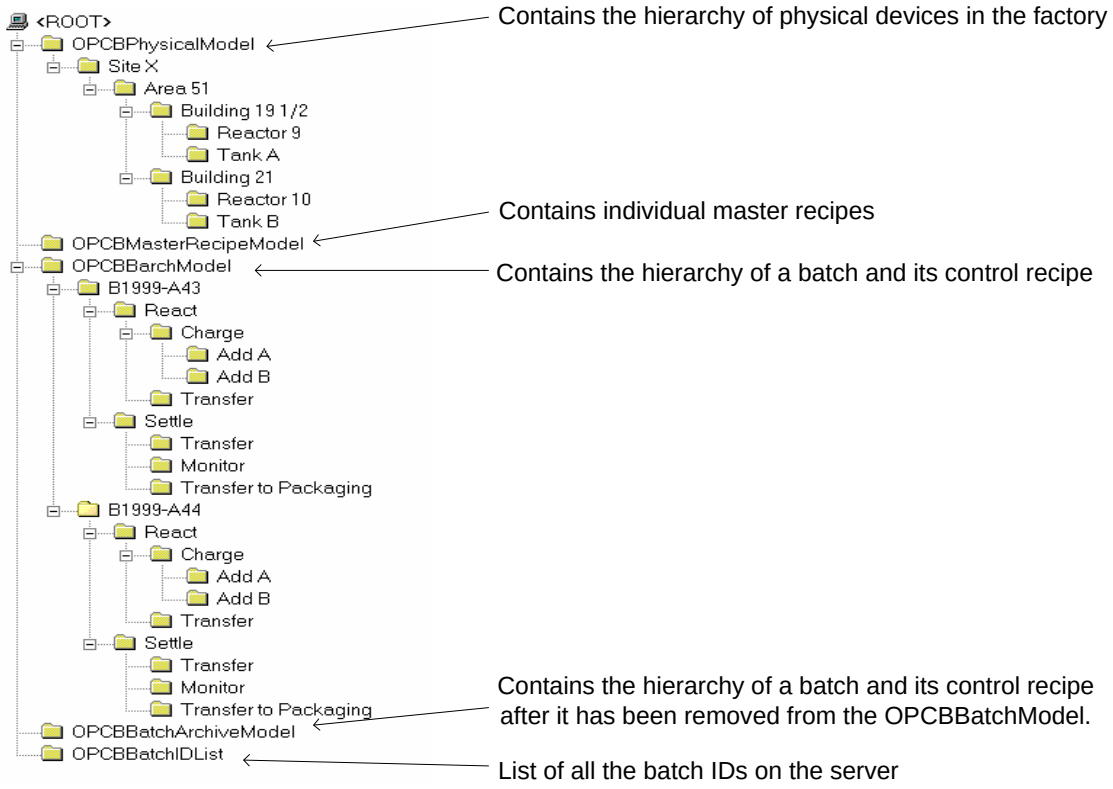


Figure 3.8. OPC Batch server hierarchy

3.3.5. OPC XML-DA

All previously described OPC specifications are based on Microsoft's OLE/COM technology. XML (eXtensible Markup Language) provide another means to describe and exchange structured information between collaborating applications. The OPC XML DA specification is an adoption of XML technology for communication on various levels of the plant hierarchy. The main advantage of OPC XML-DA is possibility to exchange XML data over internet, and upwards into the enterprise domain. The second important benefit of using OPC XML-DA is platform independence. Both Windows and Linux can be used. Unfortunately, XML-DA offers refresh rate 6 times slower than regular OPC-DA interface *[3.11]*, but for many purposes it is enough.

Not like in previous OPC specifications the OPC XML assumes no persistent connection between client-server. Client application initializes data subscription and sends periodic requests for data refresh *[Figure 3.9]*. Complex OPC XML-DA client applications offer more sophisticated mechanisms to

communicate with server. Those modifications are used to simulate traditional asynchronous callback provided with OPC COM based interface.

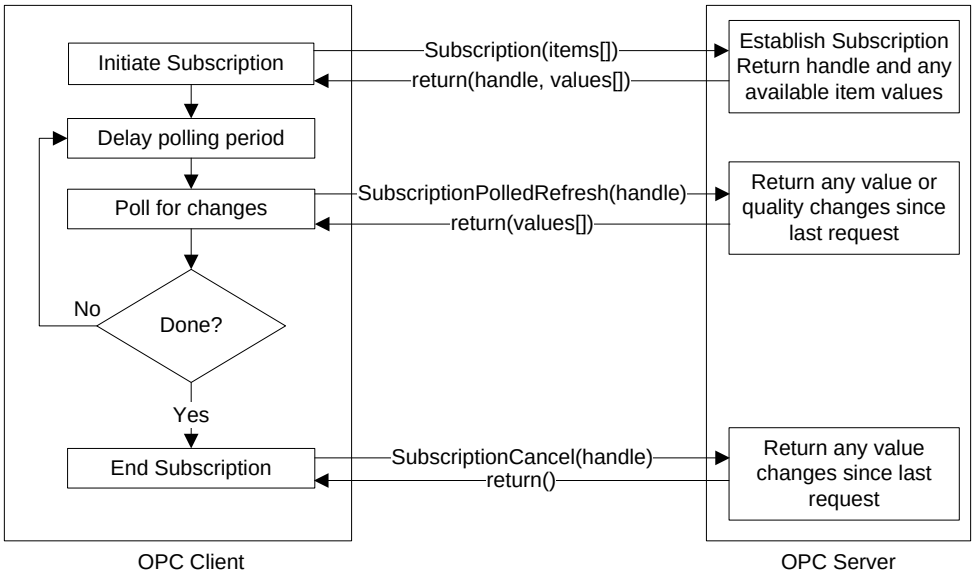


Figure 3.9. OPC XML basic polled subscription mechanism

In terms of security OPC XML is more complicated. In the past, OPC was restricted to LAN networks and security could be guaranteed with using DCOM security policy and firewalls to protect from exterior intruders. With the advent of OPC XML, process control information is no longer restricted to LAN.

Different security mechanisms were created for OPC XML. The security strategy may include restricting access to web services to authorized users and data coding with SSL (**S**ecure **S**ocket **L**ayer). In this matter, it is important to ensure that end users will enable at least one of them. It is highly recommended that, as a minimum, vendors provide security mechanism which enable possibility to globally disable writing to the Server by specific user.

3.3.6. OPC SNMP

Control networks have usually very high reliability requirements. It is critical to supply robust control and monitor of traditional industrial assets (tank, valves). Nowadays, IT assets such as computers, software, routers and switches also play an important rule. When IT assets fail, the consequences can be just as serious as when the equipment controlling the process fails. It is very important to enable possibility to centralized management of TCP/IP-based networks and

to inform system supervisors about current network condition and network breakdowns.

Simple Network Management Protocol (SNMP) is standard protocol commonly used in network management systems to manage and supervise network attached devices (switches, routers, UPSs, telephone exchanges). SNMP exposes management data in the form of variables on the managed systems, which describe the system configuration. The variables accessible via SNMP have hierarchical structure described by *Management Information Bases* (MIBs). SNMP specification does not indicate which information management system should offer it rather specify how MIB structure should look like.

OPC SNMP was created to easy integrate network monitoring feature into SCADA systems. The OPC SNMP servers use also ping protocol to manage non-SNMP devices and display round trip time as an OPC variable. Some OPC SNMP servers can also inform about network failure sending problem notifications via email or as OPC Alarms & Events (A&E) messages. Those messages can be integrated into existing Alarm & Event management solutions.

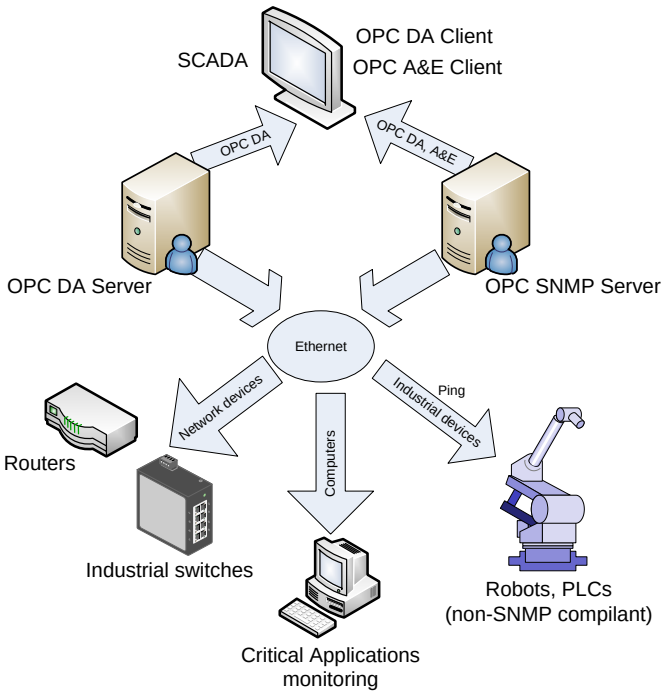


Figure 3.10. Implementation of OPC SNMP into control system.

3.3.7. OPC DX

In contrast to other previous described OPC specifications OPC DX was mainly created for horizontal integration between different OPC DA servers. The goal of the specification was to provide means for direct transfer of data from one or more OPC DA and DX servers to an OPC DX server. Before OPC DX was released, when an OPC server wanted to read from another one, an intermediate OPC client was needed. Some OPC vendors developed intermediary solutions to read data from one OPC DA server and forward it to another. These proprietary products do not have a standardized interface that would allow interoperability.

OPC DX is designed as a simple solution that enables simple server to server communication without store-and-forward intermediary and with a standard configuration interface. OPC DX do not define any new method for data exchange, instead it rather use OPC DA interface for communication between servers. What is the most important is a fact that the connection between OPC DA and OPC DX is one way connection. This means that OPC DX items, read from OPC DA server, are read-only.

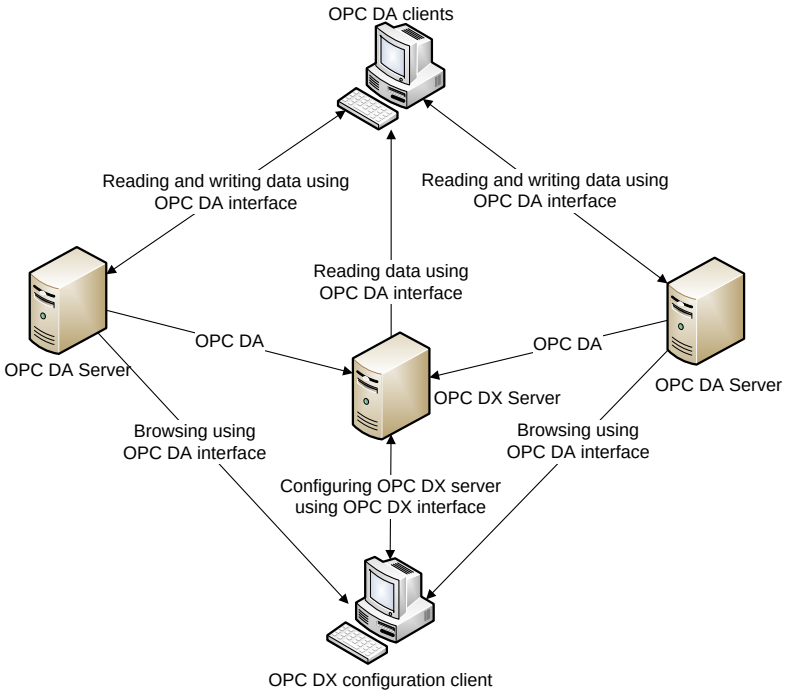


Figure 3.11. OPC DX architecture

3.3.8. OPC Unified Architecture (OPC UA)

The goal of OPC UA is to merge all previous OPC specifications and integrate into a single OPC server. The intention of the new specification was to provide a new kind of interoperability by supplying a secure, unified, platform-independent standard through which various kinds of software and hardware can cooperate.

The OPC UA specification has well defined data and service structure. A single OPC UA Server integrates process data, alarms & events, and data history into to structure called the AddressSpace. The AddressSpace is a collection of all information that an OPC server makes visible for its clients. Access to it is being made by integrated set of Services. These services offer very similar capabilities to the existing OPC DA, AE and HDA interfaces, but in a unified way.

Security model is integrated into the service structure. They are various security mechanisms implemented in UA specification:

- the authentication of *Clients* and *Servers,*
- the authentication of users,
- secure connection,
- security rights depending on user.

The specification does not define the conditions under which the specific security mechanism is required.

OPC UA allows data to be exposed in many different ways. Both TCP and Web Services over HTTP can be used as a protocol layer between client-server. Supporting HTTP and TCP allows OPC UA to run data communication over internet with HTTP as well as running optimized applications with limited resources via simple TCP without HTTP stack.

The main optimization in performance can be done not with exchanging HTTP to TCP but with switching from XML to binary coding. This will allow end users to decide between OPC DA performance and OPC XML-DA capability. For time critical applications efficient binary transmission should be used. However, for

application that does not require high performance but rather data provided in a generic way XML is a perfect solution.

Similarly to OPC DA callback functions can be used to indicate data change to client. Also pulled refresh mechanism introduced with OPC XML-DA can be used.

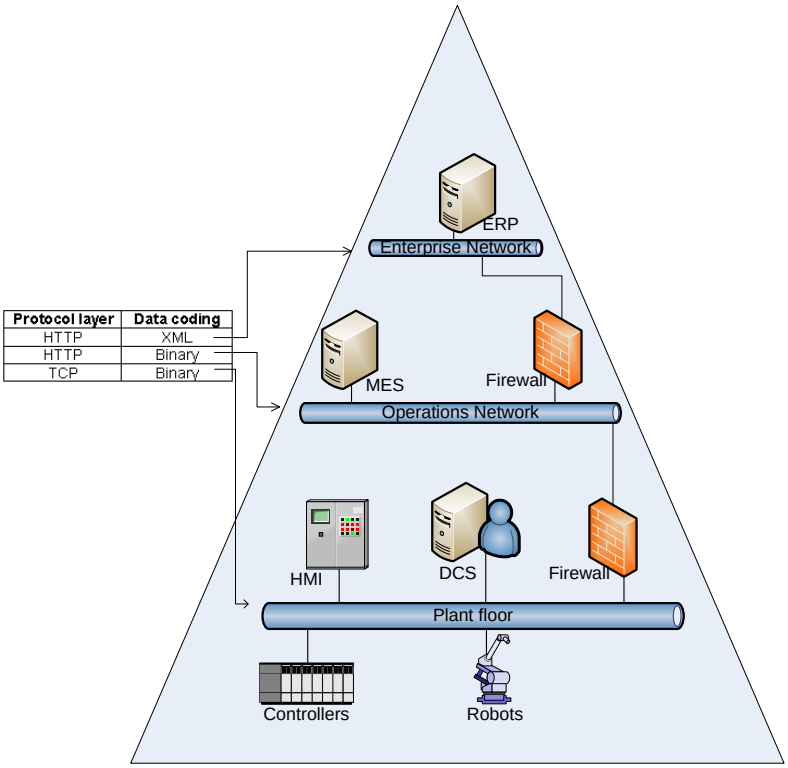


Figure 3.12. OPC UA protocols and encodings *[3.14]*

OPC UA is designed to support a wide range of server types. Today servers are situated on various plant levels from plant floor to enterprise. These servers need to be characterized by a broad scope of size, performance, execution platforms and functional capabilities. Therefore, OPC UA defines a comprehensive set of capabilities, and server vendor may implement just a subset of these capabilities.

As it was mentioned before OPC servers can be situated on various plant floors. To supply communication between them, server to server interactions are possible. In this configuration one server acts as a client of another server *[Figure 3.13]*.

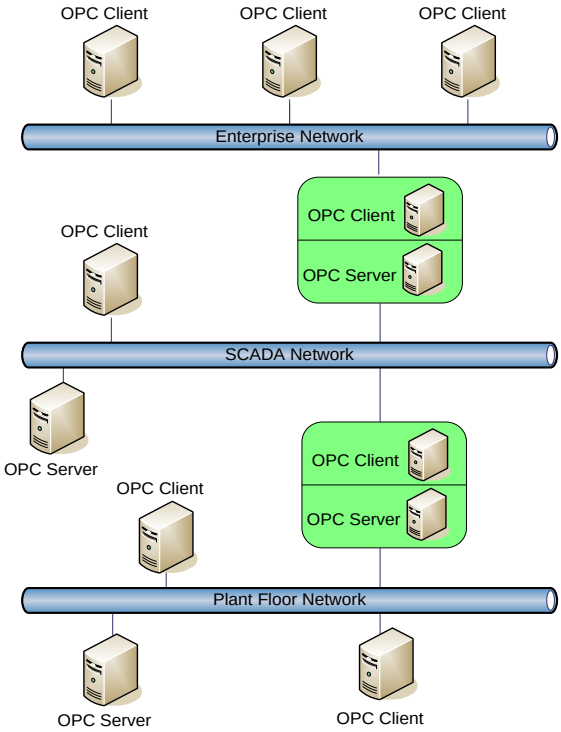


Figure 3.13. Vertical integration with OPC servers.

It seems that large amount of work is necessary to be done to migrate from classic OPC to OPC UA, but since OPC foundation provides OPC UA wrappers for OPC servers and proxies for clients there is almost no need do changes in application code. You simply compile existing product with new OPC UA wrapper/proxy and use it with native OPC UA clients/servers. However, if you want to use all additional features of OPC UA or provide platform independence (separate your new code from all COM interoperability problems) you need to do more than just using a wrapper/proxy.

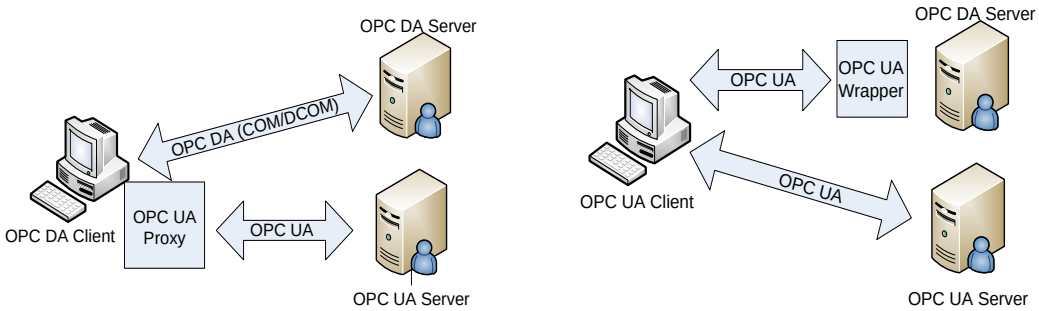


Figure 3.14. Using OPC UA wrappers and proxies.

References:

3.1. OPC Foundation (1998), *OPC Overview 1.00*

3.2. http://www.opcconnect.com/history.php (2008.07.10), *History of OPC*

3.3. OPC Foundation*, OPC DataAccess 2.02,*

3.4. OPC Foundation*, OPC DataAccess 2.05a,*

3.5. OPC Foundation*, OPC DataAccess 3.00*

3.6. OPC Foundation*, OPC AE 1.10 Specification*

3.7. OPC Foundation*, OPC HDA 1.20 Specification*

3.8. OPC Foundation, *OPC Batch 2.00 Specification*

3.9. OPC Foundation, *OPC Batch Auto 1.00 Specification*

3.10. OPC Foundation*, OPC XMLDA 1.01 Specification*

3.11. Frank Iwanitz, *XML DA opens windows beyond the firewall - The Industrial Ethernet Book May 2004*,

3.12. http://en.wikipedia.org/wiki/Simple_Network_Management_Protocol (2008.10.10), SNMP - *Wikipedia free Encyclopedia,*

3.13. OPC Foundation, *OPC DX 1.00 Specification*

3.14. Wolfgang Mahnke (2009), *OPC Unified Architecture*

4. OPC interface for software programmers

4.1. Implementing OPC in software applications

OPC DA is currently the most popular OPC specification, and recently became as a true industrial standard. In this part of the thesis I will describe in details an OPC DA wrapper.

The aim of the OPC Data Access specification was to describe the unified OPC COM Object and its common set of interfaces which must be implemented by servers' vendors to enable the interoperability. The idea with the specification was to enable the development of OPC Servers in C or C++. OPC Clients can be implemented also in other programming languages.

An OPC client connects to an OPC server and communicates to the OPC server through the COM interfaces. Interface is a set of functions which allow contact between COM object and COM client.

There are two possible ways of implementing an OPC DA Client, because, the OPC specifications always contain two sets of interfaces: OPC Custom Interface and OPC Automation Interface. It has been found that it is not possible to define a single set of interfaces which will be suitable for all purposes. The OPC clients which need high efficient data exchange will use C++ to implement OPC Custom Interface communication. In general, client programs which are created using scripting languages, like Visual Basic will use the OPC Automation Interface. It is not as fast as the Custom Interface but it is more user-friendly and do not require programming language which supports pointers. The Automation Interface is probably much more popular, there are universal, readymade solutions based on it which enable support of OPC for office applications like Excel.

All OPC servers must support the Custom Interface – that is why there are not OPC server implementations in VB. It is not required to implement the automation interface, but in almost all OPC servers it is implemented. The fundamental design goal is that OPC DA Automation Interface is intended to work as a 'wrapper' for OPC Data Access Custom Interface *[Figure 4.1]*. The Automation Interface provides almost all of the functionality of the required and

optional interfaces in the OPC Data Access Custom Interface. It maps calls from the Automation Interface to calls of the Custom Interface. If the OPC Data Access Custom server supports the interface, the functions and properties at the automation level will also work. In many cases Custom Interfaces will provide more configurability, where the Automation Interfaces will enable only some reasonable default behavior

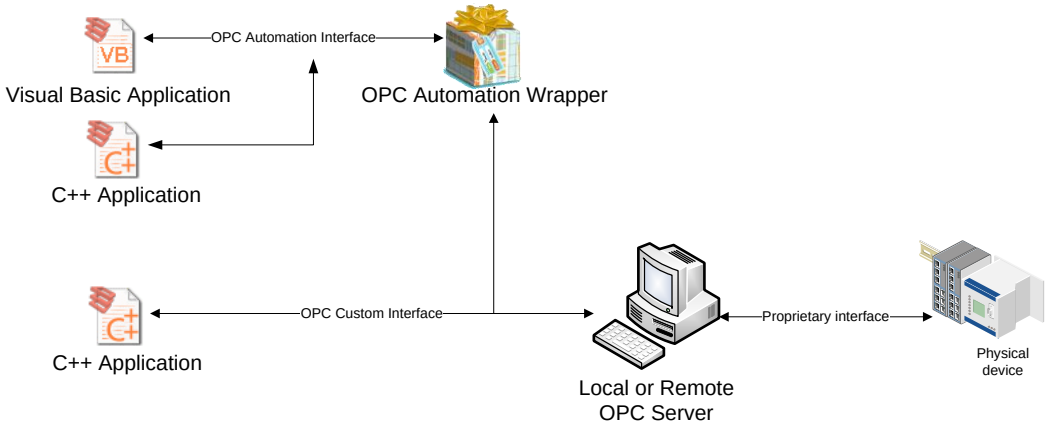


Figure 4.1. OPC Automation Interface implementation

The OPC DA specification describes briefly both interfaces, but it is more behavioral descriptions than information how to implement particular interfaces. It is not a subject of this thesis to go into details of the OPC DA specification, but in due to the contents of further chapters it is crucial to describe those interfaces which are responsible for data communication.

One of the most important issues for communication between devices is to the assure synchronization. By ‘synchronization’ to be understood that the client is able to read or write values and attributes in a single transaction. For example it is essential to read item value, quality and time stamp simultaneously. In some cases items are correlated and it is critical they are read/written at the same time. So does OPC assure it? The answer is that OPC itself do not guarantee the full synchronization, but in most cases it is fast enough to fulfill the requirements. In general, all data read or write in a single operation should preserve synchronization.

In those particular cases where synchronization is crucial, additional, self-made handshaking mechanism is needed. This mechanism similar to semaphores and can be based, on flag passing between the OPC client and the OPC

device. For example the additional synchronization is needed while reading from Excel and transferring data to a Profibus device through PLC *[Figure 4.2]*. The data should be send via Profibus to the device in one data telegram. Reading from Excel is slow, so it can exceed a single OPC refresh cycle. To indicate the moment when all data is transferred from Excel to PLC and can be send further an additional OPC variable is needed. It is being set by Excel when the data transfer is finished.

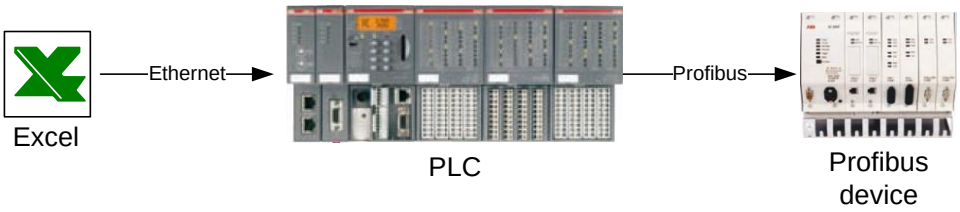


Figure 4.2. Connection between Excel and a Profibus device

Another important issue for OPC implementations is to protect data serialization. The OPC client must be able to control the order in which data is delivered to the physical device. OPC implementation must guarantee, that the write requests to the same device will be handled and transmitted in order they were demanded.

4.2. Communication between OPC Server and OPC Client

The OPC standard defines numerous ways to communicate between the server and its clients to satisfy different OPC applications. The client can specify that some operations should be performed on ‘cache’ or ‘device’. If ‘device’ is chosen, operation directly on the physical device will be requested. If ‘cache’ is selected data will be read from server’s internal memory where server keeps a copy of device data received during the last OPC refresh telegram.

Both reading and writing can be done synchronously or asynchronously:

- OPC synchronous functions run to completion before returning. It means that the OPC client program execution is stopped till operation is over.
- Asynchronous functions use callback mechanism to inform the requested OPC operation is over. The client program execution is not stopped while waiting for OPC operation completed.

Asynchronous reading and writing can be done only from the ‘device’. After function call, requested operation is queued in the server, client program execution is continued. Each operation is treated as a ‘transaction’ and is associated with a transaction ID. As the procedure is completed, the callback function with transaction ID will be triggered in the client application.

The synchronous reading operation reads the value, quality and timestamp information for one or more items in a group. The OPC client can read synchronously data either by ‘cache’ or by ‘device’. The specification does not describe details for implementing ‘cache’ and ‘device’. There can be a difference in performance depending on vendor’s implementation. In most cases, access by ‘cache’ should be faster. Data obtained by ‘cache’ reading reflect the latest value of item, its quality and time stamp. It should be valid within the OPC group UpdateRate and DeadBand percent.

While ‘device’ reading is used, server is forced to communicate directly with the hardware. Working directly with the ‘device’ is considered to be slower but more accurate. Time needed for single reading operation depends on the device itself and the network performance. It can take a very long time and depending on details of the server’s implementation, while device reading, other operations can be blocked from being performed by any other clients. For this reason in most cases, while reading from the OPC server, it is recommended to use cache reading. Device reads are intended for ‘special’ circumstances such as diagnostics. When using synchronous device reading it is important to remember the function will not return until operation is finished.

Alternative method for reading data from server is subscription. OPC Client can subscribe for particular item(s). Server obtains from data from the device at a periodic rate sufficient to hold the specified group UpdateRate. If the Item quality has changed or change of value exceeds the specified Deadband criteria, server sends to the client an exception with information about new data. The cache of the server will be also updated with a new item value

The synchronous writing operation writes values to one or more items in a group. Because it is not possible to write to server’s cache, communication to the ‘device’ is always performed. As it was mentioned before synchronous OPC

functions run to completion. It means that synchronous writing function will not return until it verifies that the device has actually accepted (or rejected) the data. For this reason clients are expected to use asynchronous write rather than synchronous write in most cases.

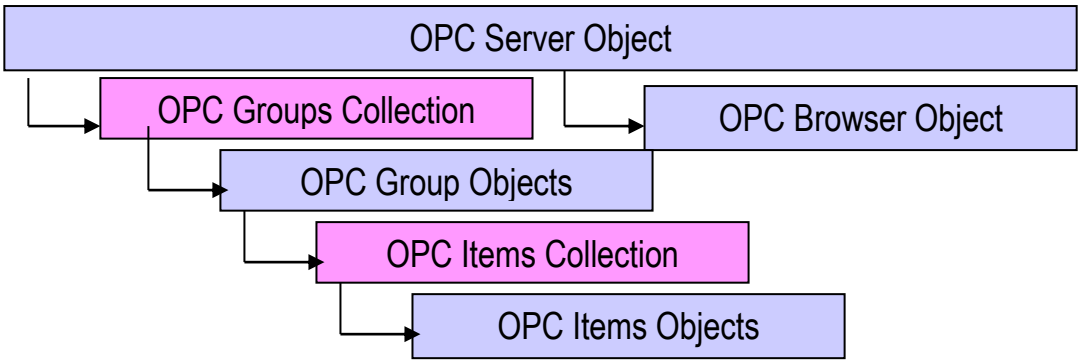


Figure 4.3. OPC client objects' structure

To describe a client software implementation the OPC Automation Interface will be used. It is similar to Custom Interface but probably easier to understand. As it was mentioned before, OPC client comprises of several object types: Server, Group and Item *[Figure 4.3]*. Additionally OPC DA introduces a *Collection* term. Generally, collections are simple objects that should support methods: Count, Item, and a hidden property called _NewEnum.

A client creates an OPCServer object *[Table 4.1]* and connects it with an appropriate server. The server object contains the OPCGroups collection and OPCBrowser object.

The OPCBrowser object is a collection *[Table 4.2]* that provides information about all data sources that exist in the server. Browsing is optional feature. Server can support the interface to browse its namespace, but it is not obligatory. If the server does not support browsing, CreateBrowser will not create this object. ShowBranches and ShowLeafs are two main methods, for browsing the server. If the server has a flat namespace by using ShowLeafs method you will receive an entire collection of names items in the server. In case of hierarchical structure, first you need to use ShowBranches method to receive the server's branch structure. By calling MoveUp and MoveDown you can easy go through all branches. On each level you can call again ShowBranches to receive the structure below or ShowLeafs to collect all items on the actual level.

The OPCGroups collection *[Table 4.3]* is a container that holds all OPC group objects the client has created, it also contains default settings for a new created OPCGroup object. OPCGroups contain interface for Public Groups connecting. Public Groups are pre-existing groups in a server. These groups can be connected rather than added.

The OPCGroup object *[Table 4.4]* provides a way for client to organize data. Almost all important settings for reading and writing such as the UpdateTime, DeadBand are being defined by setting the OPCGroup properties. Generally, all reading and writing operations are being performed via OPCGroup methods. In most cases client just define which items from group members should be read or written. Also in case of reading by subscription the client has one procedure for handling events from all items in the group – this procedure will be fired when at least one of the items changed their value or quality.

On the lowest level of the OPC data structure are the OPCItem objects *[Table 4.6]*. They represent a direct connection to data sources within the server. Beyond obvious properties which it must contain like: Value, TimeStamp, Quailty or CanonicalDataType the OPCItem object have Read and Write methods. The idea with whose methods was to provide an extremely easy way for simple applications to work with OPC data. The ease of use was achieved through performance decreasing. In most server's implementations user should assume that server will behave as if it was synchronously read/write to/from the DEVICE.

OPC Server - summary of properties		
StartTime	CurrentTime	LastUpdateTime
MajorVersion	MinorVersion	BuildNumber
VendorInfo	ServerState	LocaleID
Bandwidth	OPCGroups	PublicGroupNames
ServerName	ServerNode	ClientName

OPC Server - summary of methods		
Item	ShowBranches	ShowLeafs
MoveUp	MoveToRoot	MoveDown
MoveTo	GetItemID	GetAccessPaths

OPC Server - summary of events		
ServerShutDown		

Table 4.1. OPC Server object summary

OPC Browser - summary of properties

Organization	Filter	DataType
AccessRights	CurrentPosition	Count

OPC Browser - summary of methods		
Item	ShowBranches	ShowLeafs
MoveUp	MoveToRoot	MoveDown
MoveTo	GetItemID	GetAccessPaths

Table 4.2. OPC Browser summary

OPC Groups – summary of properties		
Parent	DefaultGroupIsActive	DefaultGroupUpdateRate
DefaultGroupDeadband	DefaultGroupLocaleID	DefaultGroupTimeBias
Count		

OPC Groups – summary of methods		
Item	Add	GeTitemGroup
Remove	RemoveAll	ConnectPublicGroup
RemovePublicGroup		

OPC Groups - summary of events		
GlobalDataChange		

Table 4.3. OPC Groups collection summary

OPC group - summary of properties		
Parent	Name	IsPublic
IsActive	IsSubscribed	ClientHandle
ServerHandle	LocaleID	TimeBias
DeadBand	UpdateRate	OPCItems

OPC group - summary of methods		
SyncRead	SyncWrite	AsyncRead
AsyncWrite	AsyncRefresh	AsyncCancel

OPC group - summary of events		
DataChange	AsyncReadComplete	AsyncWriteComplete
AsyncCancelComplete		

Table 4.4. OPC Group object summary

OPC Items- summary of properties		
Parent	DefaultRequestedDataType	DefaultAccessPath
DefaultIsActive	Count	

OPC Items - summary of properties		
Item	GeTitemItem	AddItem
AddItems	Remove	Validate
SetActive	SetClientHandles	SetDataTypes

Table 4.5. OPC Items collection summary

OPC Item - summary of properties		
Parent	ClientHandle	ServerHandle
AccessPath	AccessRights	ItemID

IsActive	RequestedDataType	Value
Quality	TimeStamp	CanonicalDataType
EUType	EUInfo	

OPC Item - summary of methods		
Read	Write	

Table 4.6. OPC Item object summary

When OPC turned out to be very popular, many vendors of automation devices have realized that working with OPC Automation require large effort to make anything more complicated than a simple read or write operation. Some of system's integrators do not need anything sophisticated and rather prefer to use ready-made office software. Many people looked for ActiveX solutions because it can be simply used in applications like Microsoft Office. It is easy to find lightweight OPC ActiveX components on the internet. Many of them are available for free, and some are sold as commercial products. There are also ActiveX toolkits available that claim to make OPC implementing fast and effective. People trying to use these ActiveX toolkits found they were limited, and went back to the automation wrapper, because it didn't constrain them in the same way.

The complexity and quality of all those solutions varies, but usually range of their functionality is limited. Sometimes their quality is very poor, it is why a common impression is that ActiveX OPC client components are not really worth bothering with, and are little or no value. Of course, in some cases, typically when fast prototyping is required, using them is the most effective solution. Furthermore, the automation wrapper is difficult to use when you need to read and write OPC data with ASP pages, scripting languages such as VBScript, or with VBA-enabled software tools such as Microsoft Excel. For the more detailed description of how to use such ActiveX component in Excel refer to *Chapter 6.8*.

There are also a few libraries which allow developing Java applications with the ability to read and write values from / to any device supported by an OPC server. Unfortunately, in most cases, those solutions will run on the Windows platform only. It is because an underlying DLL file is needed to interact between the OPC server's COM interface and the java library.

4.3. OPC on Windows CE

Nowadays PDA devices have become part of our daily lives. Similarly to industrial PCs they also went down to the factory floor. They have become especially popular in warehouses, wholesalers and large stores to help in stock management. Also in area of visualization and supervision control of the process they became used. With increase in popularity there was a need for the implementation of the OPC.

Due to the growing performance of modern computers, decreasing its size and popularity of large size flash EPROMs tablet PCs with Windows XP have become also very popular. Very often system integrators prefer to use them and do not take care after implementation of OPC.

There are numerous operating systems designed for PDA devices. The main three of them are Palm OP, Symbian and Windows CE. In this paper I will describe only OPC implementation on Windows CE. For other operating systems approach described in *Chapter 0* should be used.

Windows CE is a scaled down version of Microsoft Windows intended for use in embedded systems. Many platforms have been based on the core Windows CE. It is the most popular operating system for PDAs and most of them have it built-in.

Windows CE should be not confused with Windows XP Embedded which is NT based system and in case of implementing OPC is exactly the same as for Windows XP. In fact, Windows CE is fundamentaly different then Windows XP and offers a very small footprint of its features. Windows CE is optimized for devices with minimal storage capacities – its core can have less than a few megabytes and in most cases the whole system is burned into ROM which greatly increases its speed. Additionally, it fulfills all real-time operating system requirements. It does not necessarily mean it offers a high throughput. Rather, it means it offers facilities which, if used properly, can guarantee that system will fulfill specified requirements for the response time and reliability.

Generally, Windows CE was created as a ‘Windows like’ operating system. Its API is similar to Windows NT but many interfaces are missing. Due to the differences in hardware platform and User Interface it can be difficult to port a Windows XP application into Windows CE, but it is easier then implementing

Linux or OS/2 code. The good news it that there is a good support for this system and numerous off-the-shelf applications is already available.

Talking about OPC it is need to specify that generally systems based on Windows CE core mostly do not support DCOM technology. This makes difficult to implement OPC on those platforms. It is possible to find solutions which enable COM/DCOM support for Windows CE. Due to different requirements and hardware performance platform developers who wish to add COM runtime support to their Windows CE platforms can choose between two solutions which differ in the degree of complexity *[4.3]*.

A minimal implementation supports only in-process COM servers. It means that you have to load an appropriate DLL on client's device to communicate with it. This concept was described in *Figure 4.4.* As it was described in *Chapter 3.3.1* for in-process OPC servers communication between the client and the server is simplified because no remote procedure calls are required. The server created on Windows CE can have the same interface as regular OPC server, so this does not really create any interoperability issues. This implementation of COM is available with Windows CE 2.0, Windows CE 2.12 and newer.

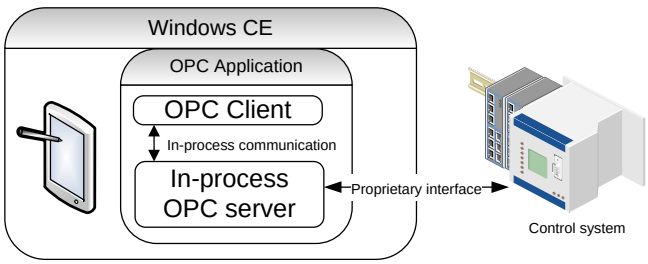


Figure 4.4. OPC in-process server implementation on Windows CE platform.

A richer COM implementation supports in-process and out-of-process servers, remote operation, Distributed COM (DCOM). Functionality is virtually identical to that provided in Microsoft Windows NT 4.0 with Service Pack 5 (SP5). This implementation is available for platform developers as a module that can be optionally included in Windows CE 3.0 and newer. Due to the security reasons it was removed from Windows CE 6.0. However, there is still an add-on pack which can be installed, but it is strongly recommended to use it only for closed networks, like for example at the factory floor level *[4.4]*.

Besides the lack of COM, there are other less important difficulties in the implementation of the OPC on Windows CE.

It is possible to find a few solutions which enable simple implementation of out-process OPC servers and clients into Windows CE *[4.5, 4.6]*. They do not require a deep knowledge about COM and OPC technologies. The concept of remote OPC server was described in *Figure 4.5*. Those additional DLLs are necessary to enable DCOM correctly marshal data exchanged between OPC server and OPC client processes. They are supplied by the OPC Foundation and usually installed when the first OPC client/server is installed on the PC. From client's PC point of view there is no difference between connecting to regular OPC server and server based on Windows CE.

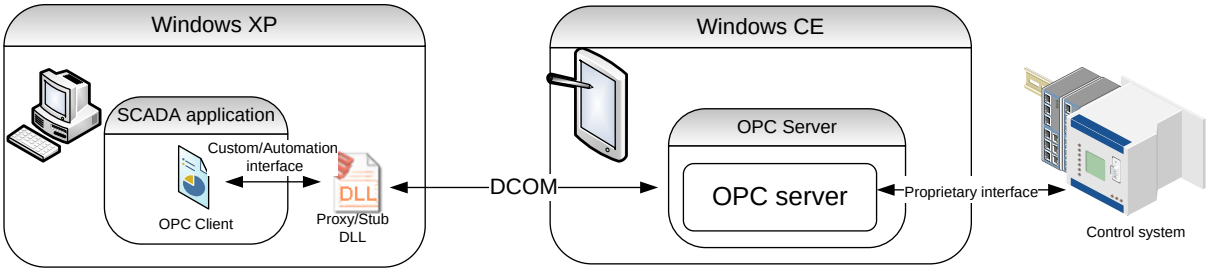


Figure 4.5. OPC out-process server implementation on Windows CE platform.

Of course it is also possible to implement in Windows CE the OPC specifications which do not require DCOM technology. Those are mostly based on XML technology (OPC UA, OPC XML), but unfortunately they are not so efficient and still not very popular.

To summarize Windows CE is a real-time system which provides good support for OPC technology. Due to the hardware platform restrictions, there are a few problems, but there are some off-the-shelf solutions which can be used for preparing self-made OPC server/client. The best advantage of it is a fact that most of existing code for Windows XP/98/NT can be reused and easy ported to Windows CE.

References:

4.1. http://www.opcware.com/WhatIsOPC.html (2008.10.10), *What is OPC*

4.2. OPC Foundation, *OPC DA Auto 2.02 Specification*

4.3. http://msdn.microsoft.com/en-us/library/ms834352.aspx, COM and DCOM in Microsoft Windows CE 3.0

4.4. http://blogs.msdn.com/cenet/archive/2008/01/04/dcom-remoting-on-windows-ce-6-0.aspx, Windows CE Networking Team WebLog

4.5. Northern Dynamics, OPC server toolkit for Windows CE

4.6. Softing, OPC toolbox

4.7. MatrikonOPC, OPC Tunneling Eliminates Setup Headaches

5. Simulation of manufacturing systems using the OPC standard

In this chapter, more unusual use of the OPC standard was described. In previous chapters the use of OPC for integration of industrial automation systems has been briefly described. Now, the possibility of using the OPC for dynamic process control and simulation will be investigated.

5.1. Methods of control systems developing

The design of control systems is multistage process. Some phases of the designing process are mandatory, others - for simpler applications - can be omitted *[5.1]*. Nowadays due to increasing number of software tools made to enhance development process it is easer to perform all required steps of prototyping. This approach is often called *Fast Prototyping*. Currently, we can observe trend to integrate all the tools needed on each stage of development into a common prototyping environment. Unfortunately, requirements are growing rapidly, plant installations are more and more sophisticated and the focus is placed on more effective control and faster prototyping. Simulation became essential step during control system development. A system where the control applications and man-machine interfaces of the real Distributed Control System (DCS) are connected to a simulation model of the process can be used for building, tuning and testing the control system implementation, and for training the operators, independently from the plant hardware.

Of course, usually control system consists of many heterogeneous subsystems, many processes are going simultaneously and it is difficult to model the relations between them, but to some extent they can be brought to such a simple model like it is presented in *Figure 5.1*. The input is represented by the blue block, which drives the controller, represented in green. The controller is connected to the controlled process, shown in yellow. Output values, represented in pink, can be read from the plant. Finally, this system includes a feedback loop, which feeds the plant output values back to the controller to adjust the controller output, making it a closed-loop control system.

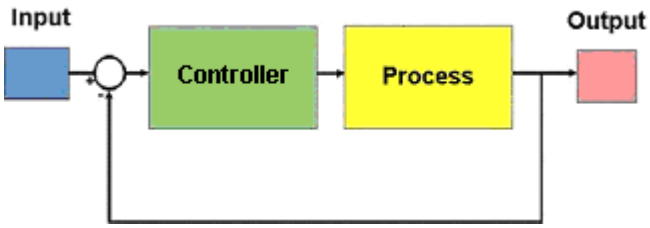


Figure 5.1. Control system architecture.

A typical path from project to its implementation in a real system is shown in *Figure 5.2*. On the first stage, model of a controller and a process should be prepared. With an appropriate model of the entire system optimization of the regulator settings can be done. The main advantage of simulating both the process and the process controller is possibility to run the simulation many times faster than it is going normally. It means that with optimization procedures optimal parameters of the controller can be found.

On the second stage combination of both real control equipment and simulated components is introduced to control loop. With such approach not only control algorithm can be tested but also the influence of the hardware restrictions (performance, delays, reliability.. etc). From the point of view of the controlling equipment, no difference should exist between controlling real process and its simulated equivalent *[5.2]*. Control-in-the-loop is a special variant of Hardware-in-the-loop concept. Hardware-in-the-loop has many advantages not only in case of time and money savings, but also in case of safety increasing. Below there are some of them indicated in order to justify the adequacy of this approach for control systems developing:

- It is possible to log process data which can not be observed on real object due to the fact it is impossible to measure it or there is no appropriate sensor,
- It is possible to verify the operation of the controller not only in the normal working conditions, but also in case of abnormal situations, for example: temperature too high and alarm should be indicated or motor emergency start procedure should be performed.. etc.

- Costs of stopping the plant, which is already working, to implement some changes is heavily reduced, due to the fact that changes can be tested on simulated process.
- The risk of damages caused by mistakes in the control algorithm is limited.
- It is possible to test the control algorithm while plant installation is being built.

On the last stage the results of previous steps are implemented in real plant installation. The new control system is being put into operation. Safety and performance test are executed, additional features (HMI) are added.

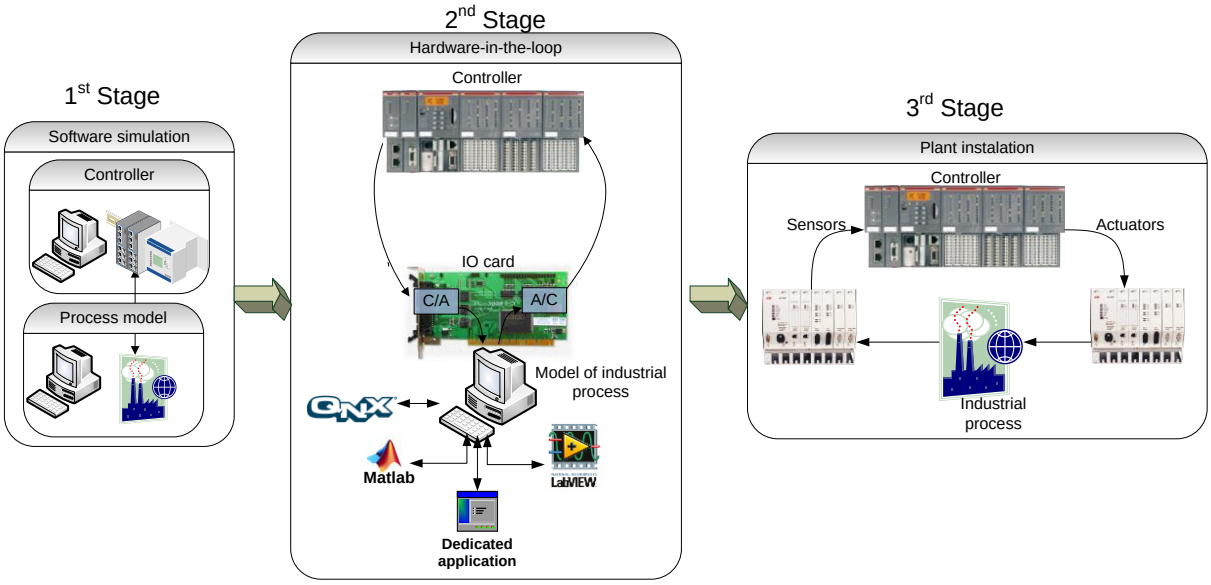


Figure 5.2. Steps of control system prototyping.

5.2. Using Matlab/Simulink for fast prototyping

As it was mentioned before, nowadays we can observe a trend to consolidate all tools needed for control system development into one package. Matlab was originally created to perform mathematical calculations for engineers, but gradually with development of so-called *toolboxes*, its functionality significantly increased. The milestone was enabling interaction with the hardware. Now, we can call Matlab complete Computer Aid Design (CAD) package for the rapid prototyping of control systems.

Pure Matlab has a command line programming environment. To perform the calculation commands are entered by the user sequentially into command line. The sequence of commands can be merged into a whole in form of M-file, which is some sort of a batch execution file. Additionally, Simulink was created as an easy to use, graphical programming environment for modeling, simulating and analyzing of dynamic systems. For modeling, Simulink provides a friendly graphical user interface (GUI). Models are being built as block diagrams, using click-and-drag mouse operations.

5.2.1. Matlab OPC Toolbox™

The Matlab OPC Toolbox™ is a set of functions which make able to connect with OPC server from Matlab/Simulink programming environment. For the experiments described in further part of this thesis only Simulink part of the OPC Toolbox is used, so the focus will be placed on describing only this part of the toolbox.

The OPC Toolbox block library contains four blocks: OPC Configuration, OPC Quality Parts, OPC Read, and OPC Write *[Figure 5.3]*.

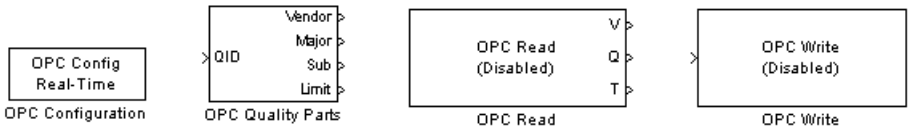


Figure 5.3. Matlab OPC Toolbox™ library

In order to communicate from Simulink with OPC servers, first you need to configure those servers in the simulation model. The OPC Configuration block manage Matlab OPC client for each Simulink model. In this block OPC server is selected *[Figure 5.4]*. It is important to set *Enable pseudo real-time simulation* option active. It will force Matlab to execute simulation in real-time. Of course, it is impossible if computer's performance is too low. In that case simulation clock will be slower than real-time clock. Matlab will show errors with information that real-time clock validation occurred.

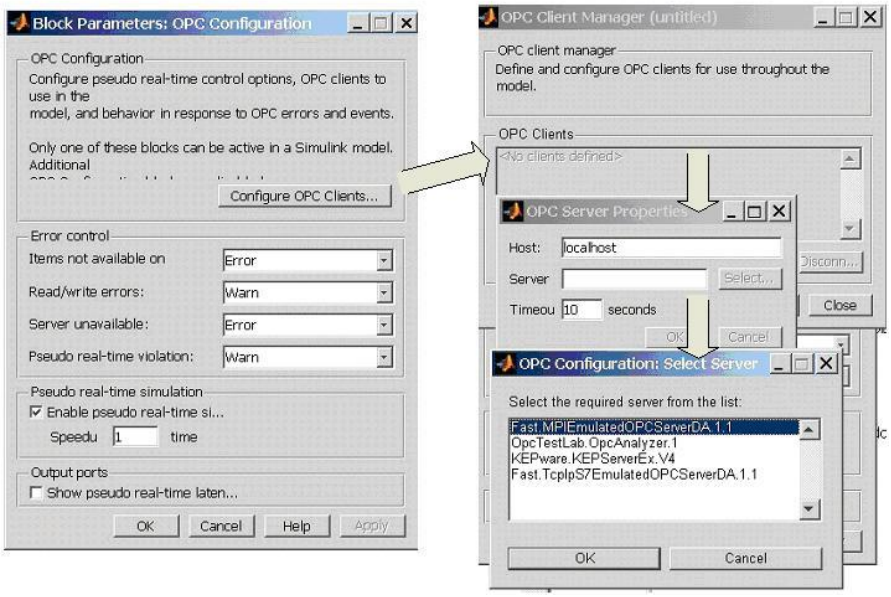


Figure 5.4. Simulink OPC toolbox configuration

To read from OPC server the OPC read should be used. This block has 3 outputs: read value, timestamp and quality. To configure the block you need to specify which items from OPC Server should be read, if more then one item will be chosen, output *Value* will be an array. With *Value port data type* it is possible to define the data type for the output *Value*. This type of data will be use further in the simulation. The OPC server is responsible for converting all data to the required type.

To configure the reading block it is also important to select the method how data will be captured from server. You can choose from 3 options (for more details refer to *Chapter 4.2* of this thesis).

Synchronous read – MATLAB stops simulation and wait for the server to return data from a read request before continuing processing. The data returned by the server can come from the server's *cache*, or you can request that the server read values directly from the *device* that the server item refers to. Reading from the device can be slower but data are more time accurate.

Asynchronous read – MATLAB sends a request to read data from the server. Ones the request is accepted it continue processing without waiting to receive any values from server. Server will use callback to inform MATLAB about new data occurrence. MATLAB will handle that event as soon as it is able to perform that task. Asynchronous read operations always return data from the device.

Data writing is realized with the OPC write block. It also can perform synchronous and asynchronous operations. Similarly for OPC write block it is possible to choose the writing method:

- Synchronous write – MATLAB sends a request to write data, stops simulation and wait for acknowledge information that data was successfully written to the device. Because writing is always done directly to the device, synchronous operation can take a significant amount of time in case the device has slow connection to the OPC server.
- Asynchronous write – MATLAB creates a request to write data, and then sends that request to the OPC server. Simulink continue simulation processing without waiting for server acknowledge that data has been successful delivered to the device. Server generates interrupt to inform Matlab about writing succeed or fail. Matlab will handle this event as soon as it is able to perform that task. In case of writing failure message is generated.

Both for reading and writing *Sample time* (item update rate) has to be specified. Sample time determinates minimal time period for data updates.

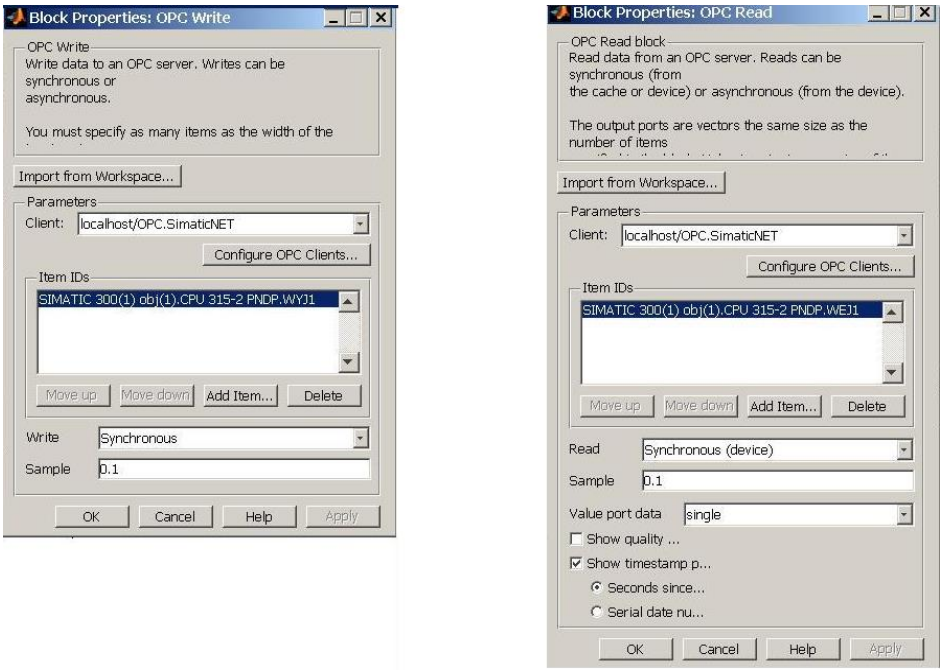


Figure 5.5. OPC write and OPC read block configuration

5.2.2. Other solutions

Using Simulink is only one of the possibilities for control systems prototyping. There are many other software packages that can be used for this purpose. The reasonable solution appears to be the use of LabView. It offers huge support for the various peripheral devices. Of course, it has also possibility to cooperate with OPC servers. According to documentation of LabView *[5.5]* with *Control Design and Simulation Module* it is possible to:

- Construct plant and control models using transfer function, state-space, or zero-pole gain,
- Analyze system performance with tools such as step response, pole-zero maps, and bode plots,
- Simulate linear, nonlinear, and discrete systems with a wide option of solvers,
- Deploy dynamic systems to real-time hardware using built-in functions, LabVIEW Real-Time Module,

Another solution is to write a dedicated application to simulate the process. With OPC wrapper it is easy to develop the OPC interface for our own application. More serious problem is how to perform process simulation. More detailed description of this approach is described in *Chapter 6.4.1*.

References:

5.1. Wojciech Grega (2004), *Metody i algorytmy sterowania cyfrowego w układach scentralizowanych i rozproszonych*

5.2. Michał Brewiński (2008), *Industrial Ethernet: Applications for Process Control*

5.3. http://en.wikipedia.org/wiki/Hardware-in-the-loop (2008.11.10), Hardware-in-the-loop simulation - Wikipedia Free Encyclopedia

5.4. Mathworks, *Matlab/Simulink OPC Toolbox™ 2 User's Guide*

5.5. http://sine.ni.com/nips/cds/view/p/lang/en/nid/203826 (2008.11.10), NI LabVIEW Control Design and Simulation Module

6. Experiments

6.1. Methodology of experiments

The main subject of this Thesis is to analyze possibility of using the OPC standard for a dynamic process control purpose. In following experiments I was trying not only to inspect the methods of control system controlling via OPC, but also possibility to use OPC during process of fast prototyping.

The aim of the most of experiments was to compare control quality in traditional control systems based on PLC and control systems in which current value of control signal is calculated by PC and process is connected to PC via OPC protocol. In this configuration, PLC is used only as a gateway between the PC and the controlled process *[Figure 6.15]*. With these tests there was an attempt to predict the impact of presence OPC in control loop for process control. Detailed model of each configuration is prepared in Matlab/Simulink environment. Simulation results are compared with results of real experiments.

Because there have been no suitable process available to use in the laboratory, process simulator has been used during those tests.

For all tests the experiment's length was constant and equal to T_{EXPER}=100s.

Because the PC computer performance has big influence on a control quality, all tests have been conducted on the same hardware platform. The computer used during the test was a PC class notebook with:

- Windows XP SP3
- Matlab / Simulink R2006b
- Intel Centrino 1,4 GHz CPU
- 750 Mb RAM memory
- 1Gbit Ethernet card
- 11MBit WiFi adapter

6.2. Digital Process Simulator

6.2.1. Device description

In a very first stage of the experiments, Digital Process Simulator (DPS) from Anerma *[Figure 6.1]* have been used as a simulation of the real controlled process. This device is very simple device, but it has complex capabilities.

Figure 6.1. Digital Process Simulator – real picture

Its brief characteristic can be described with following points:

- The device has two current inputs (4..20 mA), one of them is a an inverting input. The current signal is translated to digital signal 0..100%. Translation is made by 10 bits A/D converter, there is also digital filter for 50 Hz to damp down influence of power network,
- The device has one current output (4..20 mA) which can be simple translated to voltage output (1..5 V) with 250Ω resistor. Digital to analog translation is made with 12 bits D/A converter,
- There is possibility of setting backlash for input signal (0..20%),
- It is also possible to compensate input nonlinearity,

$$y(t) = u(t)^{NL}, \qquad (6.1)$$ NL = 0.5..2.

- Device can simulate a third order inertial process with different time constants,

$$G(s) = \frac{1}{(T_1 s + 1)(T_2 s + 1)(T_3 s + 1)} \quad (6.2)$$ T_1, T_2 ,T_3 = 0...100s with 0.1s accuracy.

- It is possible to integrate process response. If $T_I > 0$ signal is integrated:

$$y(t) = \frac{1}{T_i}\int u(t)\cdot dt \quad \text{or} \quad y(s) = \frac{1}{T_i s}u(s), \quad (6.3)$$ T_i=0.1...100s with 0.1s accuracy

- Process response time-lag can be set. According to results of the experiments apart from the time-lag value the Digital Process Simulator has about 600ms of conversion time

$$y(t) = x(t - T_L) \quad \text{or} \quad y(s) = x(s)e^{-jsT_L}, \quad (6.4)$$ T_L=0.1...100s with 0.1s accuracy

- Parameter K_P describes the signal amplification. It amplifies not the output signal but distinction of the output signal and 50%.

$$y(t) = K_P(x(t) - 50) + 50, \quad (6.5)$$ K_P=0...5

Its internal structure shows *Figure 6.2*. Backslash for input signal is simulated by *Dead Zone* block. Signal goes through a non-linear component, three first order inertial blocks. When $T_i \leq 0$ integration block is omitted in other case integration of output signal is performed. Device processing time is simulated with a *Delay* block, additionally this delay can be increased. The process zero point is set to 50% of output signal so at the very end constant value 50 is added output signal.

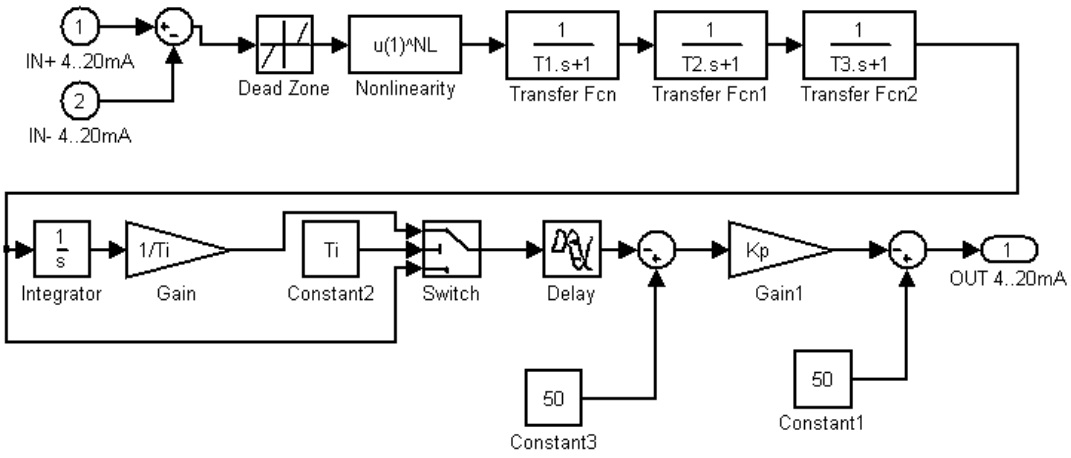


Figure 6.2. Digital Process Simulator – Matlab/Simulink model

6.2.2. PLC and Digital Process Simulator connection

In described experiment Siemens S7 300 with CPU 315 has been used. It was connected with Digital Process Simulator by current output (4..20mA) of 12 bits D/A converter and voltage input (1..5V) of 12 bits A/D module *[Figure 6.3]*.

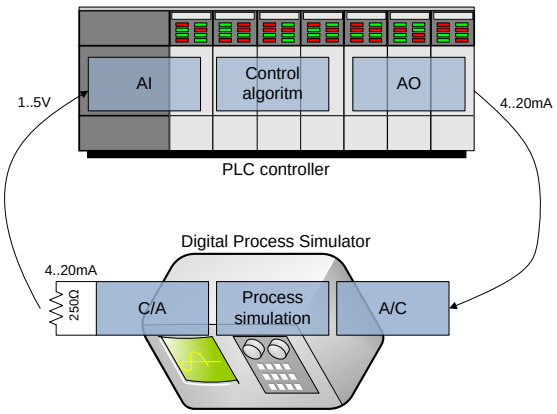


Figure 6.3. PLC and Digital Process Simulation connection

In the very beginning signal range and linearity of the I/O connection have been tested. There was a resistance 250Ω connected to the output of process simulator to convert current signal into voltage signal. This resistance could be a bit different than it should be, due to the uncertainty of resistor fabrication. For testing PLC's output values from 0..100% (0..27648) of the output signal range were written to PLC's I/O image. The results were measured on LCD screen of Digital Process Simulator. Similar procedure has been used for testing PLC input. Finally, it was assumed that both PLC input and output are linear.

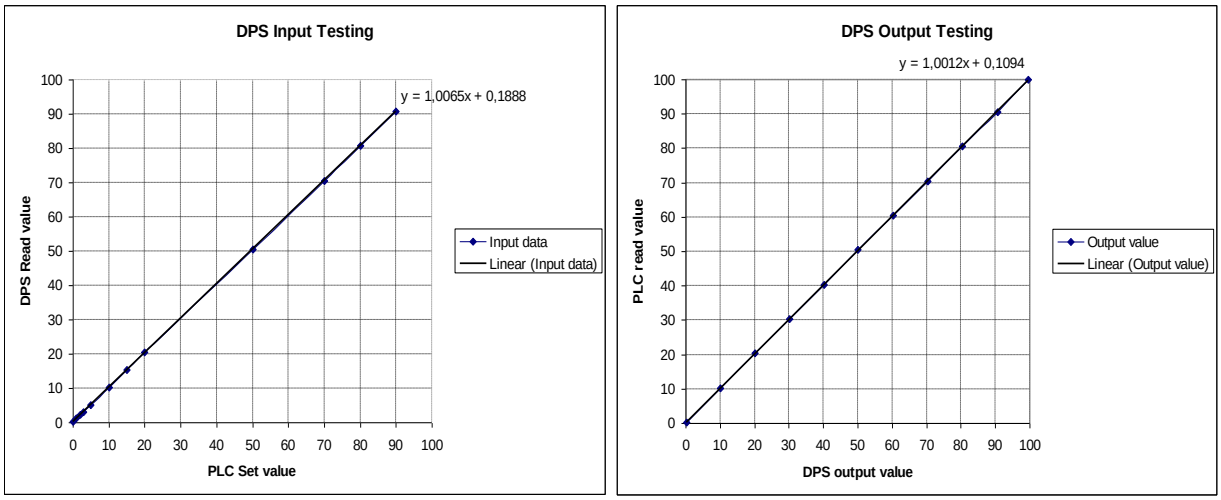


Figure 6.4. Digital Process Simulator input and output testing

6.2.3. Digital Process Simulator response time-lag

In the next step of the experiment Digital Process Simulator response total delivery time has been observed. The configuration shown in *Figure 6.5* describes idea with the test. Matlab OPC client to output of PLC sent step signal. Input and output signal were scaled to 0..100% with the gain blocks. Matlab was generating step function and transmitting it to PLC's output with OPC. In this configuration PLC was only a bridge between PC and process Step amplitude was set to 80%. Process Simulator's response was read with OPC Read block. Both reading and writing was made with OPC synchronous blocks.

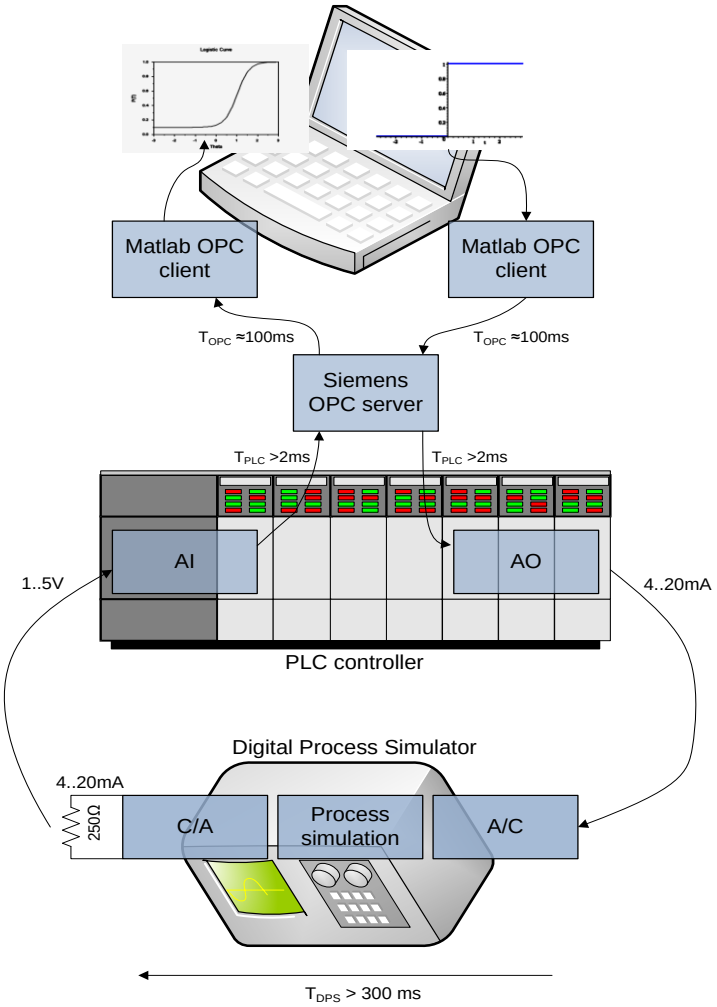


Figure 6.5. Process simulator total delivery time measuring idea

Two test has been performed for a different time-constants of Process Simulator. In both tests Digital Process Simulator was configured as a pure

inertial object (*Figure 6.2* - T_L=0, T_i=0, K_L=1, NL=0) its transfer function is described in *Equation 6.6*

$$G(s) = \frac{1}{(T_1 s + 1)(T_2 s + 1)(T_3 s + 1)}, \quad (6.6)$$

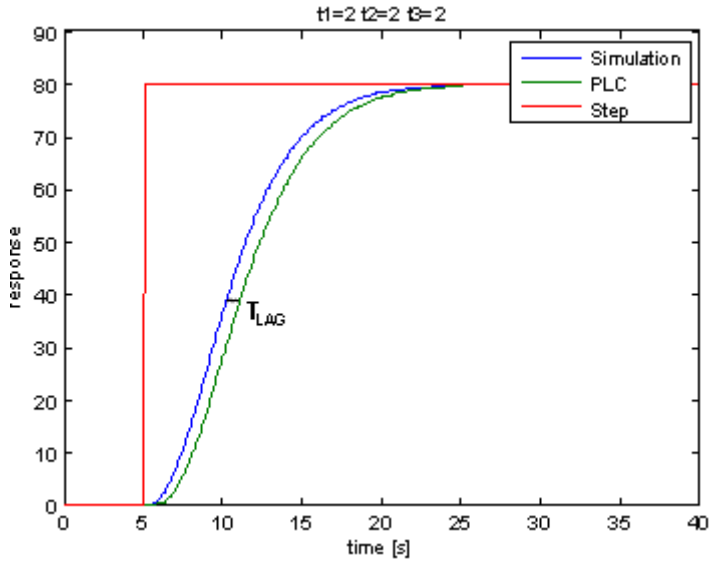


Figure 6.6. Step response of process with T_1,T_2,T_3=2s

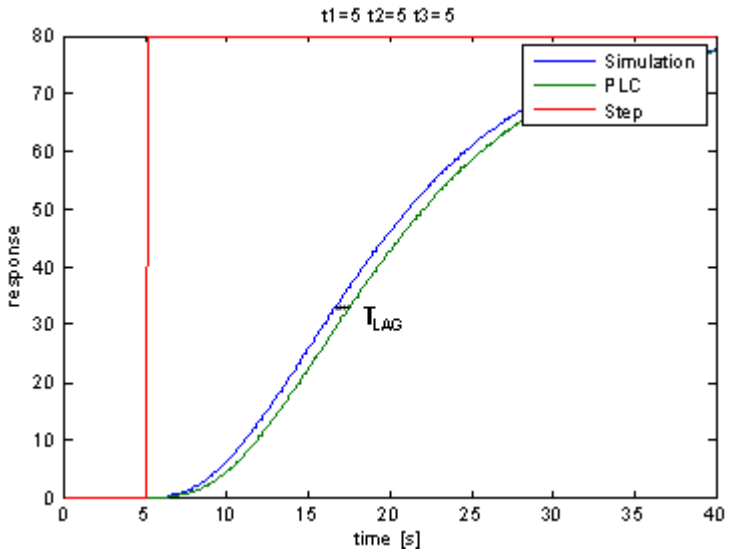


Figure 6.7. Step response of the process with T_1,T_2,T_3=5s

Process simulator's step responses have been shown on Figure 6.6 and Figure 6.7. Simulation plot is a step response of the transfer function described in [Equation 6.6] and simulated in Matlab. PLC plot was captured by PC during the experiment. Theoretically the time difference between simulation's step response and real data captured from PLC should not be higher than:

$$T_{LAG} = 2T_{ITEM} + 2T_{PLC} + T_{DPS} < 200ms + 4ms + 200ms = 404ms\,, \quad (6.7)$$

where: T_{ITEM} - OPC item refresh time

T_{PLC} - PLC cycle time

T_{DPS} - Digital Process Simulator response time

In both experiments time delay between those two responses is close to 800ms. Author of this thesis cannot explain this fact for sure, but it is probable that there is mistake in T_{DPS} value. For further experiments author assumes that response time of the Digital Process Simulator is close to 600 ms (T_{DPS}=600 ms).

6.3. Experiment 1

6.3.1. Siemens PID block

For process control PID algorithm has been established on PLC. A standard PID controller FB41 (CONT_C), available in SIMATIC Step 7 libraries has been used for process control purpose. According to its documentation, Simulink model has been made *[Figure 6.9]*. All parameters of this model have been shown in *Figure 6.8*. The idea with the first part of Experiment 1 is to verify the accuracy of this model.

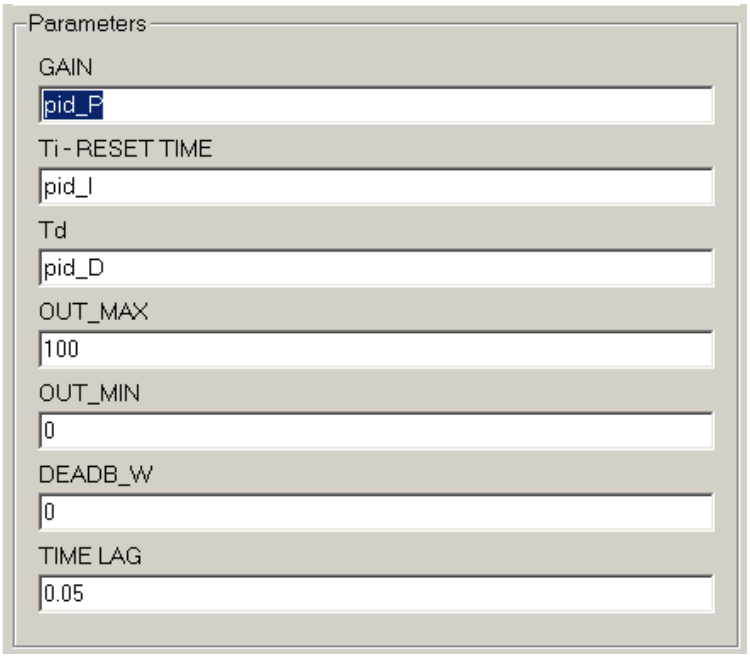


Figure 6.8. Simulink PID block parameters

The function block "CONT_C" is used on SIMATIC S7 programmable logic controllers to control technical processes with continuous input and output variables *[6.11]*. It has numerous inputs and outputs, but in simulation model only those which are necessary were implemented. For complete description of this block refer to *Figure 6.10*.

The proportional, integral, and derivative actions are connected in parallel and can be activated or deactivated individually. This allows P, PI, PD, and PID controllers to be configured. Pure I and D controllers are also possible. Output signal is limited to range OUT_MIN..OUT_MAX. If signal is out of the range anti wind-up filter is activate and error integration is stopped. Derivative action is not implemented as a true derivative. There is a Time-Lag value which specifies the settings of first order inertia on derivative action value. Time Lag value cannot be lower then half of PID block refresh cycle time.

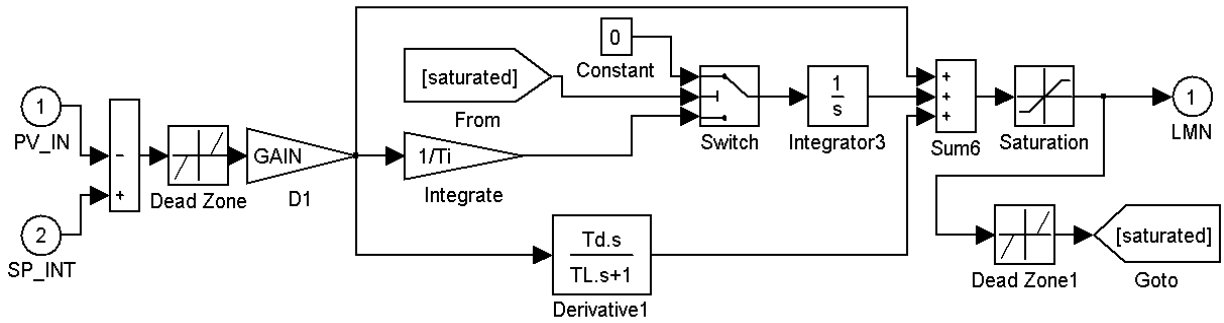


Figure 6.9. Simulink model of Siemens' *CONT_C* PID block

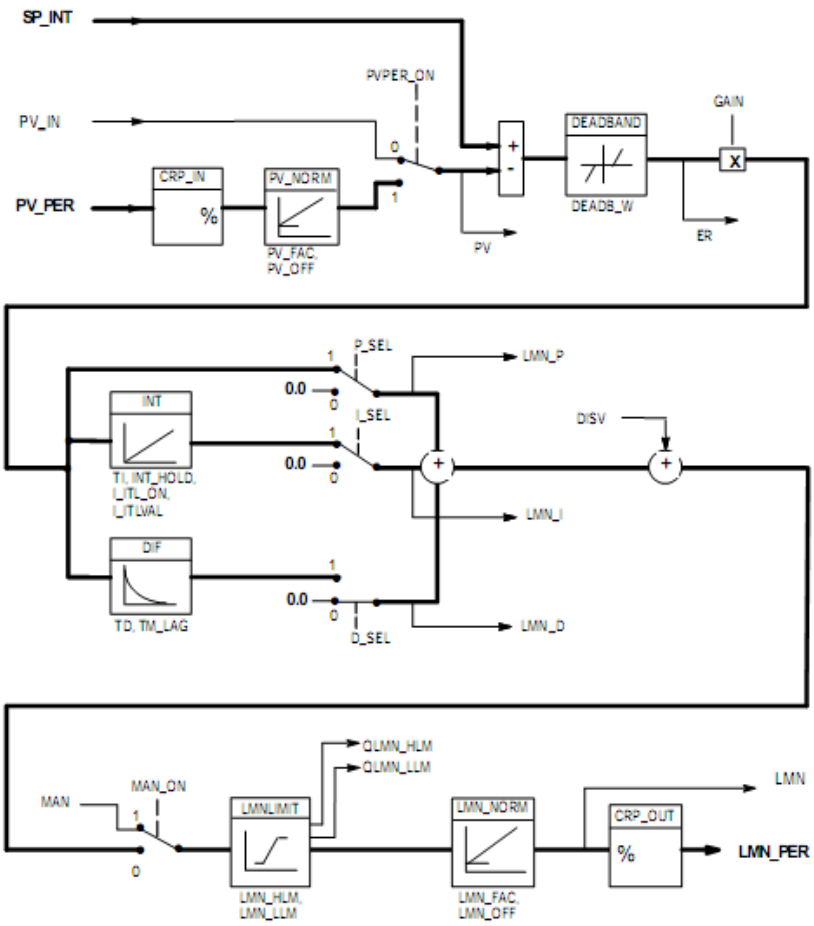


Figure 6.10. Block Diagram of CONT_C PID controller *[6.11]*

6.3.2. Finding out optimal parameters for PID block

Having Matlab/Simulink models of PID controller and Digital Process Simulator the close loop system behavior has been simulated. Appropriate Simulink model with PID block and Digital Process Simulator have been created *[Figure 6.12]*. During a single PID block cycle time, signal input and output values are constant. To simulate it zero-order hold blocks has been placed before and after PID block.

In the second step, optimal parameters of the regulator have been determined. To find out optimal PID settings Adjusted Model Technique (AMT) has been used. This method is trying to find minimum of the control quality function *[Equation 6.8]* by changing the controller's parameters.

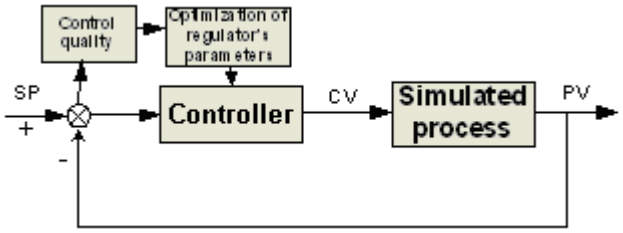


Figure 6.11. Optimization of regulator's parameters with Adjusted Model Technique

Control quality was estimated as integral of error's absolute value. Because of the signal's sampling integration has been replaced by the sum.

$$control\_quality = \sum_i \left| SP_i - PV_i \right|, \quad (6.8)$$

SP - Set point
PV – Process Value

According to description of AMT for each PID settings the control quality is evaluated. Matlab's *fminsearch* function has been used to calculate PID settings with minimal quality coefficient. For more details refer to the attached in Appendices source code of Matlab *[Appendix 8.2]*.

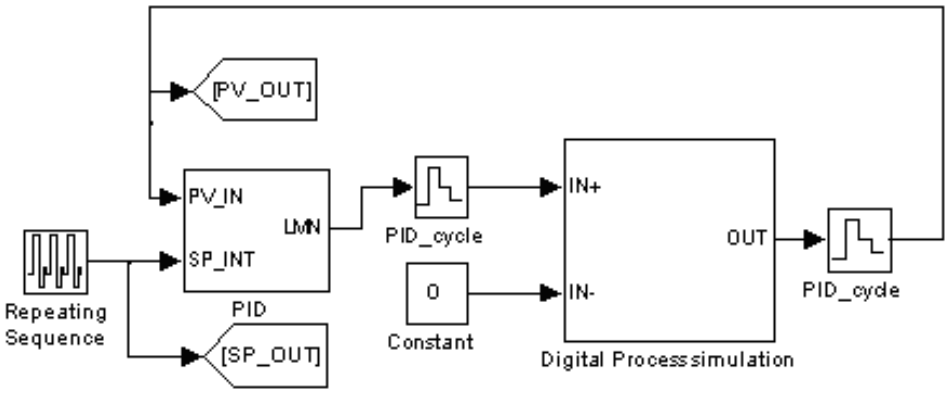


Figure 6.12. PLC based control system model.

6.3.3. Control by PLC

The controlled process transfer function was described as a 3rd order inertial object with time constants T_1,T_2,T_3 = 2s. Additionally delay associated with Digital Process Simulator T_{DPS}=600 ms is introduced into the control loop.

The close loop system has been set up as it is described in *Figure 6.13*. During the experiment Set Point and PID settings were transferred to PLC by OPC, Process Values were read from PLC and saved to the Matlab's workspace. The PID block in PLC was called each 100ms period. Results of real PLC control

and simulation were compared on a plot and control quality coefficient was calculated for each of them *[Figure 6.14]*.

Simulation's results are strongly related to method of numerical integrating differential equations. For Runge-Kutta (RK4) method real and simulation results are pretty similar, for Euler method some parts of process' control are significantly different *[*Figure 6.14*]*.

The time gap between RK4 simulation plot and PLC plot is related to OPC item refresh time and it should not be bigger than $2T_{ITEM}$ (200ms). In fact, *Figure 6.14* shows that this time varies between 200ms – 400ms. This can be associated with longer OPC item refresh cycle (T_{ITEM}) than author predicts, but probably it is related with wrong Digital Process Simulator model in Matlab. It is also probable it is a problem with badly conditioned simulation's differential equations (that is probably why Euler and RK4 simulations results are different).

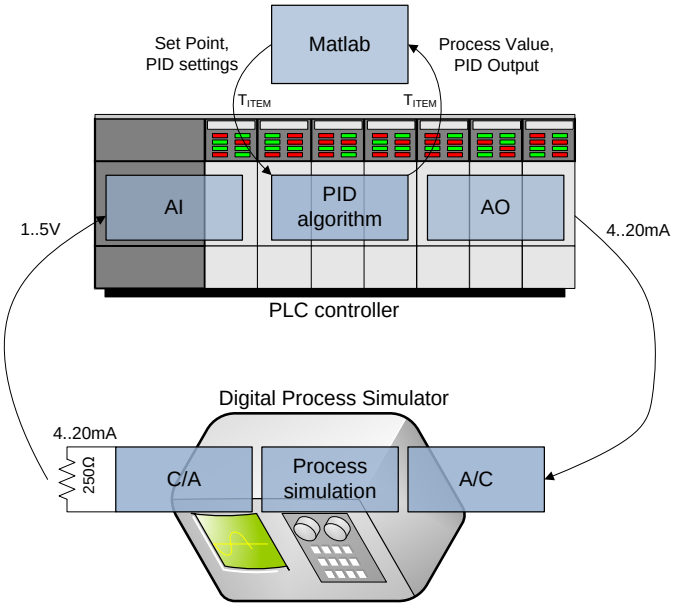


Figure 6.13. Experiment 1 – process controlled by PLC

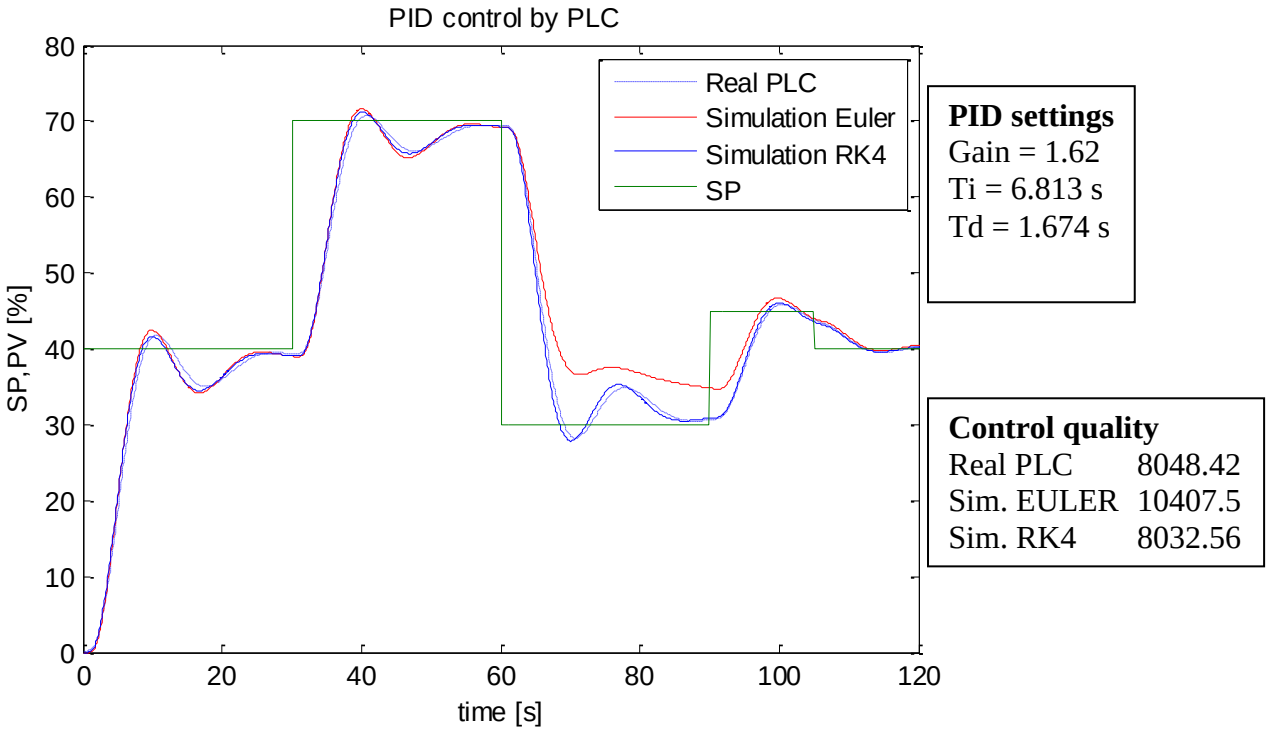


Figure 6.14. Experiment 1 – control by PLC results comparison

To compare control quality integral of error's absolute value was used *[Equation 6.8]*. Time lag ($2T_{ITEM}$) between PLC and Simulation plots has been removed

6.3.4. Control by Matlab PID

In this part of the experiment control algorithm was placed in Simulink. 'CONT_C' block simulation model *[Figure 6.9]* was used for this purpose. Process value was read via OPC and process control value was calculated real-time in Simulink and sent to PLC in next OPC server cycle. Exactly the same like in previous part PID settings were used.

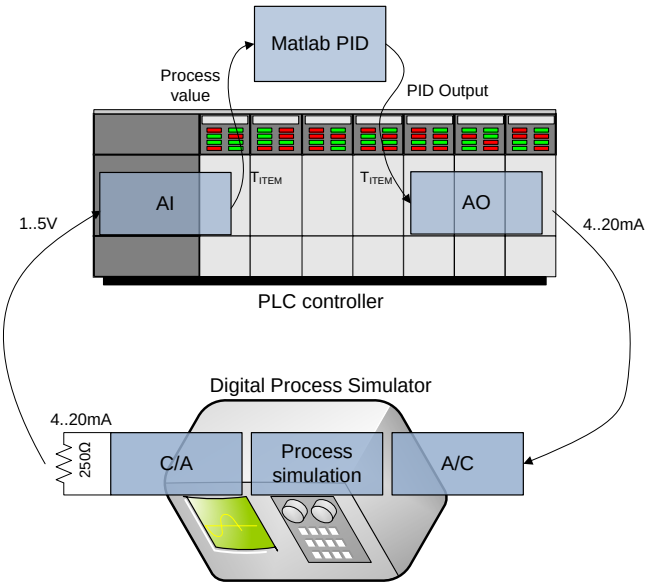


Figure 6.15. Experiment 1 – process controlled by Matlab PID

The Matlab based process control simulation model *[Figure 6.16]* is a bit different than simulation model for PLC based control *[Figure 6.12]*. OPC item refresh delay was added between the PID block and Digital Process Simulator. Control signal value is written to PLC with synchronous block each OPC refresh item time. After OPC server refresh time process value is being read with asynchronous block. OPC Server refresh rate is specified in server's configuration and determine the smallest possible interval for checking OPC item value. Default server's cycle time T_{SERVER} for Siemens is 100ms. OPC item refresh rate is specified in OPC Client and should be multiplication of OPC Server cycle time.

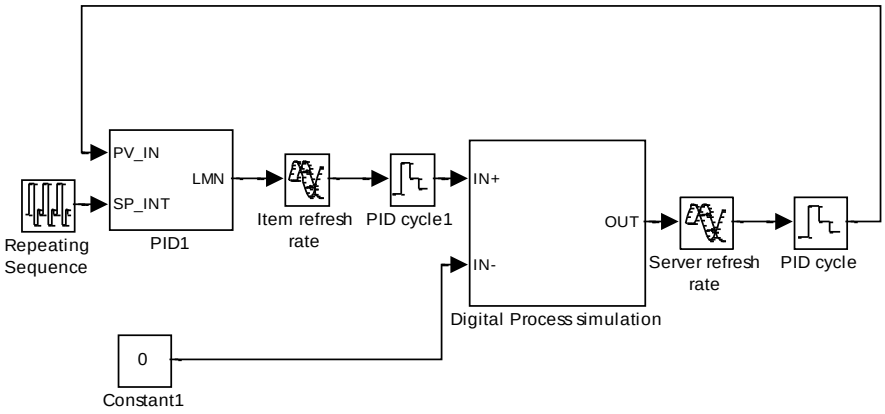


Figure 6.16. Simulink model of system controlled by MATLAB.

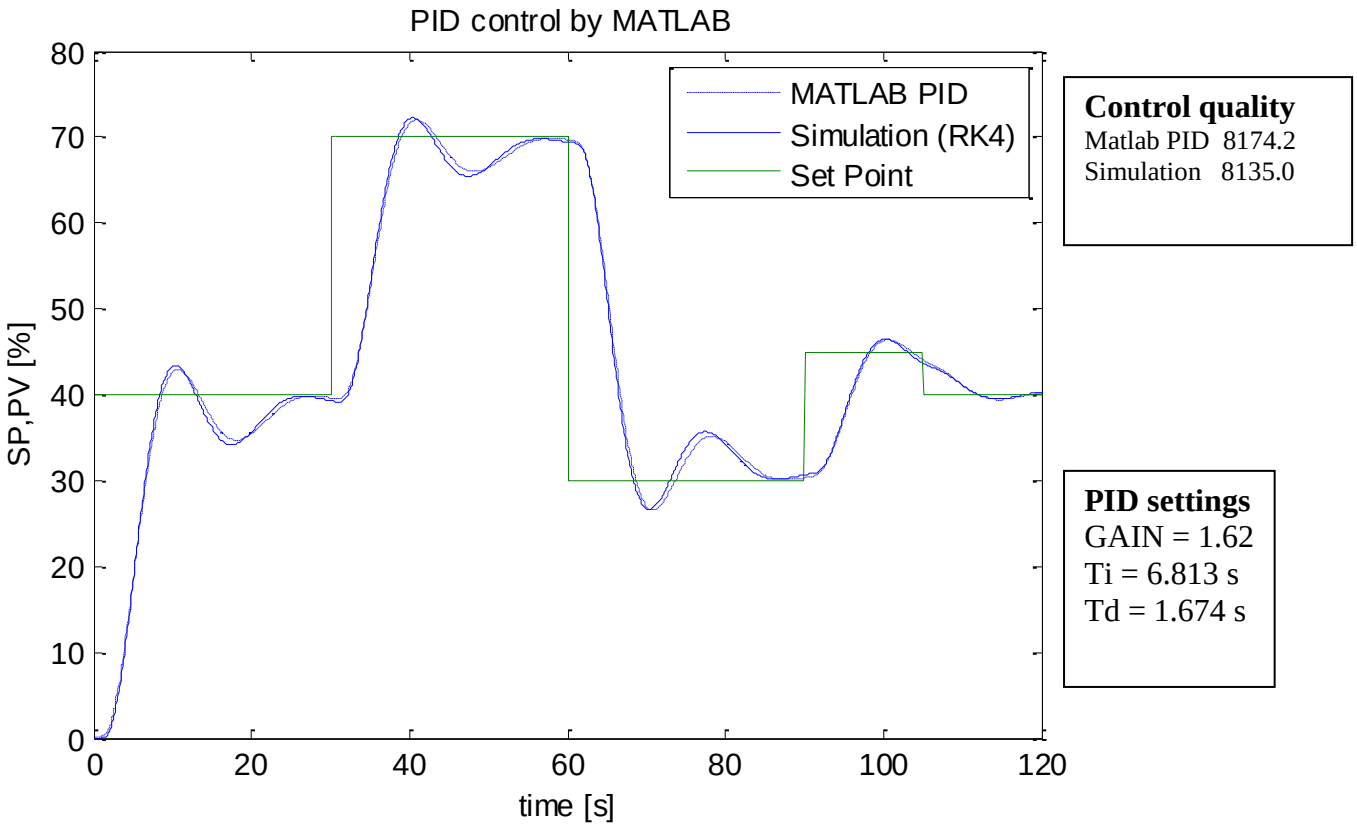


Figure 6.17. Matlab PID control results

Control quality by Matlab PID is worse then it is in traditional control systems - the control quality coefficient *[Equation 6.8]* is higher. As it was mentioned before OPC introduce additional OPC delay into control loop. Of course, there are numerous methods to compensate this delay (for example the Smith's predictor method described in *Chapter 6.7*). It is also possible to settle up more complicated control algorithm on PC.

6.4. Experiment 2

6.4.1. DAPIO card

Fortunately, for my experiments DAPIO300 – Data Acquisition Processor was available in the laboratory where the experiments were performed. This card was manufactured by Microstar Labolatories. Although, it is an old solution, based on the obsolete ISA connector, it offers pretty nice features. It has on board operating system dedicated for 32-bits operations, 486 family processor and embedded DSP routines. The interaction with an environment can be done by using 16 digital inputs, 16 digital outputs, 16 analog single ended inputs (8 differential) and 2 analog outputs. Thanks to efficient embedded processor it offers high sampling rate (at least 10 kHz, according to the manual). Additionally it has large cache memory onboard, so it is possible to capture the signal for long period with high frequency.

It took a lot of afford to manage this card running, the biggest problem was configuration of ISA interface, but finally it was running. With using DLLs provide by the card vendor it was possible to write in ANSI C dedicated software to simulate the process. Unfortunately, it was not possible to use embedded memory on the card so the sampling period was much longer than in documentation of the card, but it was still enough for this tests purpose.

Similar process to that which was used in *Experiment 1* was prepared, but now there is no Digital Process Simulator propagation delay (T_{DPS}), the DAPIO card react for input in time T_{DAPIO}<1ms. Runge-Kutta (RK4) method was used to simulate the process. The most important fragments of the application source code were copied to the *Appendix* section of this thesis.

The additional advantage of using the DAPIO card is possibility to capture the data both from the process controller (PLC, MATLAB PID) as well as from the process (DAPIO card). With using dedicated application for process simulation there user has also more detailed knowledge about how exactly process is simulated (differential equation solving method, sampling rate, processing time.. etc). All of this makes experiments more accurate.

6.4.2. DAPIO card response time-lag

First experiment with DAPIO card led to determining card's response time. Configuration similar to that shown on *Figure 6.5* was assembled and similar experiments to that described in *Chapter 6.2.3* were performed. Step signal was sent to process input and time between ideal response and DAPIO card was measured. Results show that for DAPIO card processing is very short in comparison to OPC item update time (T_{ITEM}). I can assume that time lag between the plots in *Figure 6.18* is equal to $2T_{ITEM}$. DAPIO card processing time is so little that in further experiments it will be not considered.

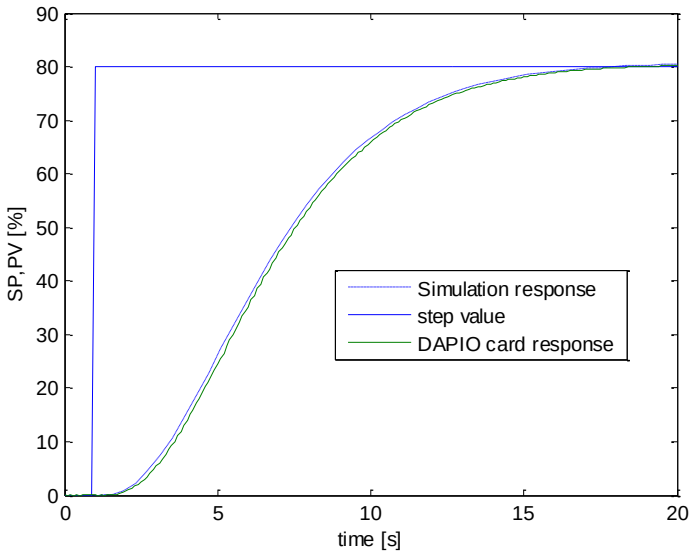


Figure 6.18. Comparison of ideal step response and response of the process simulated with DAPIO card.

6.4.3. Comparison of control by PLC and MATLAB

A PLC was used for controlling process simulation running on PC with DAPIO Card *[Figure 6.19].* Similarly to the previous experiment the controlled process in this part can be described as a 3rd order inertial object with time constants T_1,T_2,T_3 = 2s, but in this case there is no delay associated with process simulator's processing time. As it has been proven in *Chapter 6.4.2* processing time of DAPIO card is too small to be considered in the simulation.

Again optimal PID settings have been calculated with using Adjusted Model Technique. In this experiment simulation of system controlled by Matlab *[Figure*

6.16] was compared with results of the process control with real PLC. The accuracy of the close loop system model in this case was much better then in case of the experiments described in the previous chapter. As it can be seen in *Figure 6.20* the difference between simulation plot and process controlled by Matlab is very small. It means, to some extent, that the presented simulation model of the close loop system with OPC can be used for prototyping the control systems.

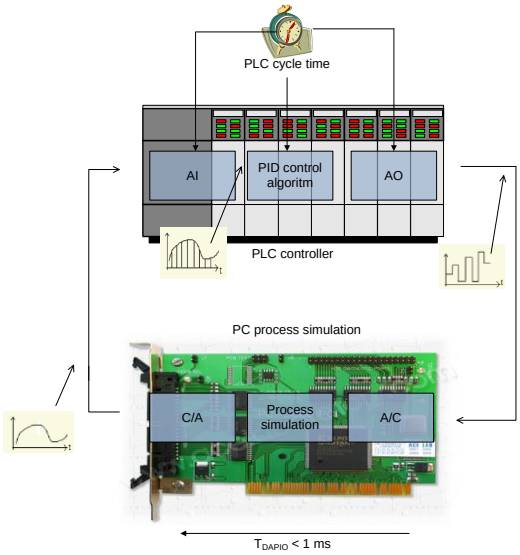
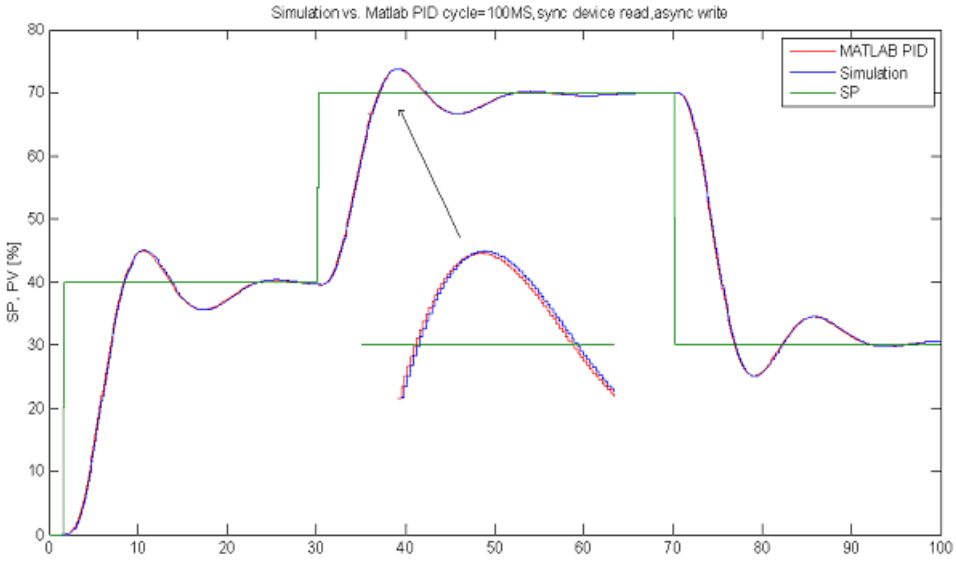


Figure 6.19. Control system with DAPIO card.



PID settings

GAIN = 1.38

Ti = 6.207 s

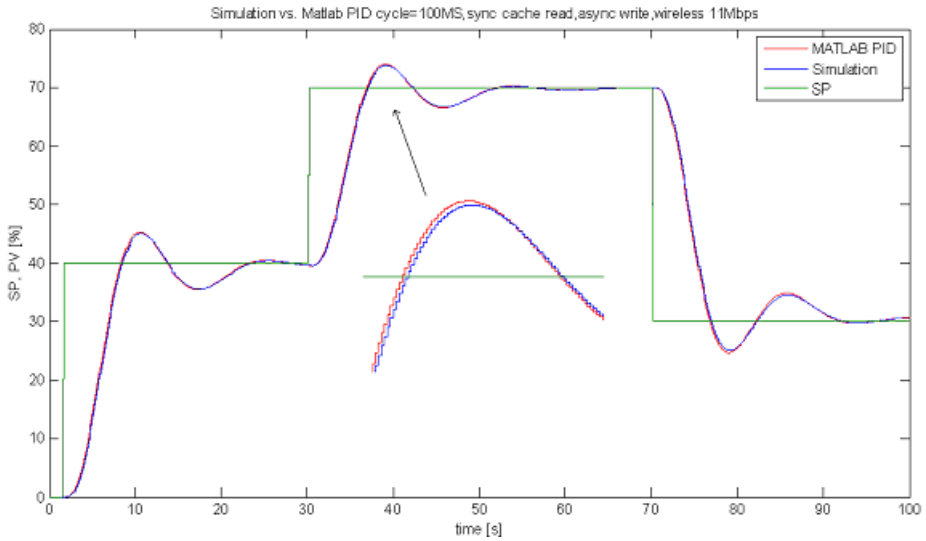
Td = 0 s

Control quality

Matlab PID 5489,8

Simulation 5491,8

Figure 6.20. Comparison of simulation results and control by Matlab PID – the best result, T_{ITEM}=100ms, synchronous write device, synchronous read.

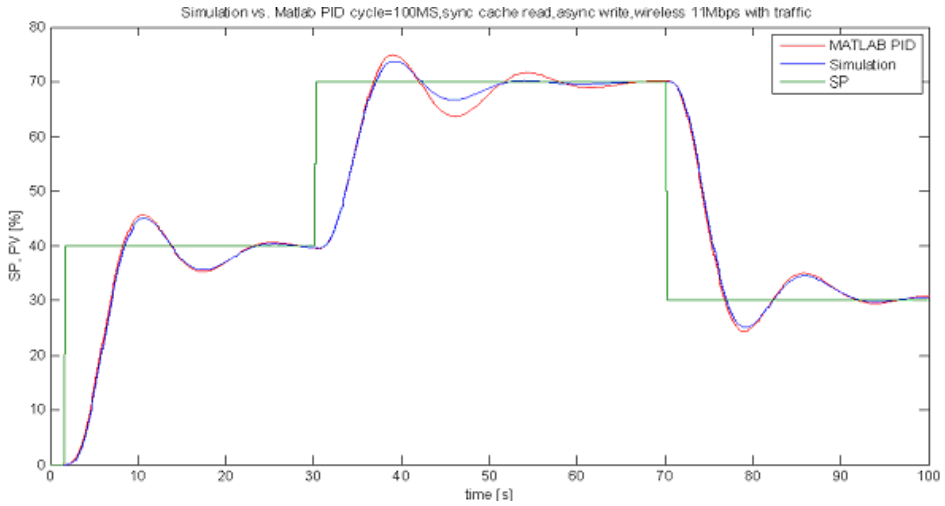


PID settings
GAIN = 1.38
Ti = 6.207 s
Td = 0 s

Control quality
Matlab PID 5566,5
Simulation 5491,8

Figure 6.21. Comparison results of simulation and control by Matlab PID – wireless without traffic.

In further part of this experiment 100MBps Ethernet connection has been replaced by regular Ethernet wireless connection with 11MBps bandwidth.



PID settings
GAIN = 1.38
Ti = 6.207 s
Td = 0 s

Control quality
Matlab PID 5921,7
Simulation 5491,8

Figure 6.22. Comparison results of MATLAB simulation and control by Matlab PID – wireless 11Mbps with traffic.

Numerous tests have been done. They are summarized in form of tables. Control quality is calculated as a sum of absolute errors between process set point (SP) and process value (PV).

$$Control\_Quality = \sum_i \left| SP_i - PV_i \right| , \quad (6.9)$$

SP - Set point
PV - Process Value

With each data read from OPC time stamp values is associated. To show how traffic influences on OPC reading a time stamps vector has been used. Theoretically distinction in time between successive time stamps should be

equal to OPC item refresh rate (T_{ITEM}). In reality it varies depending on network and CPU load. *Figure 6.24* illustrates how Ethernet traffic impacts on control process. Histogram in *Figure 6.25* shows how real item refresh rate depends on network load and OPC configuration.

OPC quality is calculated as sum of absolute errors between OPC item read time stamps and OPC item refresh rate (T_{ITEM}) and divided by number of time stamps (6.10). It means the results of this calculation will show the average time deviation of single OPC item reading.

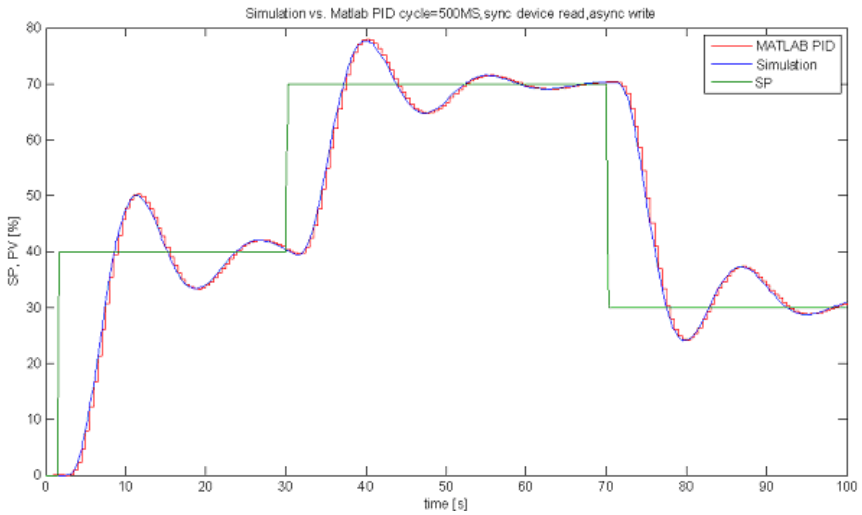
$$OPC\_Quality = \frac{1}{N}\sum_{i=1}^{N-1}\left|(TimeStamp_{i+1}\text{-}TimeStamp_i - T_{ITEM})\right|, \quad (6.10)$$

Network traffic and computer load strongly influence on OPC reading quality. Good indicator of OPC performance is also control quality coefficient. As it can be seen there is a huge distinction between process control quality by regular Ethernet cable connection and wireless with traffic.

Table 6.1. OPC item refresh rate T_{ITEM}=100ms, OPC server refresh T_{SERVER}=100ms.

Configuration	Test 1		Test 2		Test 3	
	Control quality	OPC quality [ms]	Control quality	OPC quality [ms]	Control quality	OPC quality [ms]
Synchronous cache read, synchronous write	5539,60	8,72	5526,60	3,68	5550,30	40,32
Asynchronous read, asynchronous write	5807,70	162,80	5777,80	161,75	5797,60	162,00
Synchronous cache read, asynchronous write wireless with traffic	5921,70	45,33	5992,30	47,46		
Synchronous cache read, asynchronous write wireless	5566,50	9,43	5714,20	21,51		
Asynchronous read, synchronous write	5781,10	162,39	5836,10	162,50	5795,30	162,30
Synchronous cache read, asynchronous write	5544,40	13,31	5490,30	4,17	5551,60	5,82
Synchronous device read, synchronous write	5674,60	6,99	5507,70	5,50	5486,70	5,45
Synchronous device read, asynchronous write	5489,80	6,39	5493,90	7,19	5484,50	3,88
Matlab PID - simulation	5491,80					
PLC PID - asynchronous read, synchronous write	5364,20	48,59	5362,00	47,32	5360,12	42,32
PLC PID – simulation	5440,80					

In the next part of the experiment different OPC item refresh time T_{ITEM}=500ms was used. OPC server refresh rate was set to default value T_{SERVER}=100ms. Different OPC configurations were simulated. For each T_{ITEM} new optimal PID settings have been calculated.



PID settings

GAIN = 1.401

Ti = 6.638 s

Td = 0 s

Control quality

Matlab PID	6408,5
Simulation	6432,8

Figure 6.23. Comparison of simulation results and control by Matlab PID, T_{ITEM}=500ms, synchronous device read, synchronous write.

Table 6.2. OPC item refresh rate T_{ITEM}=500ms, OPC server refresh T_{SERVER}=100ms.

Configuration	Test 1		Test 2		Test 3	
	Control quality	OPC quality [ms]	Control quality	OPC quality [ms]	Control quality	OPC quality [ms]
Asynchronous read, asynchronous write	6436,10	39,8	6237,90	62,6	6093,40	49,45
Asynchronous read, synchronous write	5933,00	61	6428,00	53	6174,30	35,05
Synchronous cache read, synchronous write	6618,51	5,2	6610,80	4,65	6624,60	5,35
Synchronous cache read, asynchronous write	6505,20	23	6777,20	4,55	7069,80	6,75
Synchronous device read, asynchronous write	6408,50	3,8	6415,40	11,65	6401,30	10,3
Synchronous device read, synchronous write	6434,80	7,1	6391,70	5,95	6389,80	10,5
PLC PID	5496,90	9	5503,00	7,95	5502,90	5,85
PLC PID – simulation	5625,10					
Matlab PID – simulation	6432,80					

Table 6.3. OPC item refresh rate T_{ITEM}=600ms.

Configuration	Server refresh rate T_{SERVER}=200ms		Server refresh rate T_{SERVER}=600ms	
	Control quality	OPC quality	Control quality	OPC quality
Asynchronous read, asynchronous write	6094,00	49,96	6116,20	139,38
Asynchronous read, synchronous write	6079,50	59,1	6082,70	182,34
Synchronous cache read, synchronous write	6176,50	4,48	6352,60	6,6
Synchronous cache read, asynchronous write	6657,40	1,62	6181,70	5,46
Synchronous device read, asynchronous write	6147,90	2,9	6145,70	7,5
Synchronous device read, synchronous write	6147,70	2,14	6143,80	6,42
Matlab PID – simulation	6129,70			

Table 6.4. OPC item refresh rate T_{ITEM}=200ms, Ethernet with a huge traffic.

Configuration	Server refresh rate T_{SERVER}=200ms	
	Control quality	OPC quality
Asynchronous read, asynchronous write	5952,70	322,4
Asynchronous read, synchronous write	6253,50	324,86
Synchronous cache read, synchronous write	6001,50	44,92
Synchronous cache read, asynchronous write	5853,70	46,92
Synchronous device read, asynchronous write	5931,70	63,24
Synchronous device read, synchronous write	6156,30	51,2
MATLAB PID – Simulation	5624,00	

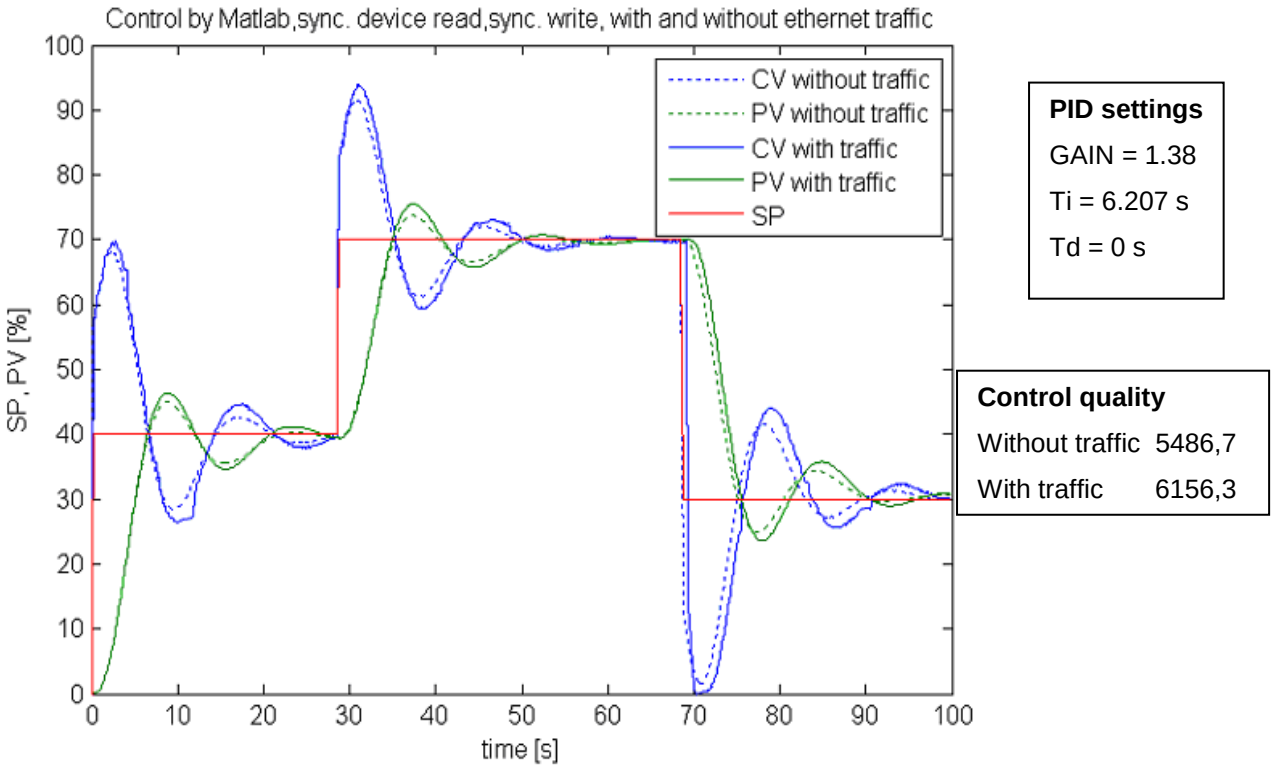


Figure 6.24. Control by Matlab PID via Ethernet with and without traffic, T_{ITEM}=100ms, synchronous device read, synchronous write.

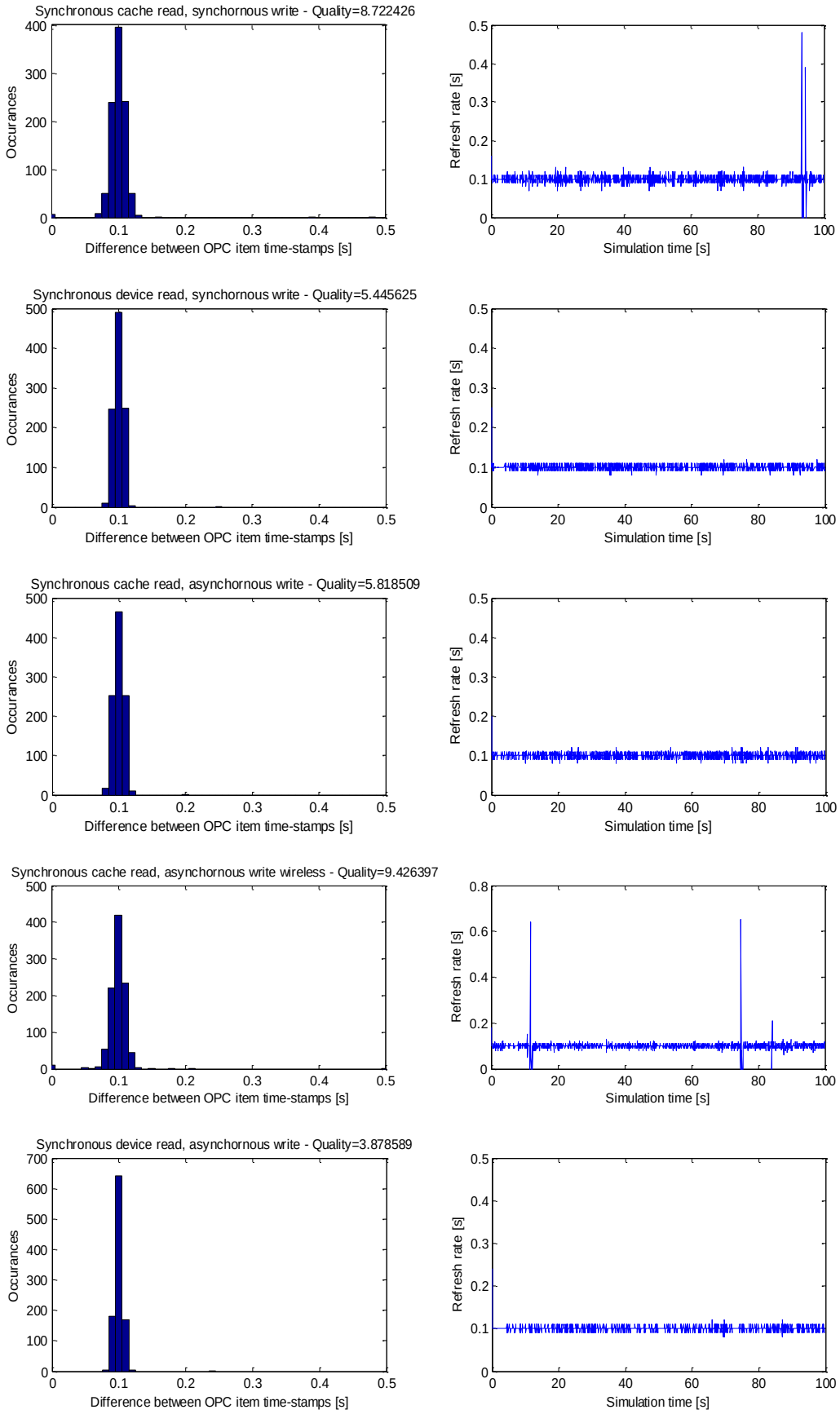


Figure 6.25. OPC refresh rate

6.4.4. Asynchronous reading from OPC server.

According to results of the experiments, asynchronous reading has a significantly worse OPC quality factor then synchronous one. As it can be seen in *Figure 6.26* very often the OPC client needs about 500 ms to asynchronously read data from the device. If one looks in details into time stamps vector, he will recognize that many items has the same time stamp. It can be easily explained. According to documentation of OPC toolbox, while asynchronous reading Matlab sends request to the OPC server for new data, simulation is being processed until event from the server arrives. Probably this even cannot be processed each simulation step because of CPU overload. Please remember, that while waiting for this event processing the simulation is running and obsolete data are used for simulation. For asynchronous reading the process control value has characteristic “saw” shape *[Figure 6.27]*. This effect is caused by fact that the control value (CV) refreshment frequency is slower then process value (PV) refreshment frequency.

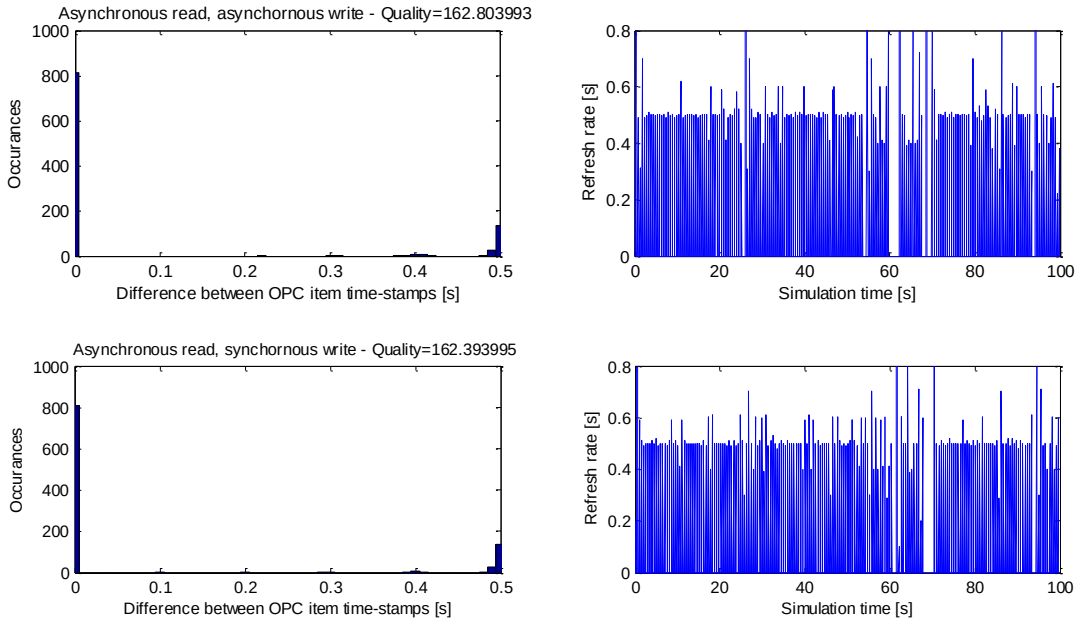


Figure 6.26. Asynchronous data read – Item refresh rate histogram.

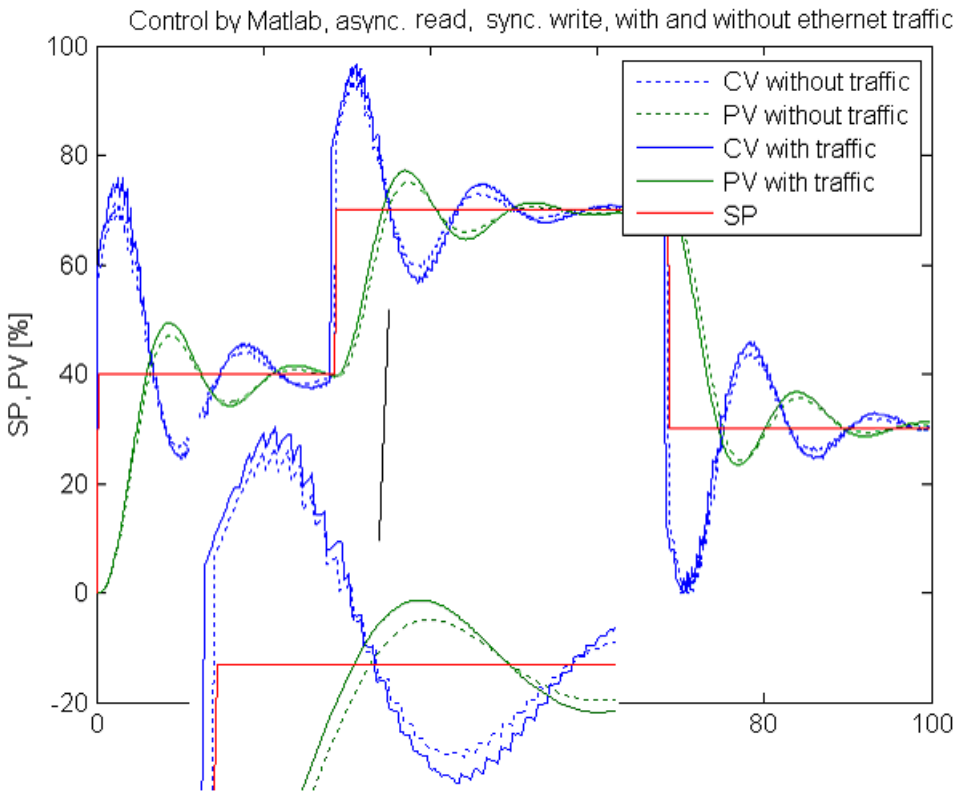


Figure 6.27. Control by Matlab PID. Asynchronous data read, synchronous data write.

6.5. Experiment 3

6.5.1. Using Matlab for unstable process control

In classical control theory stability means that for any bounded input, over any period of time, the output will also be bounded. In practice, for continuous, linear control systems it means that the system is stable if the real part of all the poles of its transfer function is less or equal to zero.

During this part of the experiments faster and unstable process was controlled. Its transfer function has 3 poles (-10, -5 and 0.5). One of them is placed in the right half of the complex plane, which means system is unstable.

$$G(s) = \frac{1}{(s-0.5)(0.1s+1)(0.2s+1)}, \quad (6.11)$$

Similarly to previous experiments optimal PID parameters have been obtained with Adjusted Model Technique. Different configurations have been tested. Results are presented in *Table 6.5*.

Table 6.5. Unstable process control. OPC item refresh rate T_{ITEM}=100ms.

Configuration	Test 1		Test 2		Test 3		Test 4	
	Control quality	OPC quality [ms]	Control quality	OPC quality [ms]	Control quality	OPC quality [ms]	Control quality	OPC quality [ms]
Synchronous cache read, synchronous write	4259.2	5.93	5166.60	6.95	4210.90	4.46	4525.4	41.91
Synchronous cache read, asynchronous write	4894.8	4.58	4217.40	4.71	4319.70	28.69	4600.9	5.36
Synchronous device read, asynchronous write	4175.0	6.20	4173.40	6.41	4187.30	6.53	4190.6	6.26
Synchronous device read, synchronous write	4196.6	5.43	4162.40	8.22	4192.60	9.54	4163.3	6.07
Synchronous device read, synchronous write with traffic	4461.8	38.42	4929.70	20.36	4935.40	43.69		
Synchronous device read, asynchronous write with traffic	5068.3	7.35	4700.80	14.51	5276.60	52.80		
PLC PID – simulation	4139.2							

The unstable control process has higher requirements in case of control quality. Using asynchronous reading effects system had huge overregulation. Effects described in *Chapter 6.4.4* occurred and control quality was so weak that it is unreasonable to put results into *Table 6.5.*

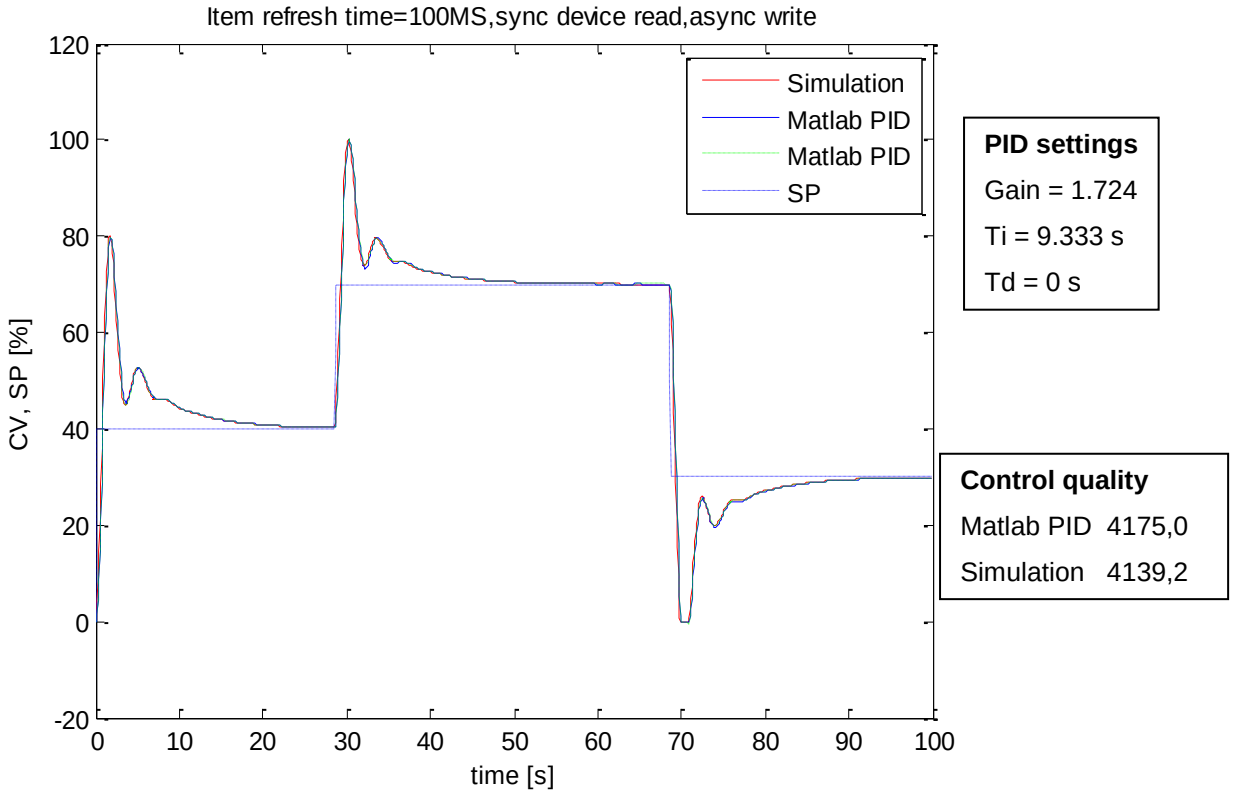


Figure 6.28. Control by Matlab PID. Asynchronous device read, synchronous write.

6.5.2. Influence of TCP traffic on control quality.

In this part impact of TCP traffic on control quality is measured. PLC and PC with MATLAB were connected with each other via hub *[Figure 6.29]* additionally three other computers were connected to the hub. Between PC1 and PC2 artificial TCP traffic was generated. The results have been presented in *Figure 6.30*. To generate small TCP traffic small files were transferred from PC1 to PC2. According to the snifter software installed on the third computer transfer was about 6.8 MB/s. For huge TCP traffic two 1 GB files were sent simultaneously between PC1 and PC2. Transfer rate can be estimated for about 80 Mbps. Traffic lower then 10% of bandwidth causes a very small impact on control quality, paradoxically in many cases control with additional TCP traffic has a little better quality factor then without traffic. For huge bus load (approx. 80Mbps) in first seconds of simulation process value rush into boundaries. It is not possible to control the process.

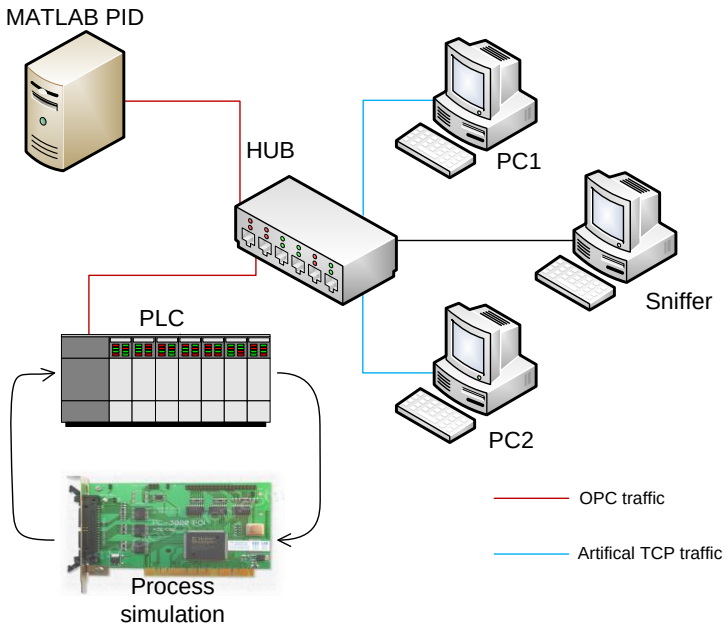
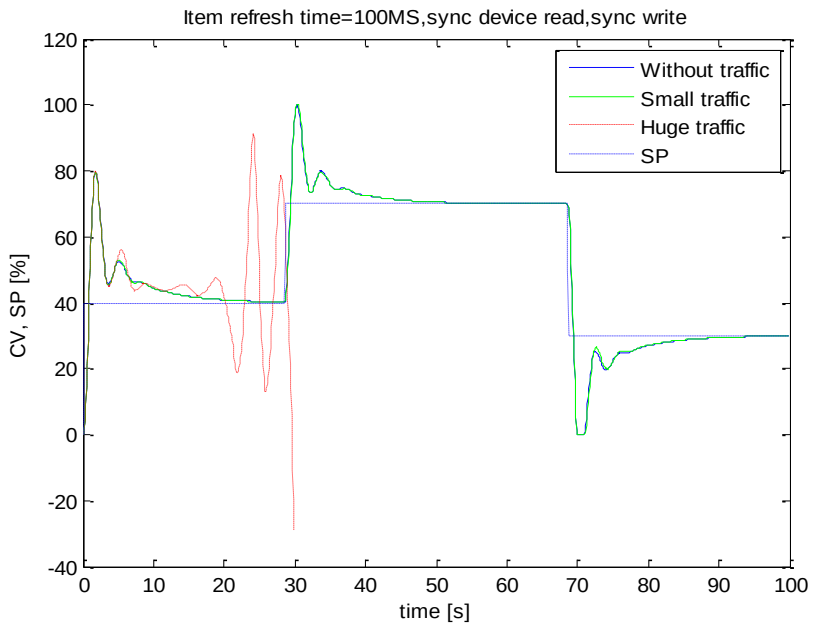


Figure 6.29. System configuration for artificial TCP traffic generation



Control quality without traffic - 4196.60

Control quality with small traffic - 4176.80

Figure 6.30. Influence of TCP traffic on control performance.

6.5.3. OPC traffic analysis.

To analyze OPC traffic on Ethernet cable similar configuration to that shown in *Figure 6.29* was used. In this case there was not need to generate additional traffic by PC1 and PC2. All data communication between PLC and OPC Server was captured by the Sniffer PC. *The Wireshark Network Analyzer* was used for Ethernet frames atomization.

The result of Ethernet traffic analyses has been described below. The OPC Server periodically exchanges data with the PLC *[Figure 6.31]*. This telegram is exchanged even when there is nothing to read or write. Frequency of refreshes depends on configured server's refresh rate. Default Siemens OPC server refresh rate T_{SERVER}=100ms have been used.

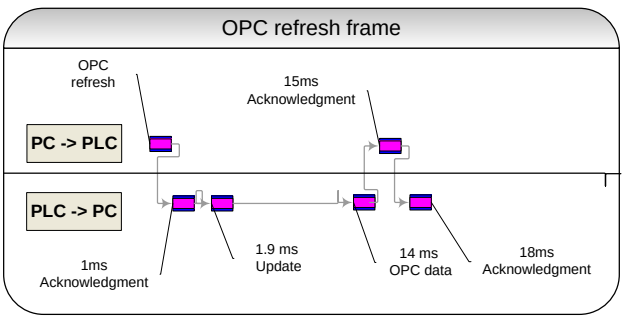


Figure 6.31. OPC refresh frame.

Apart from this communication depending on configuration additional frames are being sent during the control process. All possible configuration have been tested and results are presented in *Figure 6.32*.

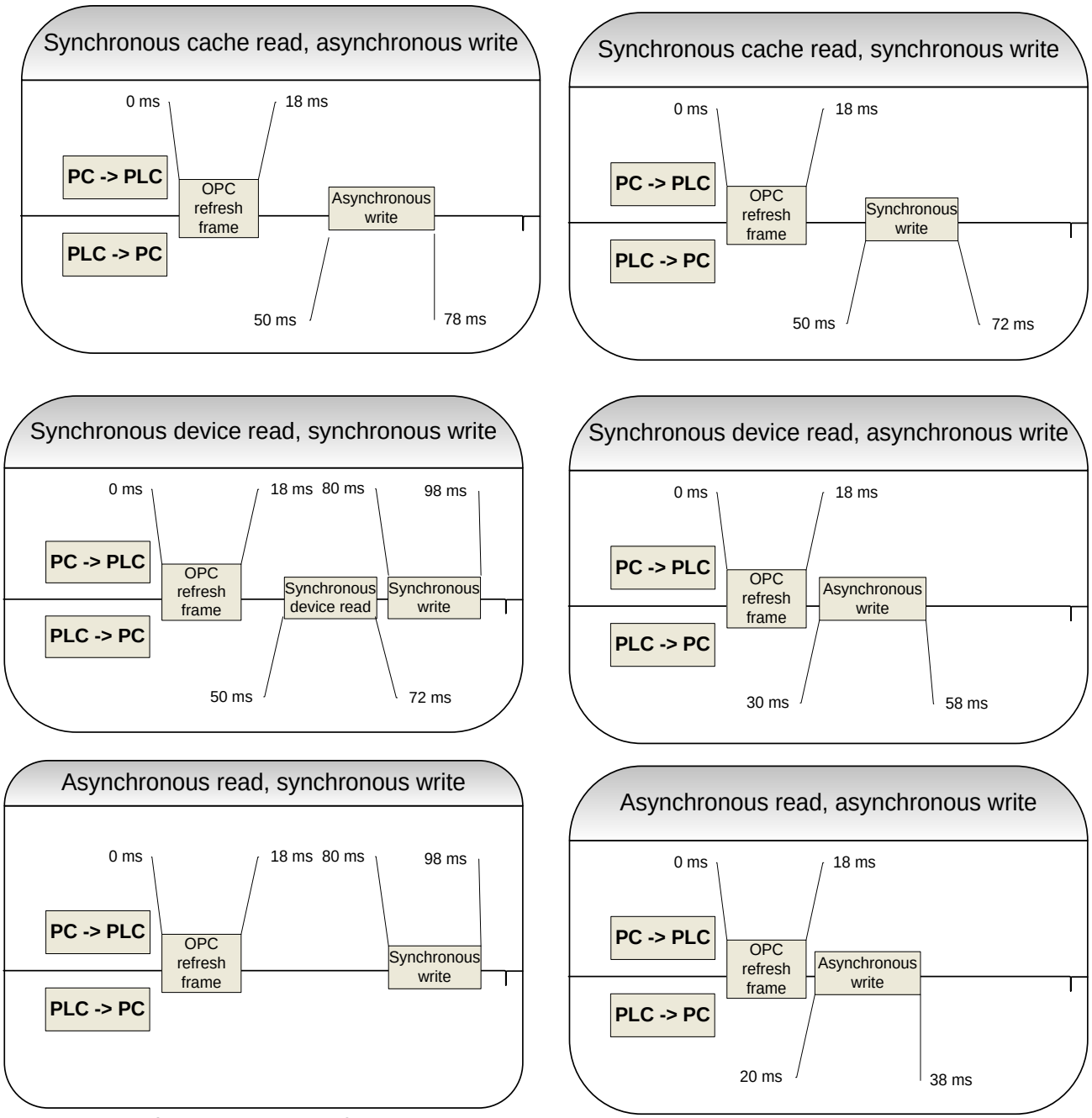


Figure 6.32. OPC frames.

6.6. Experiment 4 - Influence of TCP traffic on performance of writing OPC items to PLC.

In previous tests, OPC reading performance was measured with using OPC quality factor *[Equation 6.10]*. In this part similar test will be done to test various methods of OPC writing. Because there is no time stamp associated with value written to OPC server, a different approach have been used. Computer with MATLAB and OPC toolbox generated square wave and write it into PLC's output. DAPIO card was connected to the output of the PLC.

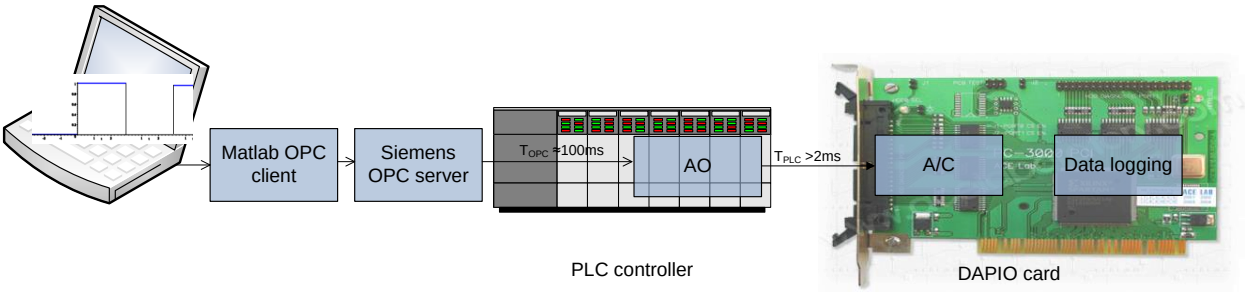


Figure 6.33. System configuration for OPC writing test.

The reference wave have been generated by Matlab and transferred via OPC to PLC analog output module. OPC Server refresh rate and OPC item refresh rate were set to T_{ITEM}= T_{SERVER}=100ms. Wave's period was set to 200ms (change of state each OPC refresh). The generated wave has been captured via analog input of DAPIO card. Signal's sampling frequency was set to 1 kHz.

For further processing DAPIO card signal's log has been transferred to Matlab's workspace. Four factors have been taken into account for writing performance tests:

- jitter of wave length
- jitter of the high and low state
- variation of wave length

Table 6.6. Reconstructed wave parameters depending on network load.

		Synchronous				Asynchronous			
		Mean jitter [ms]	Variance [ms]	Jitter Hi [ms]	Jitter Low [ms]	Mean jitter [ms]	Variance [ms]	Jitter Hi [ms]	Jitter Low [ms]
Experiment 1	Ethernet	0.4444	0.8498	0.3333	0.3889	0.6389	1.6833	0.7222	0.7778
	Wireless	0.7778	1.5275	0.9444	0.6111	0.8333	1.2019	0.6667	0.4444
	Wireless with traffic 10%	2.6111	5.3903	2.7778	2.6111	11.2143	56.4624	22.7143	27.5
	Wireless with traffic 50%	6.7813	112.1049	57.5625	57.4375	14.6786	72.2679	27.5	33.2143
Experiment 2	Ethernet	0.4231	0.9253	0.4323	0.493	0.6128	1.7256		
	Wireless	0.7222	3.266	0.8889	1.7778	1.8056	3.0092	1.6667	1.2778
	Wireless with traffic 10%	2.5	2.1213	1.8889	0.8333	3.9167	41.4387	23.3333	23.0556
	Wireless with traffic 50%	5.7105	45.0368	20.7895	24.4737	30	232.5182	397.5455	421.9091

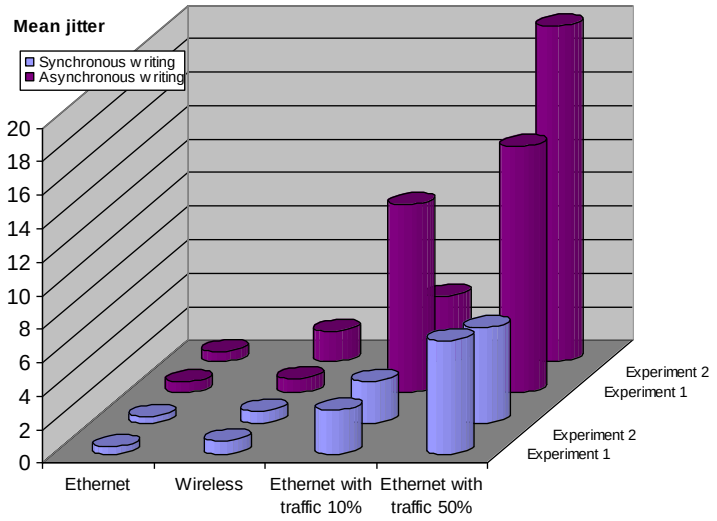


Figure 6.34. Mean jitter of measured wave.

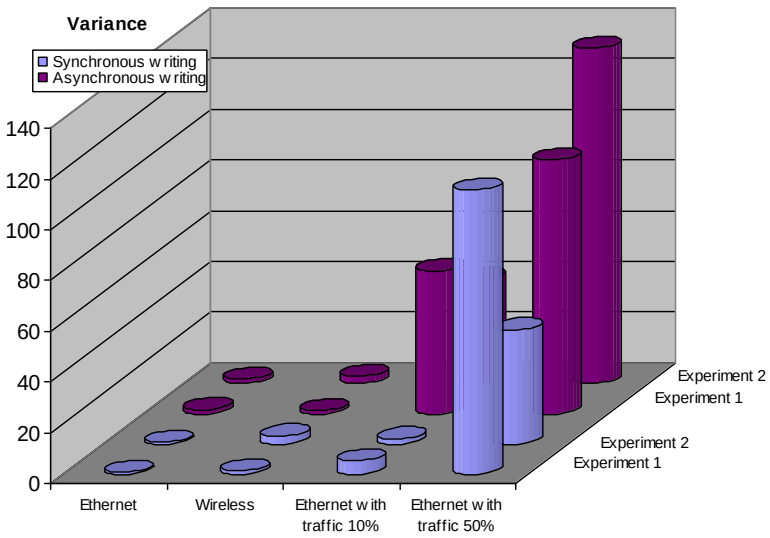


Figure 6.35. Variance of measured wave.

6.7. Experiment 5

6.7.1. Usage of a Smith predictor for control with OPC.

As it was mentioned before, OPC introduces two additional delays into control loop. The Smith Predictor is one of the most well-known methods for compensation constant delays in stable processes. It is known as easy and effective, but it requires a good model of the process under control.

Let $P(s) = G(s)e^{-sT}$ be a transfer function of the process with delay and $P_m(s) = G_m(s)e^{-sT_m}$ be a model of the process with delay. For $P(s) = P_m(s)$ transformation described in *Figure 6.36* is accurate. On this scheme, it was assumed that there is only one delay in control loop which is situated after the process (process value signal is delayed). In the case of control with OPC both control signal and process value are delayed. With the method presented on the *Figure 6.36* only the delay situated after controlled process can be moved outside the control loop. Practical implementation of that scheme is shown in *Figure 6.37*. For further consideration this algorithm will be called as a basic Smith predictor.

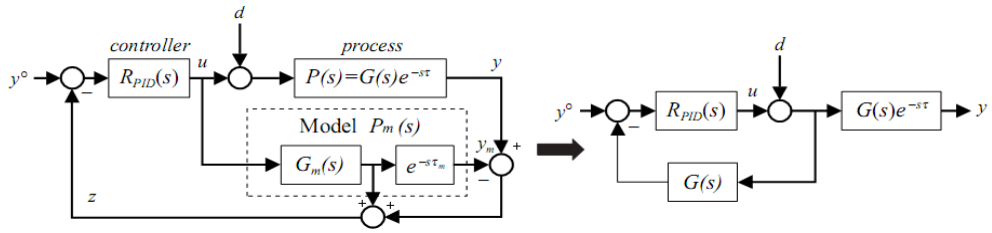


Figure 6.36. Idea with a basic Smith Predictor

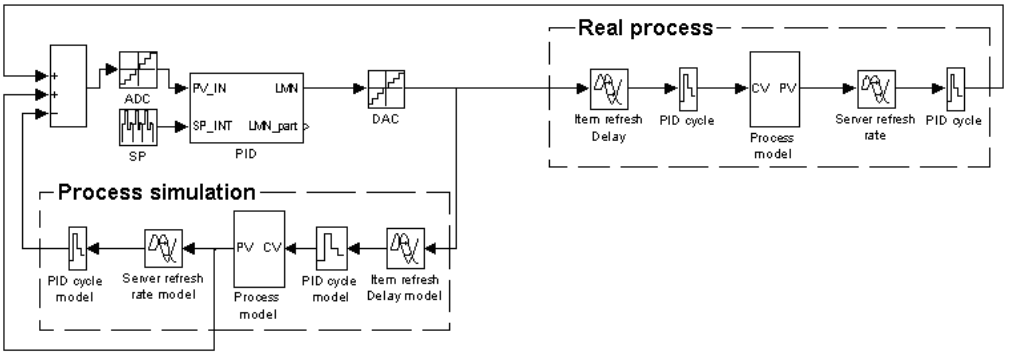


Figure 6.37. Simulation model of close loop system with a basic Smith predictor

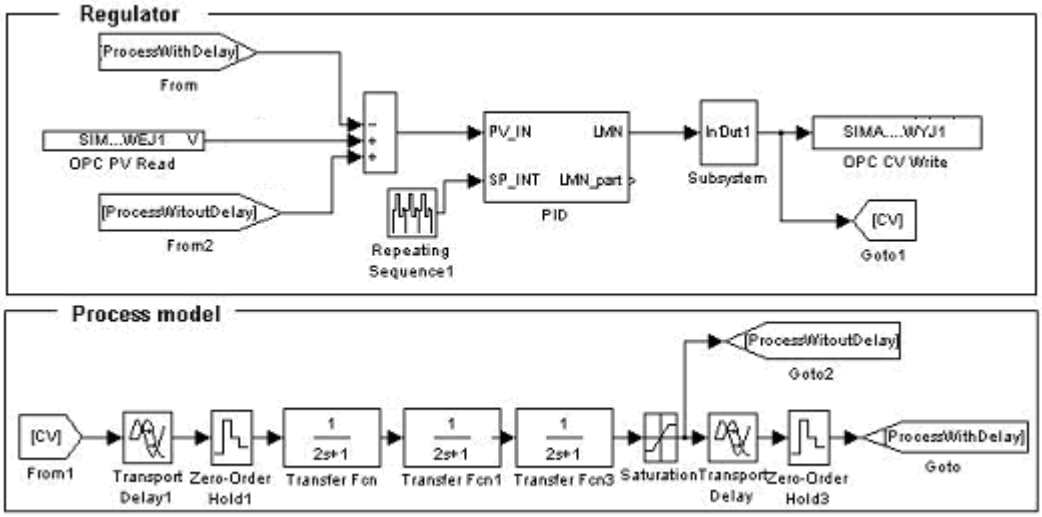


Figure 6.38. Control system controlled via OPC with a basic Smith's predictor applied.

As it can be see in *Figure 6.37* basic Smith predictor approach do not eliminate delays associated with control signal transmission. To solve the problem of delayed control signal, modified Smith Predictor approach was proposed *[6.16]*. This version of Smith algorithm was designed for eliminate network delays. The main idea is described in *Figure 6.39*. Further in this paper, presented approach to Smith's predictor will be called as a network Smith predictor.

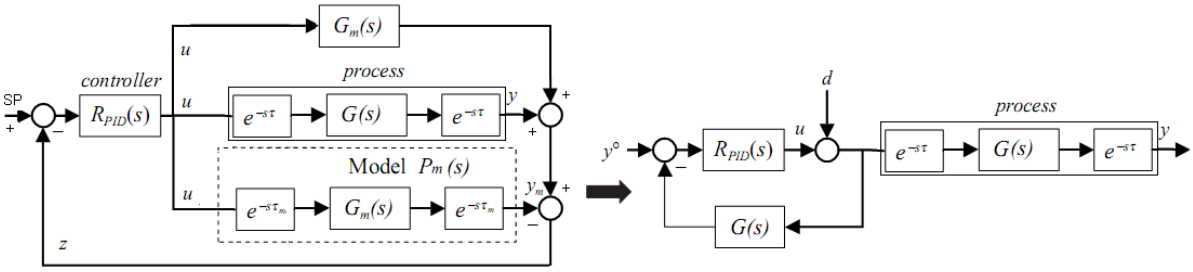


Figure 6.39. Smith Predictor algorithm for network delays.

6.7.2. Stable process control with a Smith predictor.

Both kinds of Smith's predictor have been tested in further part of this experiment. Similarly to experiment described in *Chapter 6.3* at first stage a stable process transfer functions has been chosen as a process equivalent. The controlled process in this part can be described as a 3rd order inertial object with time constants T_1,T_2,T_3 = 2s. First optimal PID settings have been calculated for system controlled by PLC *[Figure 6.19]*. The Adjusted Model Technique and a model described in *Figure 6.12* have been used for this purpose. In the next

step optimal PID parameters have been calculated for Matlab PID control system, Matlab PID with basic a Smith predictor and Matlab PID with a network Smith predictor. All tests have been performed. Their results are presented in *Table 6.7*. As it can be seen applying Smith's predictor into the control systems in most cases improved the control quality. Exception is the case in which the real system was controlled by the Matlab PID with a network Smith predictor. As it can be seen in *Figure 6.40,* there is a constant error in controlling by Matlab with network a Smith predictor. The Smith Predictor algorithm assumes that the process and its simulation are exactly the same. In this case real controlled process and its simulation are the same because they are both simulated – one of them with Matlab, second with dedicated application connected to DAPIO card. The PV value came from DAPIO card. Author predicts that this constant error between the process and its model is caused by small mistake in PLC's analog modules or DAPIO card scaling.

	Control by PLC	Control by Matlab without Smith predictor	Control by Matlab with basic Smith predictor	Control by Matlab with network Smith predictor
Simulation	5302	5560	5409	5300
Real control system	5326	5486	5422	5588

Table 6.7. Control quality for the stable process.

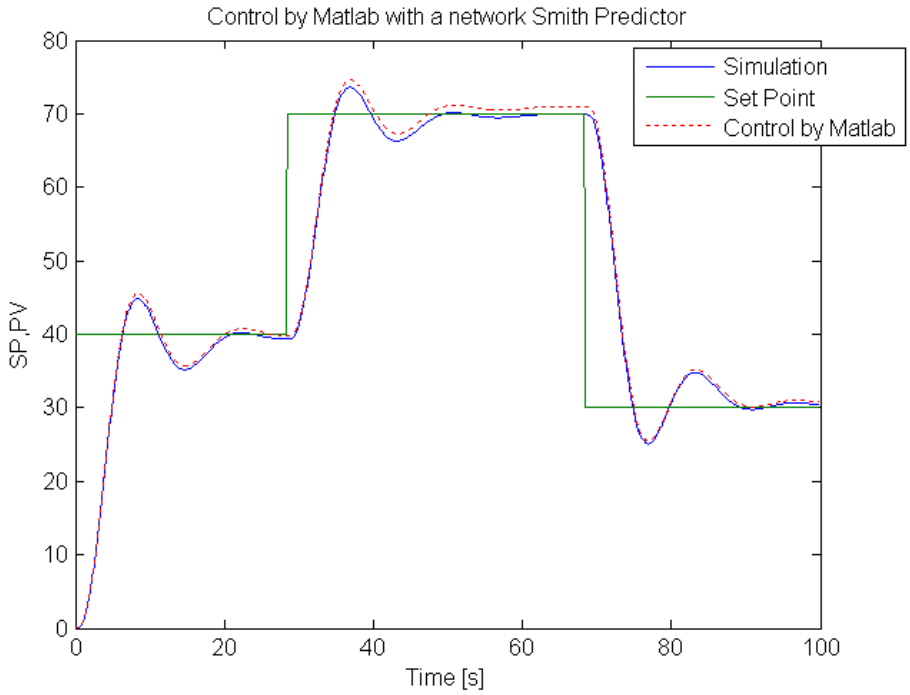


Figure 6.40. Control system with a network Smith predictor.

To prove that the Smith's predictor is working the plot with controlled process's reactions on set point value have been presented in *Figure 6.41*. As expected, the best control has been obtained with PLC base control system. Control system with the network Smith predictor had similar reaction time, but due to the constant error between set point and process value, control quality was poor. It can be observed that a basic Smith's predictor effect with better control quality than a system controlled by Matlab without Smith predictor.

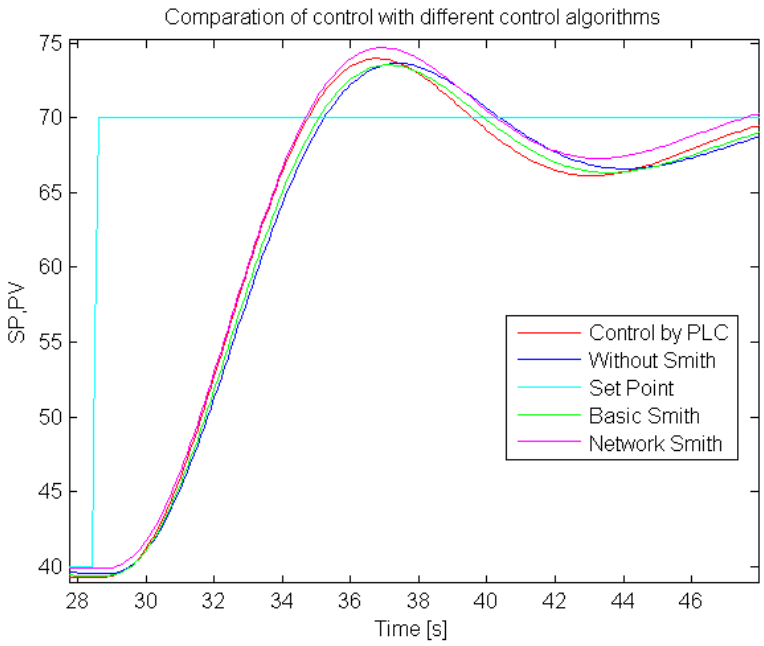


Figure 6.41. Control systems reaction for set point change.

6.7.3. Unstable process control with using a Smith predictor.

In this part numerous experiments have been conducted for controlling unstable process with a basic and network version of Smith's algorithm. As it was done in previous experiments 3rd order unstable object with one of poles in the right half of the complex plane have been chosen (-10, -5 and 0.5). Four experiments have been performed their results have been presented in *0*.

Table 6.8. Control quality for the unstable process.

	Control by PLC	Control by Matlab without Smith predictor	Control by Matlab with basic Smith predictor	Control by Matlab with network Smith predictor
Simulation	943	2440	1362	919
Real control system	936	2414	1720	-

In contrast to a stable process in this case, the difference in control quality between simulation and real control occurred also for the basic Smith algorithm. There was no constant error of regulation but small the differences between the controlled process and its simulation in Smith's algorithm have been observed. Application of the network Smith algorithm for unstable object control caused the real close loop control system lost its stability where as the simulation of this system was stabile.

To avoid the differences between the model and the real system modification of a Smith algorithm was proposed. If one compare *Figure 6.42* and *Figure 6.37* he will see a kind of a second feedback loop has been introduced into a system. The difference between delayed PV of real process and its simulation is connected to input of the process simulation. Highlighted modification will resist against occurrence of constant deviations from the process's set point value. Theoretically, both real PV and its simulation should be equal, but in case of small differences, the suggested modification will help to keep both values equal. It must be remembered that this compensation is done with a delay of 2 network delays, because both simulation and real process value are delayed.

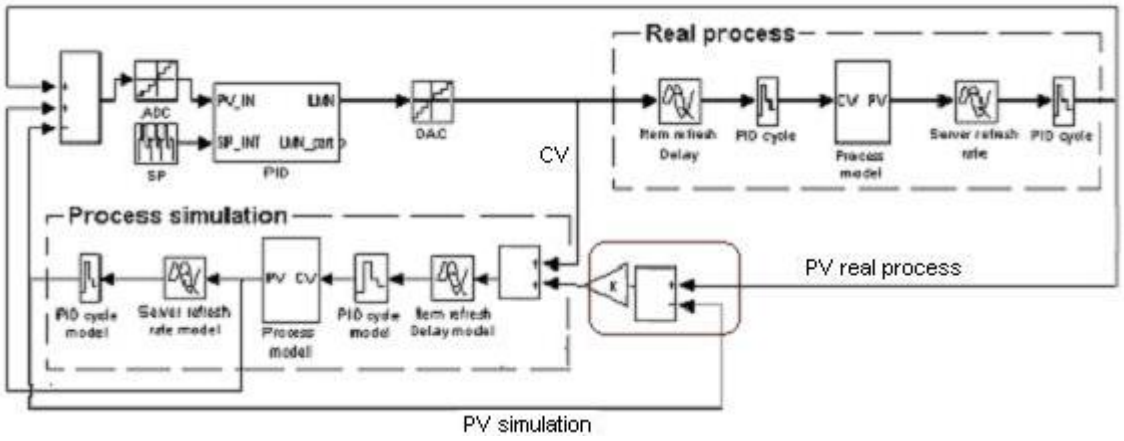


Figure 6.42. Modification of basic Smith algorithm to reduce constant error between simulation and real process – MATLAB's simulation model

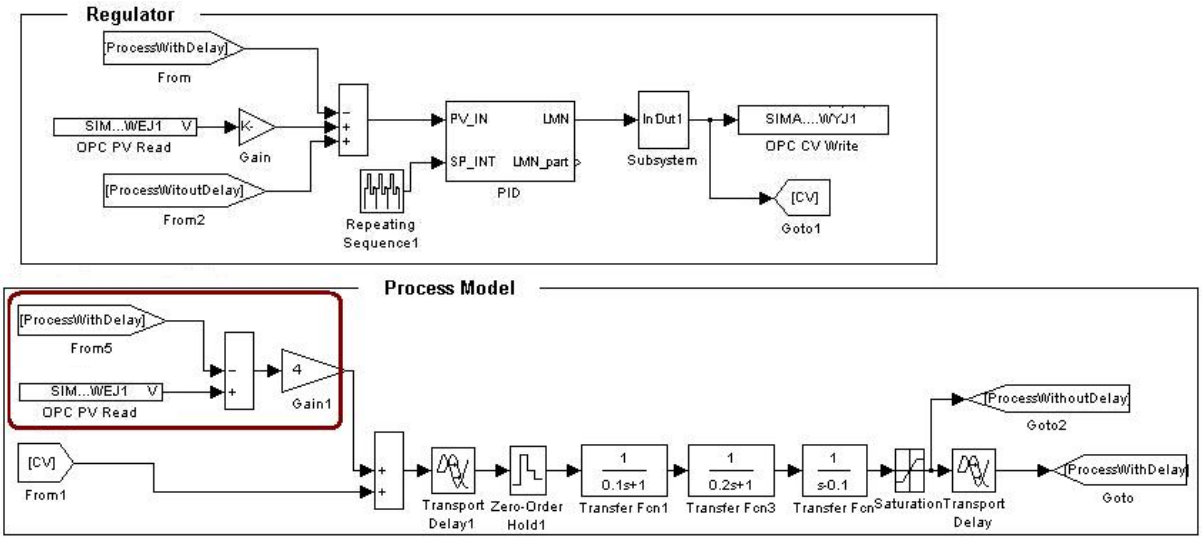
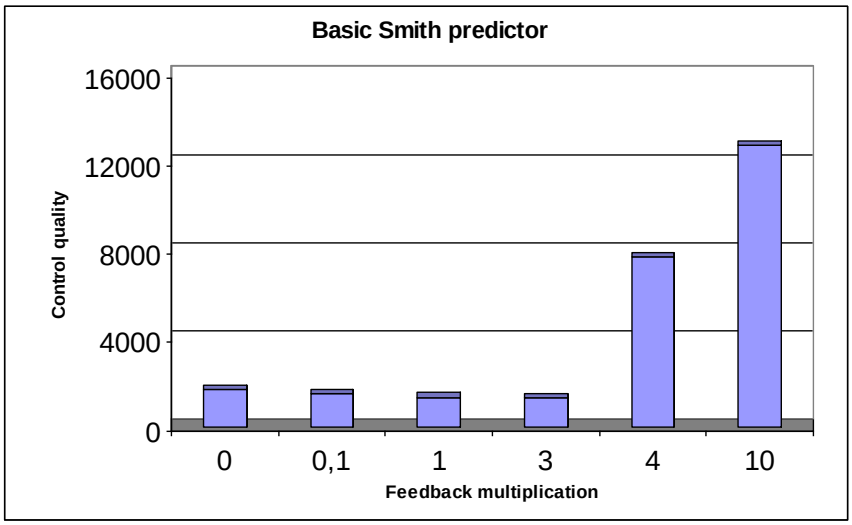


Figure 6.43. Modification of network Smith algorithm to reduce constant error of process simulation – real object controlled with OPC

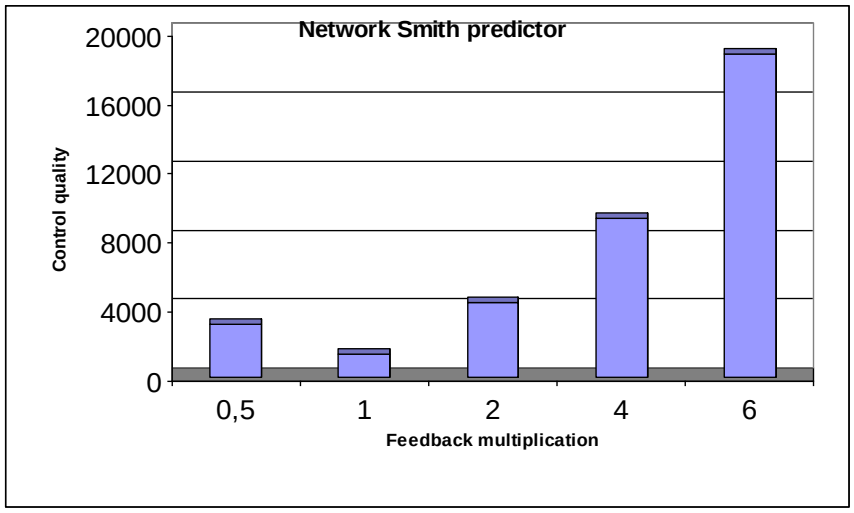


Feedback multiplication	Control quality
0	1720
0,1	1528
1	1354
3	1348
4	7726
10	12785

Simulation	1362

Figure 6.44. Influence of feedback on control quality in a basic Smith predictor.

Afterwards this modification was applied also for the network Smith algorithm. The impact of this on control quality can be observed in *Figure 6.44* and *Figure 6.45*. As it can be seen the control quality is varying with changing the influence of this feedback on the process's simulation input. Optimal value of this multiplication must be found because when it is too high system is loosing stability.



Feedback multiplication	Control quality
0,5	3080
1	1285
2	4306
4	9214
6	18785

Simulation	919

Figure 6.45. Influence of feedback on control quality in a network Smith predictor.

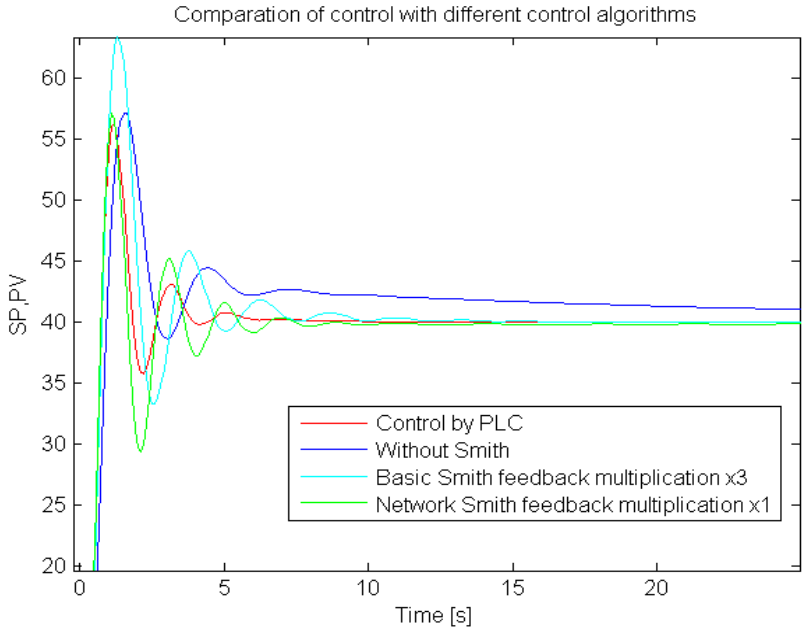


Figure 6.46. Unstable process control - reaction for set point change.

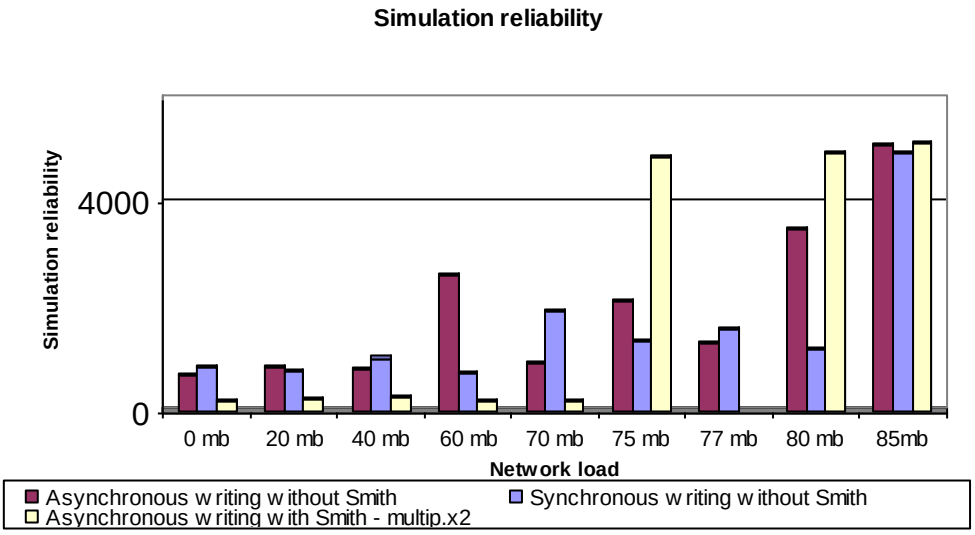
In the next step simulation's reliability have been tested. Additionally the impact of network load was measured. Configuration described in *Figure 6.29* was used for traffic generation. *PC1* and *PC2* were sending *ping* telegrams to each other. For each system's configuration the difference between simulation and real control system results have been calculated with following formula:

$$Symulation\_Reliabili\,ty = \sum_{i} \left| PV_{SIMULATION} - PV_{REAL} \right| , \quad (6.12)$$

where:

PV_{REAL} – Real process value

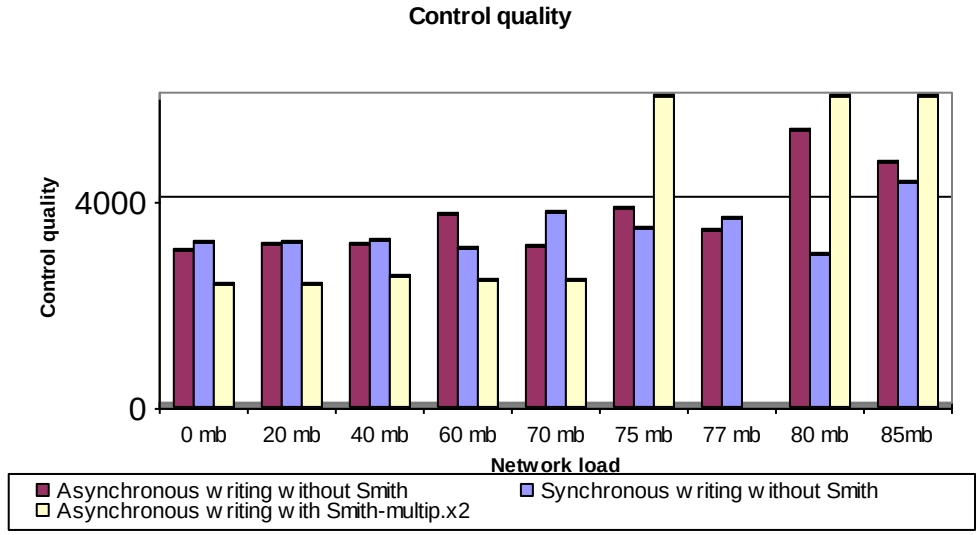
$PV_{SIMULATION}$ – Process simulation value



	0 Mb	20 Mb	40 Mb	60 Mb	70 Mb	75 Mb	77 Mb	80 Mb	85 Mb
Synchronous writing without Smith predictor	817	766	1006	700	1893	1314	1553	1186	4943
Asynchronous writing without Smith predictor	681	820	788	2611	917	2109	1281	3480	5104
Asynchronous writing with Smith predictor feedback multiplication x2	165	206	261	198	174	4873		4950	5139

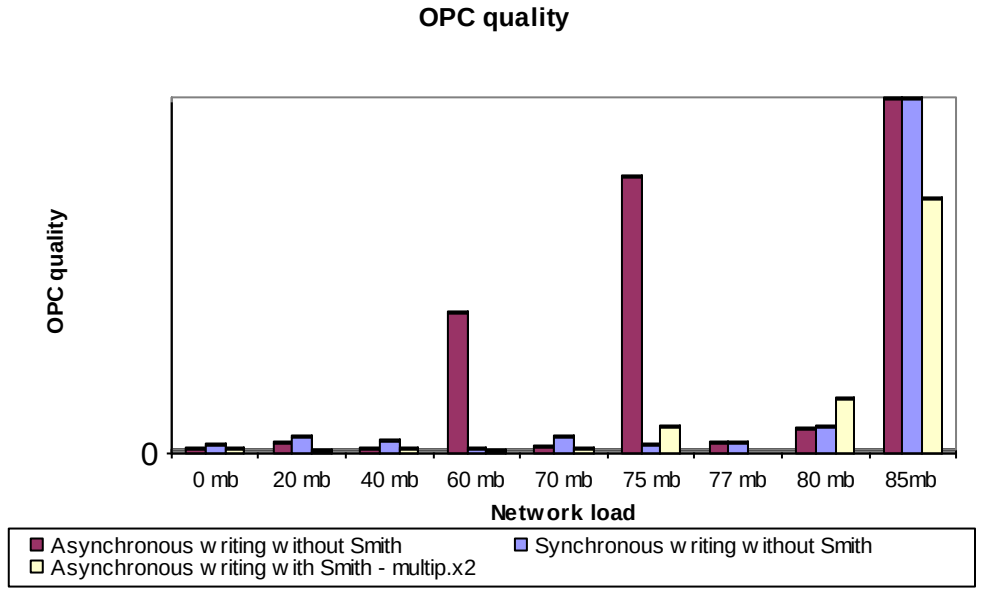
Figure 6.47. Simulation reliability.

In the next stage control quality has been calculated using sum of error's absolute values *[Equation 6.9]*.



	0 Mb	20 Mb	40 Mb	60 Mb	70 Mb	75 Mb	77 Mb	80 Mb	85 Mb
Synchronous writing	3171	3157	3208	3073	3774	3453	3659	2951	4329
Asynchornous writing	3037	3153	3118	3737	3115	3836	3407	5355	4730
Asynchornous writing with Smith predictor feedback multiplication x2	2374	2374	2500	2442	2443	6291		6503	6746

Figure 6.48. Control quality.



	0 Mb	20 Mb	40 Mb	60 Mb	70 Mb	75 Mb	77 Mb	80 Mb	85 Mb
Synchronous writing without Smith predictor	42	82	69	22	92	44	54	140	2016
Asynchronous writing without Smith predictor	15	56	19	789	32	1566	52	133	2456
Asynchronous writing with Smith predictor feedback multiplication x2	23	12	20	13	16	146		304	1435

Figure 6.49. OPC quality.

As it was shown in *Figure 6.47* - *Figure 6.49* the network load below 40% of the bandwidth do not results in noticeable difference in the quality control. Adding a Smith predictor with feedback multiplication made a system more stabile and network load below <70% do not have an impact on control quality. After crossing that border control quality is rapidly decreasing.

6.8. Experiment 6 - Implementing OPC in Excel

In many cases system developers need to prepare a simple solution for data logging and visualization. Of course, it is not always worth to buy SCADA software for this purpose. In those cases it is possible to use regular off-the-shelf office software like Microsoft Office. This chapter describes a simple solution to implement OPC data reading into Excel. At the end, discussion about the performance of this solution has been carried out.

To read data from the OPC server to Excel sheet an ActiveX component form Softing has been used. This component can be easy integrated into any ActiveX supporting application without a large programming effort (or event no programming effort). Its entire configuration is being done via input dialogs. The

logic of the application can be implemented in Visual Basic for Application (VBA) which is embedded into MS Office.

In this experiment Excel was used to visualize the control process from the previous experiments. Similarly Matlab PID was used as a controller and process was simulated with a DAPIO card. VBA macro was prepared to capture the data from the OPC server and save in into Excel's spreadsheet. Unfortunately Excel is not suitable tool to plot a real-time data. It was necessary to move drawing of the plots after the end of the simulation. Of course it is possible to prepare in Excel a real-time visualization just by showing actual value of the process.

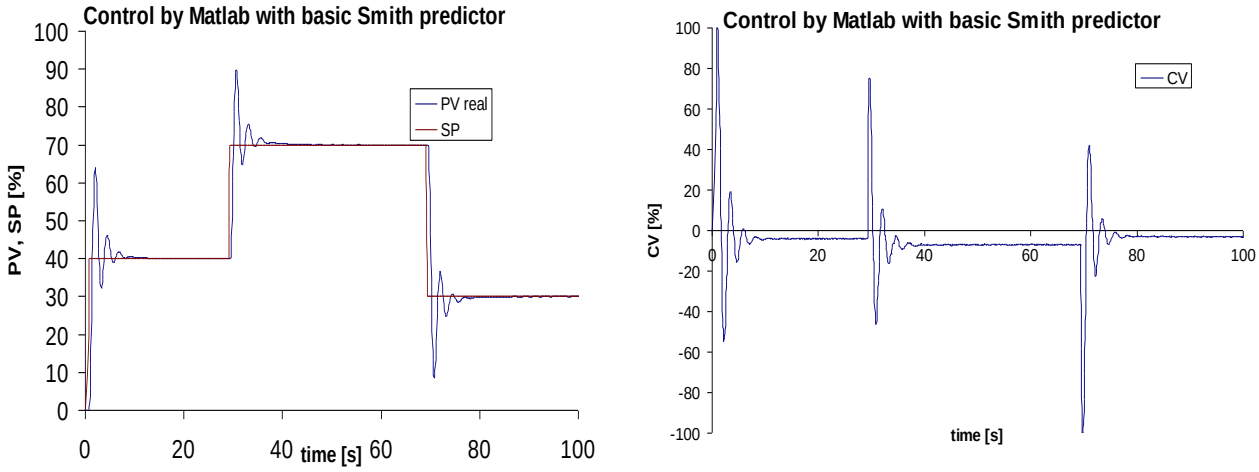


Figure 6.50. Control visualization in Excel

In the next step OPC reading/writing performance has been checked. It is probable that ActiveX component is not as much efficient as regular OPC Automation/Custom interface. To test it an empty array of 1000 integers has been created in PLC. Similarly an array variable was created in VBA. Matlab was no longer used. Before the experiment the cells from 1..1000 were filled with appropriate numbers form 1 to 1000. Excel was reading one by one value from those Excel's cells and wrote it into the variable in VBA memory. Next, those numbers were written to the OPC server using synchronous writing. After successful writing entire array those numbers were read again from PLC to variable in Excel's memory, each of them was incremented by VBA application to make sure all variables has been successfully read and written. In the last step numbers from the memory were again written to Excel's spreadsheet.

Time was measured in each operation. The results are presented in *Table 6.9.* As it can be observed the most time consuming operation is writing to Excel's spreadsheet. This should be remember while using OPC in Excel.

Table 6.9. Performance test of OPC ActiveX client in Excel.

	Experiment 1 [ms]	Experiment 2 [ms]	Experiment 3 [ms]	Experiment 4 [ms]	Experiment 5 [ms]
Reading 1000 values from Excel	15,625	15,625	15,625	19,53125	15,625
Writing 1000 values to Excel	31,25	31,25	46,875	46,875	31,25
Both RD&WR 1000 values to Excel	46,875	46,875	62,5	66,40625	46,875
Writing 1000 values to OPC	15,625	15,625	15,625	11,71875	15,625
Reading 1000 values from OPC	31,25	15,625	35,15625	35,15625	31,25
Everything	109,375	93,75	113,2813	113,2813	109,375

6.9. Experiment 7 - OPC on Windows QNX platform

Unfortunately, Windows is not a real-time operating system and it does not offer all features crucial to create real-time control application. For that reason, real-time operating systems (RTOS) have been developed. In *Chapter 4.3* Windows CE has been described. Windows CE is a sample of system which was developed for platforms with limited resources. In this part a real-time operating system created to perform mission-critical applications on more sophisticated platforms then in case of Windows CE has been described. QNX operating system was created to use in all situations where running 24h a day, 365 a year, nonstop is needed. In its normal form it offers similar capabilities to Windows XP and Linux (GUI, file system, networking.. etc). It can be installed on regular PC computer but also on various other platforms (ARM, PowerPC, XScale, MIPS).

QNX has much in common with Linux/Unix. Unfortunately, like in case of Linux there is no COM/DCOM built-in into the system. As it was described in *Chapter 4.3* a lack of COM technology support is a serious obstacle for realization of OPC specification. The next chapter can be use also as a description how to use OPC in Linux.

6.9.1. Connection between Windows and QNX/Linux

There are a few implementations of DCOM in Linux, but in fact they are not very popular for creating OPC in Linux or QNX. The tunneling approach has become more popular. It was described in *Figure 6.51*. This solution is also often used for even for communication between Windows systems. System's integrators often choose it because it eliminates the problems associated with the configuration of DCOM security settings. Tunnel removes the burden of figuring out how to manage OPC working on different platforms, in different domains and networks. It does not require advance knowledge about Windows configuration. User just needs to install the gateway on both OPC server and OPC client computers. Communication between the client-side and server-side OPC tunnel is done via a TCP/IP connection. The client can browse and connect to the servers as if they were running on the same machine, while the gateway exchanges data across a network without DCOM. To find out more about off-the-shelf software for OPC tunneling refer to *[4.6]* and *[4.7]*

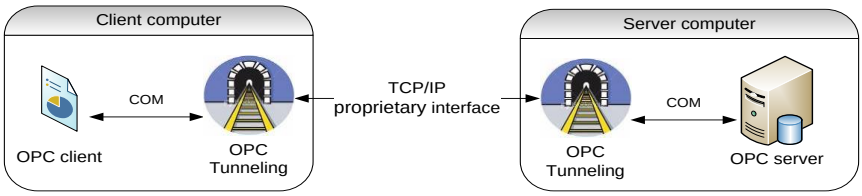


Figure 6.51. OPC tunneling solution.

Sometimes, control system developers choose to write their own tunneling solution. In further part of this paper a description of a dedicated solution for data communication between QNX and Windows has been described.

The idea with it was presented in *Figure 6.52*. A process simulation has been prepared in QNX. A PLC device was used as a process controller. The process control value is obtained from PID algorithm which was running on PLC. Process output was transferred to PLC as an input for PID block. The OPC protocol was used to communicate between Windows and PLC. Two gateways were created on the edge between Windows and QNX. At the QNX side the gateway was integrated into the process's simulation application. At the Windows side separate application in Java was prepared. To communicate with OPC server JEasyOpc open source library was used. The prepared application

was both an OPC client and a gateway to communicate with QNX. Socket connection has been opened to communicate between Windows and QNX. A proprietary XML messages with information about current process value and control value have been exchanged between gateways.

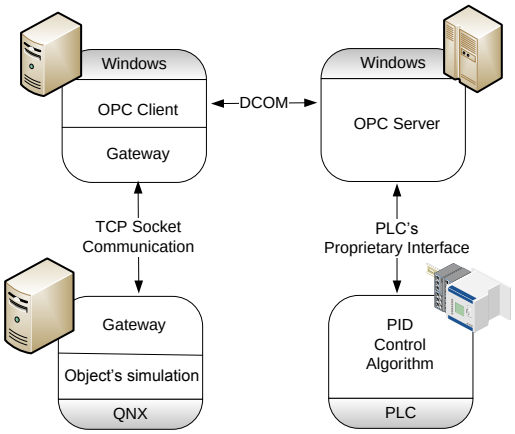


Figure 6.52. QNX and Windows communication.

6.9.2. Process simulation in QNX system.

A tank with a constant liquid temperature and inflow has been chosen as a typical example of industrial process. The control task is to keep the designed temperature and level in the tank. The liquid outflow and a power of the heater can be regulated. The main idea with this part has been to check if it is possible to simulate process in real time with QNX and control it with real PLC.

The following system of equations represents the dynamics of this test process.

$$\frac{dV}{dt} = \frac{1}{\rho}(w_i - w)$$
$$\frac{dT}{dt} = \frac{1}{V\rho}(w_i(T_i - T) + \frac{Q}{C}) \qquad (6.13),$$

where w_i = 0.4 − inflow stream$[\frac{kg}{s}]$ w − outflow stream$[\frac{kg}{s}]$

ρ = 1000 − liquid massdensity$[\frac{kg}{m^3}]$ V − liquid volume$[m^3]$

T_i = 100 − inflow liquid temperature[°C] T- outflow liquid temperature[°C]

Q - heaterpower[W] C = 1820 - liquid specyficheat$[\frac{J}{kg * K}]$

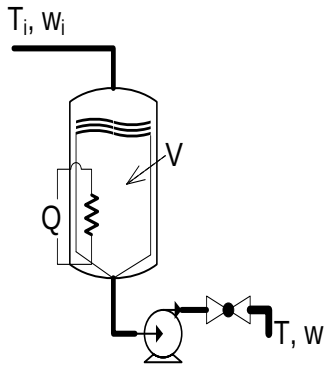


Figure 6.53. Model of the controlled tank.

The model of the process has been developed in Simulink. To meet the requirements of the process a dedicated PID controller has been prepared. The internal construction of this module has been shown in *Figure 6.54*.

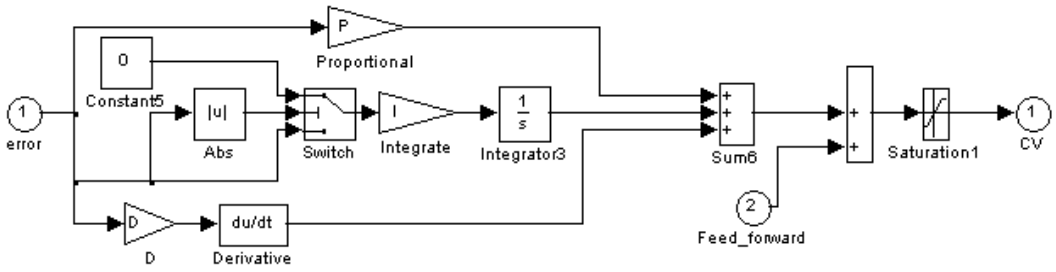


Figure 6.54. Model of PID controller for QNX simulation model control.

The main difference comparing to the standard PID controller is the fact that the error's signal is included into integration action only when the absolute value of error is lower than the set threshold. Additionally for each PID controller *feed forward* input is included into output signal. Based on the model of the process the appropriate feed forward values were calculated. Any error in these calculations will be offset by regulator's integrating action.

As it was done for previous experiments optimal PID settings have been evaluated with adjusted model technique. The quality of control was calculated as a sum of difference between process value and set value. The temperature does not affect the control of liquid level. First PID settings have been calculated for optimal flow control. At the second stage optimal temperature

control was settings were evaluated. Additionally, this algorithm has been modified to force at least minimal integration impact on control value.

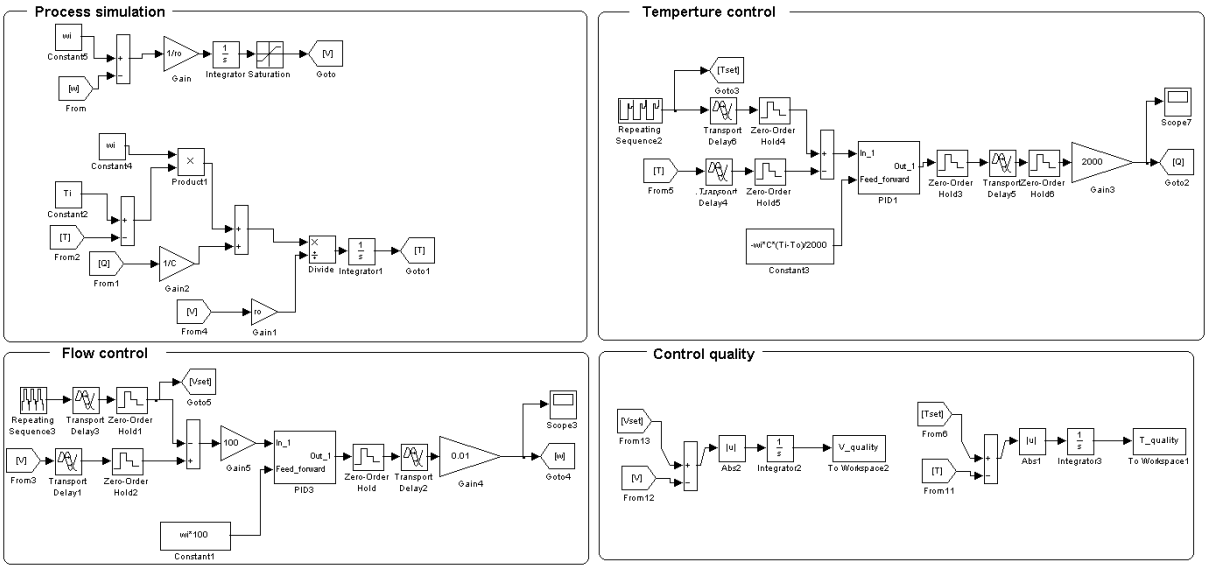


Figure 6.55. Model of the process in Simulink.

On QNX platform a model of the process has been also implemented. A separate thread was solving the deferential equations with Runge-Kutta method. Other thread was responsible for communication with Windows and writing to the database. Additionally GUI refresh was also a separate thread.

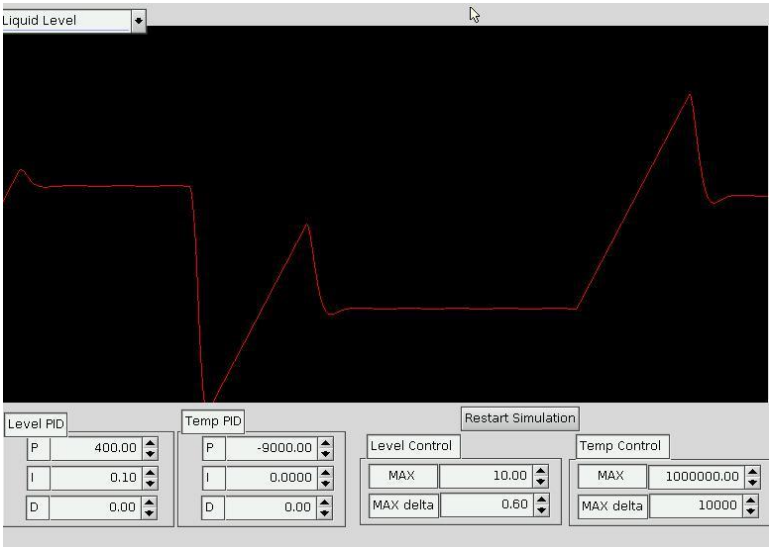


Figure 6.56. GUI of prepared QNX process simulation software.

This part of experiments was carried out by using the ABB AC800M PLC controller. This controller is much more powerful then previously used Siemens S7-300. The programming idea is a bit different than in case of regular PLC.

Programming is more object-oriented which makes it more similar to PC programming. The Ladder diagram programming has been replaced by a higher level programming language called Structured Text, which is very similar to Pascal. In Structured Text it is possible to easy write advanced and compact code in a logical and structured way.

The PID controller described in *Figure 6.54* has been implemented into PLC program. Each 250 ms the PID control task is performed on PLC. Additionally, the process state is read by MATLAB OPC client to capture the process data for further visualization.

Three experiments have been performed. They results have been described in *0*. Similarly to the previous experiments it can be observed a high compatibility between the simulation and the real controlling device.

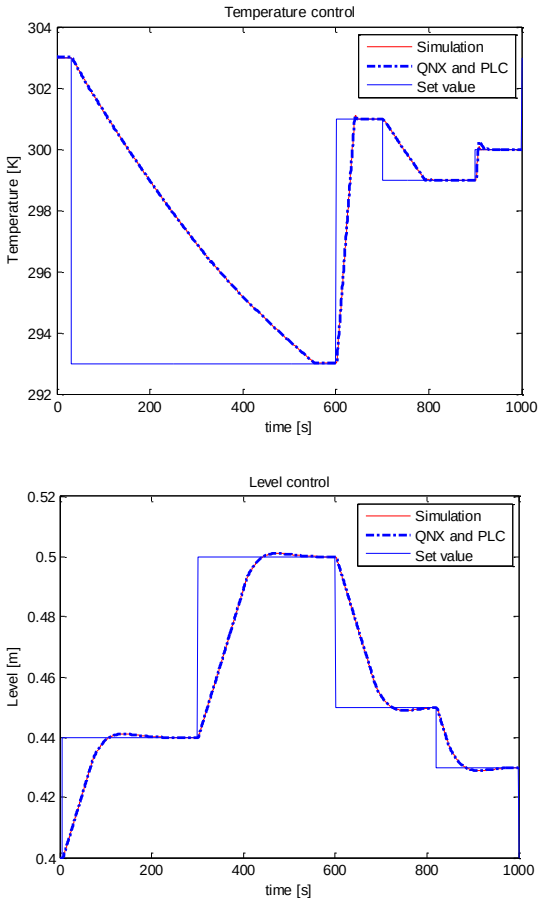


Figure 6.57. Level and temperature control by PLC.

Table 6.10. Control of QNX process simulation.

	Control quality			
	Simulation in Matlab		Simulation in QNX with PLC	
	Level	Temperature	Level	Temperature
Experiment 1	17.7875	5148.2	17.7125	5153.6
Experiment 2			17.6592	5144.4
Experiment 3			17.7127	5153.6

References:

6.10. www.anerma.be, Digital Process Simulator documentation

6.11. Siemens, SIMATIC Standard Software for S7-300 and S7-400 PID Control

6.12. Michał Brewiński (2008), Industrial Ethernet: Applications for Process Control

6.13. http://en.wikipedia.org/wiki/Hardware-in-the-loop (2008.11.10), Hardware-in-the-loop simulation - Wikipedia Free Encyclopedia

6.14. Mathworks, Matlab/Simulink OPC Toolbox™ 2 User's Guide

6.15. http://sine.ni.com/nips/cds/view/p/lang/en/nid/203826 (2008.11.10), NI LabVIEW Control Design and Simulation Module

6.16. Vatanski, Georges, Aubrun, Rondeaua and Jämsä-Jounelab, Control compensation based on upper bound delay in networked control systems

7. Summary

In this chapter summary of all previous chapters is made. Additionally, an attempt was made to predict a future of OPC and objectively judge which OPC specification have become a real industrial standards.

In this Thesis a different possibilities of implementing OPC technology in modern manufacturing process has been described. Without any doubts, nowadays OPC Data Access and OPC Alarm & Events have become a true industrial standard. They are especially popular for interconnection between the factory floor devices and SCADA, almost all PLCs support it. The standardization in terms of communication between industrial devices surely reduced the integration cost. Unfortunately, because of using Microsoft COM technology, OPC DA forces the operative system technology selection for developing client applications. Since DCOM is very limited in non-Windows systems probably in the future OPC DA will be gradually replaced with OPC UA or OPC XML. For now, those protocols are still not very popular, but it is sure that in the future it will change. The question is if vendors will now switch to OPC XML or they will wait until OPC UA will be fully specified. The huge advantage of both OPC XML and UA is possibility to transfer the data via Internet, which sometimes is needed to interconnect with ERP applications.

The main goal of this thesis was to investigate a possibility to use OPC standard for dynamic process control purpose, but additionally a brief description of al the OPC specifications and its practical use to integrate control systems have been made.

A few experiments have been performed. First Digital Process simulator was used a process. Unfortunately, there was not enough information about this device to prepare a reliable model of it in Matlab/Simulink environment. Even with this not very accurate model it was possible to calculate PID settings for the process controller and effectively control the process via OPC.

In the next step a special card connected to the PC with analog inputs and outputs was used and dedicated software has been prepared to simulate the process. With the model of the close loop system, PID settings were calculated. Additionally, the influence of network traffic for control quality was tested. Those

tests shown that in general small network traffic do not influence on control quality.

In some configurations, regular wireless (802.11g) has been used. According to results of the simulations, for normal conditions wireless control produces similar results as a regular Ethernet connection. As long as PC with the Matlab PID is in range is within range of the network there is hardly any difference between them.

One of the tasks of this thesis was to specify the best settings of OPC for the dynamic processes control. From experiments which have been performed can be concluded that type of writing method to OPC has not so big influence on control quality coefficient. Asynchronous writing usually lead to better results, especially when network traffic is big.

There was not a big difference between reading from the device and from the cache, but it should be remembered that the executed tests exposed that asynchronous reading with Matlab/Simulink results in significally worse control quality then other methods of reading.

The conducted experiments prove that prepared model of close loop system with OPC is very precise. This model can be use for fast prototyping of industrial systems. In addition, it provides possibility to verify changes in working installation before they will be applied in real plant installation. It is sure that the importance of simulation for control systems development will increase in the future. Tools for process simulating will become more scalable and effective.

There are several ways to reduce the negative impact of the delays which OPC introduces into control loop. A part of performed tests was also an attempt to implement a Smith predictor. The results of experiments shown that in fact this algorithm is working, but for unstable processes it should be used carefully because it can result in overall control system instability.

Additionally, in one of the experiments, Excel was used to collect data from OPC. A simple SCADA application was prepared. An ActiveX component was used. The experiments shown that there is almost no difference between

regular OPC wrapper and the ActiveX component. The most time consuming operation was writing to Excel cell.

At the end, to check the possibilities of using OPC on non-windows platforms, the experiment with QNX operating system was performed. With simple gateway program running on Windows it was possible to interconnect OPC server running on Windows and process simulation running on QNX.

To sum up, the experiments described in this paper confirm that OPC technology provides easy and reliable way for distributed control over network. It should be pointed that no additional programming afford was needed to perform all described tests. They have been conducted with using of-the-shelf software and hardware. Dedicated application has been only prepared to be a simulation model of the process for DAPIO card and as QNX process simulation.

8. Appendix

8.1. Process simulation by DAPIO card

```
//vector of right hand sites of process differential equations
void rhs(double *u,double *x,double *outputs)
{
    const double p1 = -0.5;
    const double p2 = 1;
    const double p3 = 1;
    const double T1 = 1;
    const double T2 = 0.2;
    const double T3 = 0.1;
    const double k = 1;
    outputs[0]=(-p1*x[0]+k*u[0])/T1;
    outputs[1]=(-p2*x[1]+x[0])/T2;
    outputs[2]=(-p3*x[2]+x[1])/T3;
}

//process simulation thread
void process_thread()
{
    FILE * pFile;
    double dh = 0.001;
    double h2=dh/2;
    double h3=dh/3;
    double h6=dh/6;
    double
u[USIZE],x[VSIZE],dx1[VSIZE],dx2[VSIZE],dx3[VSIZE],dx4[VSIZE],x1[VSIZE],x2[VSIZE],x3[VSI
ZE];
    double tmp[VSIZE];
    double u_mem[1000],x_mem[1000];
    int k,i,j;
    x[0]=0;
    x[1]=0;
    x[2]=0;
    j=0;
    OpenConnection(); //open DAPIO card

    //wait for start of process controling
    while(ReadValue()<2){}

    for (i=0;i<100000;i++)
    {
        u[0] = ReadValue(); //read one value from DAPIO card reading is done each 1ms

       //one value per 100 is saved in memory (one per 100ms)
        if ((i % 100)==0)
        {
            u_mem[j] = u[0]*10;
            x_mem[j] = x[2]*10;
            j++;
        }

       //RK4 solver implementation
        rhs(u,x,dx1);           //dx1 = rhs(u,x(i,:),t(i));
        //x1 = x(i,:) +h2*dx1;
        for (k=0;k<VSIZE;k++)
        {
            x1[k]= x[k] + h2*dx1[k];
        }

        rhs(u,x1,dx2);          //dx2 = rhs(u,x1,t(i)+h2);

        // x2 = x(i,:) +h2*dx2;
        for (k=0;k<VSIZE;k++)
        {
            x2[k]= x[k] + h2*dx2[k];
        }

        rhs(u,x2,dx3);           //dx3 = rhs(u,x2,t(i)+h2);

        // x3 = x(i,:) +h*dx3;
```

```
        for (k=0;k<VSIZE;k++)
        {
            x3[k]= x[k] + dh*dx3[k];
        }

        rhs(u,x3,dx4);            //dx4 = rhs(u,x3,t(i+1));

        // x(i+1,:) = x(i,:) + h3*(dx2+dx3) + h6*(dx1 +dx4);
        for (k=0;k<VSIZE;k++)
        {
            x[k] = x[k] + h3*(dx2[k]+dx3[k]) + h6*(dx1[k]+dx4[k]);
        }

        WriteValue(x[2]); // write process value to DAPIO card
    }
    WriteValue(0); //set DAPIO card output to 0V
    CloseConnection(); //close DAPIO card

    //save process values to the file understandable by MATLAB
    pFile = fopen ("myfile.txt","w");
    if (pFile!=NULL)
    {
        fprintf(pFile,"a=[");
        for (i=0;i<1000;i++)
        {
            fprintf(pFile,"%f %f;",u_mem[i],x_mem[i]);
        }
        fprintf(pFile,"];");
        fclose(pFile);
    }
}
```

8.2. Matlab PID optimization algorithm

```
%File optimize.m
%Optimize PID parameters with tuned parameters method
global pid_P pid_I pid_D pid_TL o_o;

%initial PID parameters
pid_P=1;
pid_I=10;
pid_D=0;
pid_TL=200;

%vector of optimized parameters
x0=[pid_P,pid_I,pid_D]

%configure optimization procedure
oo=optimset('Display','on','TolX',.0001,'TolFun',.0001,'MaxFunEvals',1000,'Maxiter',1000
);

%start optimization procedure, control quality is calculated by ctrl_quality.m
fminsearch('ctrl_quality',x0,oo);
```

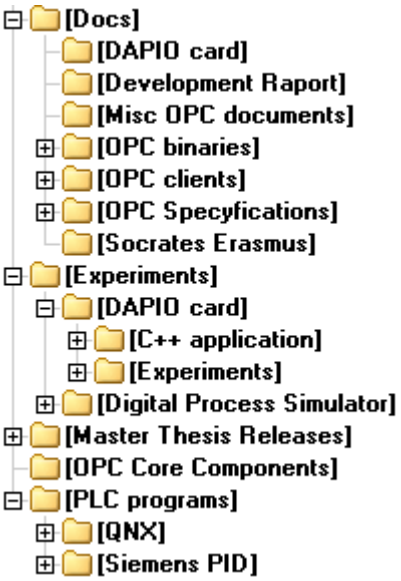
```
%File ctrl_quality.m
%control quality coefficient for tunned parameters method
function [error]=ctrl_quality(x0);

global pid_P pid_I pid_D

%copy actual PID parameters, they will be used in Simulink model
pid_P = x0(1);
pid_I = x0(2);
pid_D = x0(3);

if (pid_D < 0)
    pid_D = 0;
end;

if (pid_I >= 1e10)
    pid_I = 1e10;
```

```
end;

%start the simulation in simulink
sim('th4_test_optim_matlab.mdl');

%control quality is being calculated in simulink. Copy it to function return value
error=Quality(end)
```

8.3. List of Tables

8.4. List of Figures

8.5. Content of the CD



8.6. Glossary

AMT - ***A****djusted* ***M****odel* ***T****echnique*. The method base on the process/object model aims to find minimum of the control quality function by changing the model; parameters.

API - ***A****pplication* ***P****rogramming* ***I****nterface* is a specification of ways the one part of software can communicate with the other part in order to support the building of applications.

CAD - ***C****omputer-****A****ided* ***D****esign*. A use of computer technology in designing process. Nowadays due to the increase in computers performance, various CAD software packages became a major tool for engineers for designing, simulating and optimization.

COM/DCOM – *(****D****istributed)* ***C****omponent* ***O****bject* ***M****odel*. The standard introduced by Microsoft to enable reuse of software components and simple communication between them with no knowledge of their internal implementation.

DCS - ***D****istributed* ***C****ontrol* ***S****ystem*. A control system in which controllers are not located in one central location but rather distributed. Each sub-system is controlled with one or more controllers.

DDE - ***D**ynamic **D**ata **E**xchange.* Technology created by Microsoft in late 1980s to supply data exchange between Windows applications in unified way.

DHCP - ***D**ynamic **H**ost Configuration **P**rotocol.* A network application protocol used by devices to obtain configuration to operate inside an Internet Protocol (IP) network.

ERP - ***E**nterprise **R**esource **P**lanning.* A software system used to manage and coordinate all the resources, information, and functions at the company floor.

FPGA - ***F**ield **P**rogrammable **G**ate **A**rray*. A semiconductor device which internal logic can be programmed by the user in such way it is possible to create a very complex structure (i.e. software processors) from very simple blocks (AND, XOR gates)

GUI – ***G**raphic **U**ser **I**nterface* is a type of User Interface by which electronics devices (computer, hand-held, mp3) can communicate with the user in graphical form.

HMI – ***H**uman **M**achine **I**nterface*. A set of means by which user interact with the system

IPC – ***I**ndustrial **P**ersonal **C**omputer.* A family of computers customized for industrial usage. Those computers are based on these same components, mostly use the same software, but offers features different from regular PCs in terms of reliability, compatibility, expansion modules and long-term supply.

MES - ***M**anufacturing **E**xecution **S**ystem.* Software for managing and monitoring work-in-process on a factory floor. The aim of those systems is to provide automatic flow of the information at the factory floor. Nowadays it is usually embedded into ERP systems.

OPC - ***O**LE for **P**rocess **C**ontrol* which stands for Object Linking and Embedding (OLE) for Process Control. The OPC Specification aim was to provide a common way for real-time plant data communication between control devices from different manufacturers. It is based on developed by Microsoft COM and DCOM technologies.

PAC - ***P****rogrammable* ***A****utomation* ***C****ontroller.* Device that combines the features of a PC-based control system with programmable logic controller (PLC) so it provides not only the reliability of a PLC, but also computing power of a PC.

PLC - ***P****rogrammable* ***L****ogic* ***C****ontroller* – a digital computer used for automatic control of industrial processes. Those devices are more resistant for harsh industrial conditions and are more accurate for real-time applications

SCADA - ***S****upervisory* ***C****ontrol* ***A****nd* ***D****ata* ***A****cquisition*. This term refers to a centralized, computer based industrial monitoring and control system which supports Operators in manufacturing process.

SNMP - ***S****imple* ***N****etwork* ***M****anagement* ***P****rotocol.* Protocol designed for administration of devices connected to Internet Protocol network.

SSL - ***S****ecure* ***S****ockets* ***L****ayer* is a cryptographic protocol to provide security communication over networks such as the Internet

UPS - ***U****ninterruptible* ***P****ower* ***S****upply.* The device or system which aims to ensure ustopped power supply to other devices. It usually equipped with battery pack, in some cases – when large power is required, it is supported with diesel power generators

VPN - ***V****irtual* ***P****rivate* ***N****etwork*. A computer network in which some links between nodes are made with secured channels within some larger network like Internet