POLITECNICO DI TORINO

**Master's Degree in Electronic Engineering**

Master's Thesis

# Design and analysis of a communication system for industrial monitoring in the textile industry

**Supervisor**
prof. Renato Ferrero

**Candidate**
Michael Piran

Accademic Year 2020-2021

*Ogni traguardo lo devo a mio padre e a mia madre*

# Acknowledgements

Ringrazio il professor Renato Ferrero per avermi seguito in questa attività di tesi. Dedico questo traguardo ai miei genitori. Sarò infinitamente grato per il loro aiuto e sostegno. Ringrazio mio fratello Samuele, per essere sempre un punto di riferimento e un esempio da imitare. Gran parte del merito lo devo a mio fratello Cristian per aver condiviso anche questa avventura. A mia zia Cristina, un immenso grazie per avermi guidato durante gli anni accademici e per essere sempre disponibile ad aiutarmi. Un grazie affettuoso ai nonni per essermi sempre stati vicini. Al piccolo Mattia, auguro di poter esaudire i propri sogni. Ringrazio Adelaide per avermi accompagnato in questi ultimi anni.

# Contents

# List of Tables

# List of Figures

# Chapter 1

# Introduction

## 1.1 Scenario

The master's thesis activity was carried out at *Officine Gaudino spa* from the first of July to the 30th November 2021, situated in *via XXV Aprile, 16, Cossato(BI).*

#### Company description

Officine Gaudino is located in Biella which is famous for being a textile hotspot where the world's greatest wool spinners are located. The company was born around 1920 and it is specialized in the production of ring spinning machines for wool, fine and synthetic fibers. In the last few years Officine Gaudino acquired the brand LEMA LEZZENI s.r.l., known worldwide for the production of twisting machines, cops winders, fancy yarn twisters and drawing frames. Today Officine Gaudino offers a wide range of textile machinery such as Ring Spinning, Cops Winder, Doubling and Twisting, Twisting Doubling and Re-Twisting, and Drafting machines.

#### Thesis goal

The goal of the thesis is to design a communication system that allows to interface a production line to a central system. There is a PC or a PLC (Programmable Logic Controller) that collects and modifies the key parameters and monitors the production progresses. The communication system should be compatible with hardware devices used in Officine Gaudino and should be able to work in a textile environment.
The necessity of this kind of activity arises because customers require a production line that can be managed by a central system and can be interfaced with MES(Manufacturing Execution System) systems.

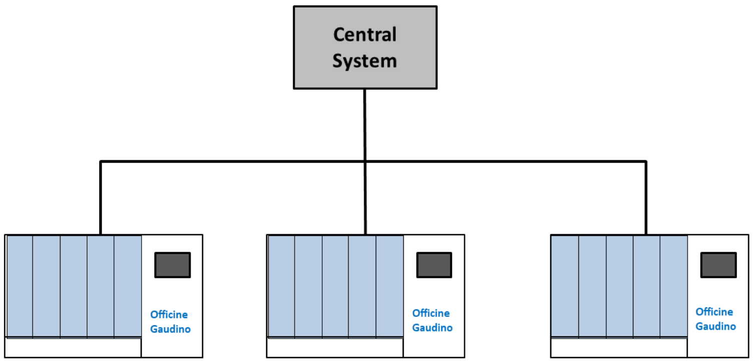


Figure 1.1: System Description

*Figure 1.1* gives a schematic overview of the system. Production line machines are connected to a central system placed in the office room of the customer company. The central system reads key parameters from PLCs inside each machines, so it can monitor the real time status of the production, and should be able to change setting parameters. In this way users have a complete overview on what is happening in the production line, managing the technical information and reducing the human effort. The key parameters that are monitored or can be modified are:

- Information on the production progresses such as remaining time, end of production, package weight and meters of yarn processed;
- Properties of processed material such as yarn count, yarn ironing and final package size;
- Error signals;
- Actuator settings such as speed of motors, positional information from encoders, electrical absorptions (currents, voltages, electrical powers) and torques developed;
- Recipes management. A recipe contains the settings parameters required by machines to process a specific material. This allows to easily process different kind of yarns, because users just have to choose the right recipe.

**Requirements**

The system will be employed on a textile industrial environment, such as spinning mill and winding. These places are characterized by high temperature, dust and dirt due to wool processing, presence of chemical agents and vibration due to machinery in the production line. So the communication system needs to satisfy some base requirements:

- Few data losses;
- Electrical noisy resistant;

- Interoperability, i.e., ability to communicate with other devices from different vendors and with different communication protocols;
- Dirty environment resistant;
- Vibration resistant;
- Time-critical communication;
- Security.

The most important requirement is interoperability. It is necessary a cross-platform communication protocol that allows an easy communication among different vendor-specific devices. Furthermore, it is crucial to adopt a protocol widely diffused in automation industry environments. In this way it is possible to integrate Officine Gaudino's machines with production machines of other brands.
The purpose is monitoring the production, so the environment conditions are not too hostile, because the system is not placed on board, near sensors and actuators. This situation allows to have a less resistant transport protocol that can reach high speed performance, such as Ethernet.
Communication delay is not a fundamental prerequisite, because a real-time communication is not required. The central system monitors the production progresses and the decisions taken involve the modification of setting parameters of the processed material. Their effects are not instantaneous because usually they modify speed of motors or actuators.
Another important parameter is security. The implementation of security expedients is important to avoid external attacks, especially if the system is integrated with MES equipment.

### Hardware and Software

Here a description of the hardware and software tools adopted by Officine Gaudino is presented. The company adopts Hardware devices from Eaton[1] and these components are employed in the project:

- PLC: *Eaton XC-300*:
  - CPU: 960MHz, ARM CortexA7 Dual Core;
  - Memory: 512 MB DRAM, Flash 1 GB internal memory and 128 kB non-volatile data memory;
  - Supported protocol: CANopen, Modbus, EtherNet/IP, OPC-UA, easyNet, EtherCat;

  Datasheet available on PLC Datasheet.

[1] https://www.eaton.com/it/it-it/products.html

- HMI (Human Machine Interface): *Eaton XV-303*:
  - HMI with an incorporated PLC;
  - CPU: 800MHz, ARM Cortex-A9;
  - Memory: 512 MB DRAM, Flash 1 GB internal memory and 128 kB non-volatile data memory;
  - OS: Windows Embedded Compact 7 operating system;
  - Supported protocol: TCP/IP, Profibus, CAN, Modbus, EtherNet/IP, OPC UA;

  Datasheet available on HMI Datasheet.

The company adopts the following software:

- *XSOFT-CODESYS v3.5.16* for the PLC programming;
- *Galileo v10.5.1* for programming the graphic interface of the HMI.

Based on the hardware specifications, these three protocols are considered for developing the communication system: **Modbus TCP, EtherNet/IP** and **OPC UA**.

### Document organization

The document presented is organized in this way;

- Section 1.2 illustrates the basic theory concepts of automation and a general overview of Ethernet and PLC technologies;
- Chapter 2 gives an overview on the communication protocols considered. In particular, it describes Modbus TCP, EtherNet/IP and OPC UA specifications;
- Chapter 3 indicates the criteria used to individuate the best communication protocol for this project;
- Chapter 4 describes the project implementation;
- Chapter 5 gives a general assessment of the work done, describes the difficulties encountered and explains the future developments of the project.

## 1.2 Theoretical aspects

The theoretical concepts touched by the following chapters are introduced in this section. A particular attention is given to automation, PLC, HMI, Ethernet, protocol layering and socket programming.

### 1.2.1 Automation industry - Basic concepts

A production system can be defined as a group of people, equipment and organized procedures with the goal of carry out manufacturer industry operations[1]. There is an interaction between humans and machines. Considering the human role in the automatic processes, three categories can be highlighted:

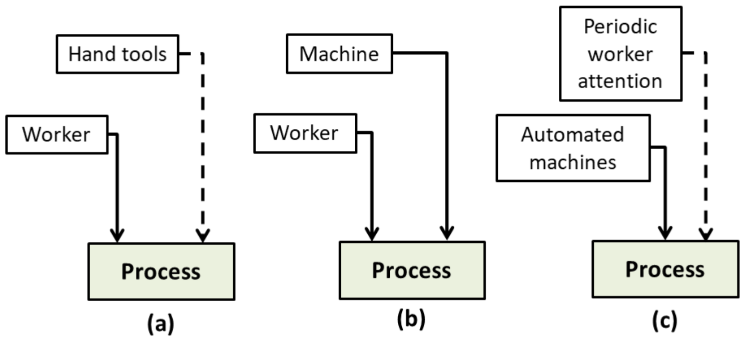


Figure 1.2: Manufacturing systems's categories

a. *Manual work systems* (*Figure 1.2a*): Humans work only with manual tools, without the help of machines;

b. *Worker-machine systems* (*Figure 1.2b*): Workers do their jobs together with the production machines;

c. *Automated systems* (*Figure 1.2c*): A process is performed without a direct human participation. There are two different levels:

    1. *Semi-automated machine*: During the machine's working cycle, the worker performs small operations that allow to move forward the production;
    2. *Fully-automated machine*: For a long time, more than one working cycle, the machine can works without the human participation.

There are some advantages from the introduction of automation in manufacturer industry:

1. *Increasing productivity*;
2. *Reducing products costs*;
3. *Improving worker safety*;
4. *Reducing or eliminate routine manual and clerical tasks*;
5. *Reducing manufacturing lead time*;
6. *Improving products quality*;
7. *Accomplishing processes that cannot be done manually.*

*Pyramid of Automation (Figure 1.3)* is defined as a representation of different automation levels inside the industrial environments. Nowadays trend is to connect and exchange information between all these different levels, starting from the field level toward the office level. The idea of level connection is not new, already in 1970 the *Computer-Integrated Manufacturing (CIM)* theory was conceived. There are different definitions for that concept, some of them emphasize the role of information, other's the importance of integration:

- *CIM is the application of computer science technology to the enterprise of manufacturing in order to provide the right information to the right place at the right time, which enables the achievement of its product, process and business goals*[2];

- *CIM is the integration of the total manufacturing enterprise through the use of integrated systems and data communications coupled with new managerial philosophies that improve organized and personnel efficiency*[3].

Overall there are different agreements on the name and number of levels, but the main approach is to detect six functional levels: Field level, Control level, Supervisory level, Planning level, Enterprise level and Cloud. They are organized in a pyramidal way, the lowest levels manipulate signals from controllers, sensors and actuators employed in a production line. A very detailed description of the system is required and information change frequently in time. On the other hand, higher levels have a more general description of the system, they manage information on the status and progresses of the production line. This kind of data change less frequently in time and they are manipulated by the company management. Finally a multi-level network is developed to allow the data exchange among different levels. Communication over higher layers is IP network dominant over Ethernet, while fieldbusses are involved in field levels. In the future, Ethernet networks will be involved also in lower level, because the real-time problem of today standards will be overcame. Here the six automation levels are described:

[2]Digital Equipment Corporation (DEC) (Ayres, 1991)

[3]Computer and automation Systems Association of the Society of Manufacturing Engineers (Singh, 1996)

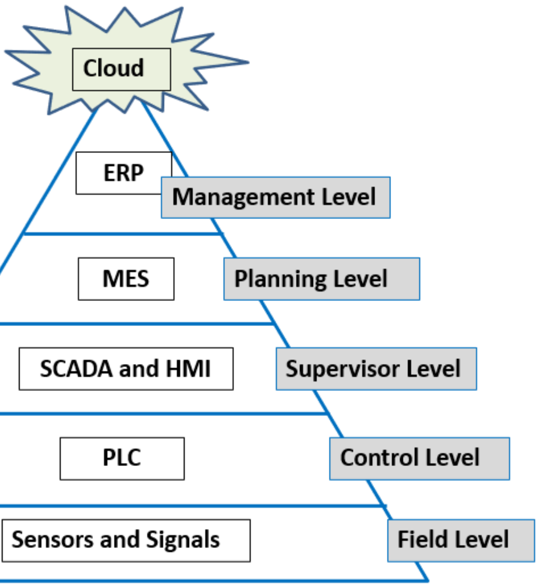


Figure 1.3: Automation Pyramid

- *Level 0 - Field Level*:
  It is the lowest level, also called the machine level. It manages devices like sensors and actuators (electric motors, hydraulic pump, etc.), so in this level the physical work is performed;

- *Level 1 - Control level*:
  *PLCs* are present in this level. They receive electrical informations from sensors and input devices. They manipulate data and provide output signals to the actuators;

- *Level 2 - Supervisory Level*:
  Supervision and monitoring tasks are performed. Acquisition systems such as *SCADA (Supervisory Control And Data Acquisition)* or user interfaces like *HMIs* manage the information related to lower levels;

- *Level 3 - Planning Level*:
  Management systems like *MES* are employed in this level. They monitor the entire production process of a company, from the raw material to the final product. So management receives these real-time informations, and get familiar with the status of production resources and quantity of goods in stock;

- *Level 4 - Enterprise Level*:
  It requires information from all the lower levels. *ERP (Enterprise Resource Planning)* systems are employed in this level. They are management software that can be used to collect, store, manage and interpret data from many business activities;

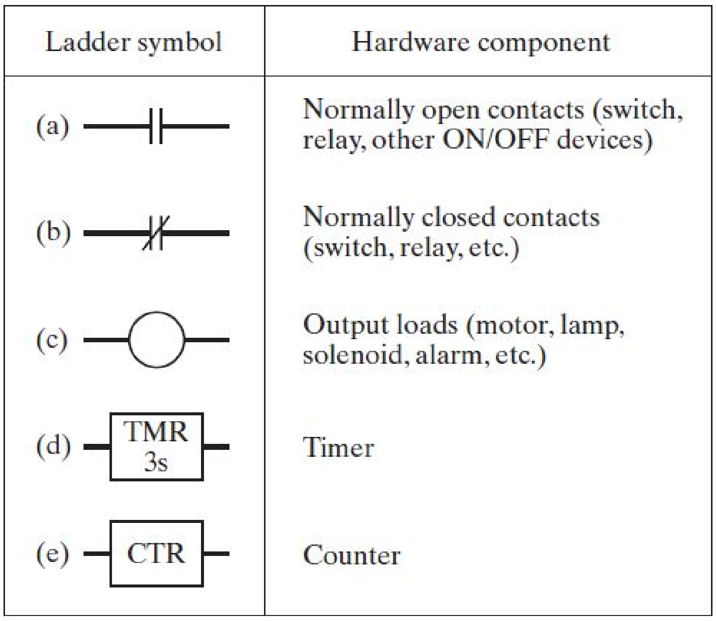
- *Level 5 - Cloud*:

Data coming from lower levels are directly sent to the cloud. This provides an easily access to data and the possibility to implement web services.

### 1.2.2 PLC and HMI

PLCs are controllers that implement logic functions for the management of industrial machines and processes. They can operate in electrical noisy environments. The base components are: processor, memory unit, power and I/O modules. PLC scans input signals from sensors, elaborates them based on a set of logical instructions stored in a programmable memory and provides output signals to control actuators. There is a scan cycle that defines the sample time of the inputs and how frequently the program is executed. Standard IEC 61131-3, *International Standard for Programmable Controllers*, defines five languages for PLC programming, three are graphic-based and the other two are text-based.

1. *Ladder logic*: There is a set of symbols (*Figure 1.4*) that directly represent the devices and their connection to the PLC, like relays, contacts, solenoids, timers, counters and motors;



| Ladder symbol | Hardware component |
|---|---|
| (a) | Normally open contacts (switch, relay, other ON/OFF devices) |
| (b) | Normally closed contacts (switch, relay, etc.) |
| (c) | Output loads (motor, lamp, solenoid, alarm, etc.) |
| (d) TMR 3s | Timer |
| (e) CTR | Counter |

Figure 1.4: Ladder logic

2. *Function block diagram*: The instructions are made by function blocks implementing specific functions that manage inputs and outputs;

3. *Sequential function charts*: It graphically represents sequential functions through series of transition from one state to the next;

4. *Instruction list*: Text-based, low level programming language used to build ladder logic diagrams. The typical set of instructions adopted is: *AND, OR, NOT, STR(store in the accumulator), TMR(timer), CTR(counter)* and *OUT(output the data)*;

5. *Structured text*: Ladder logic or Instruction list are limited to ON/OFF signals. This is an high level programming language that allows to manage more complex data structures. It is also possible to implement more complex control algorithms and communicate with computer-based systems.

HMIs are user-friendly panels used in industrial systems that interface humans with the machinery. They translate PLC signals to human informations, so the operator can monitor production processes. Typically they are touch-screen devices with an operating system.

### 1.2.3 Ethernet technology

Communication network can be divided in two categories[3]:

- *Connection-oriented*: also called *Circuit-switched.* There is dedicated connection between transmitter and receiver. It is guaranteed that messages reach the final destination and the network performance is not touched by other connections. The main drawback is the cost: there is a fixed connection, so costs are fixed independently on the network use;

- *Connectionless*: also called *Packet-switched.* Data transmitted are divided into packets. They are composed of hundreds of bytes and all of them have a destination address. More connections can coexist in parallel, in other words the physical network is shared among multiple devices, so less networks, lower costs. The drawback is the degradation of performance due to bandwidth occupation.

Considering the distance covered, packet-switched networks can be divided in two categories:

- *Wide Area Networks (WANs)*: They provide a communication over a long distances and travel through continents and oceans. They typically work at lower speed, from 1.5Mbps to 155Mbps. The connection delays are from few milliseconds to tens of seconds;

- *Local Area Networks (LANs)*: They are shorter connections and can reach higher speed, from 10Mbps to 2Gbps. Delays are around few milliseconds.

Ethernet technology, standard IEEE 802.3, is used in packet-switched LAN networks. There are different version of Ethernet cables:

1. *10Base5*: Ethernet over coaxial cable;

2. *10Base2 (thin-wire Ethernet)*: cheaper than previous version;

3. *10BaseT*: Ethernet over twisted wires and without ground shield, 10Mbps;

4. *100BaseT (Fast Ethernet)*: 100Mbps;

5. *1000BaseT (Giga Ethernet)*: 1Gbps.

Ethernet is a bus technology, where each device is connected to the shared bus. The access protocol of Ethernet is called *Carrier Sense Multiple Access with Collision Detect (CSMA/CD)*. CSMA means that more than one device can simultaneously access Ethernet, a carrier signal determines if the bus is empty in a specific instance and a transmission of data can start. CD means that when a data collision arises, the host stops the transmission, waits for a little interval of time and then tries to retransmit.
An Ethernet address is 48 bits length and three categories can be distinguished:

- *Unicast address*: it indicates a physical device in a network;

- *Broadcast address*: it is used for sending messages to all connected devices. All address bits are at one;

- *Multicast address*: it sends messages to a group of devices connected.

The Ethernet frame is organized in this way:

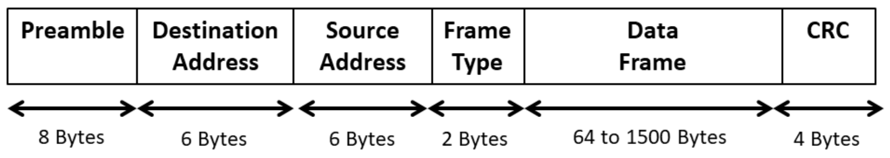


Figure 1.5: Ethernet frame

- *Preamble*: 8 Bytes. Allows sender and receiver to establish a synchronization;

- *Destination address*: 6 Bytes. Destination address of the communication;

- *Source address*: 6 Bytes. Source address of the communication;

- *Frame type*: 2 Bytes. Indicates the protocol used to process the frame;

- *Frame data*: 46 to 1500 Bytes. Data field;

- *Cyclic Redundancy Check (CRC)*: 4 Bytes. Detects possible frame's transmission errors. The transmitter evaluates this code, the receiver, with the received data, recalculates it. If the two CRC are different it means that an error arises.

### 1.2.4 Protocol Layering

A communication protocol indicates how data are exchanged between receiver and transmitter and how errors, delays, faults and congestion are managed[2]. In a complex communication systems, it is really hard to manage with a single protocol all these aspects of the communication. So the entire communication management is organized in different levels. Each of them is implemented with a single protocol and provides services to its neighbours. In this way the network developer's job is simplified, because it is focused only on a sub-aspect of the communication.
Considering the communication between two devices, the information exchanged are written in the highest layer. Then they are translated in the lowest layer and sent over the physical network. This procedure is called *Encapsulation*, the transmission frame is created by adding detailed fields to the originally frame. The receiver of the communications, performs the inverse of the encapsulation until it retrieves the original packets.

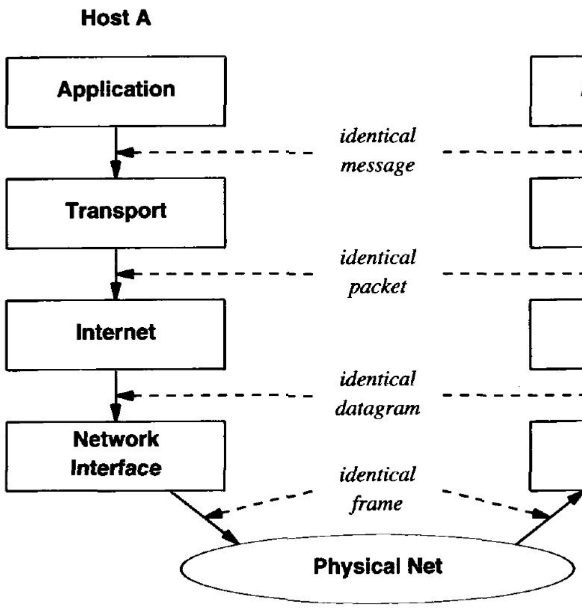


Figure 1.6: OSI information transmission

#### OSI/ISO Model

OSI model was developed in'70 by *International Organization for Standardization* and it became a standard, ISO7498. OSI stands for *Open Systems Interconnections.* It helps the network developers in the implementation of communication protocols. This is not a strictly set of rules, but only general guidelines that a protocol should follow. Complex communication systems exchange information with devices from different vendors, so it is necessary a reference model hardware independent. If each manufacturer develops a communication device with its own protocol, it is impossible to communicate with other vendor-specific devices. So to avoid this protocol confusion, OSI inserts structure uniformity. Here the seven layers of the model are described:

- *Level 1: Physical Layer*
  It takes into account the physical part of the communication. It exchanges unstructured raw data through a transmission medium. It considers all electrical and mechanical characteristics of the communication, which include tasks like activating, maintaining, and terminating the physical connection. Typical protocols example are: *Bluetooth*, *RS-232*, *OTN (optic fiber)* and *RS-485*;

- *Level 2: Data-Link Layer*
  This level guarantees an error-free connection between two nodes. It uses error correction algorithms to allow reliable transmission through the physical layer. It also adds source and destination address to the packets that are transmitted. Typical protocols example: *Ethernet*, *Wi-Fi* and *Point-to-Point Protocol.*

- *Level 3: Network Layer*
  It is responsible of packets routing. Through specific algorithms, this level defines the optimal path that packets should follow to reach the destination address. The network layer also is responsible for establishing and terminating network connections and re-establishing broken network connections. *IP* protocol is a typical example.

- *Level 4: Transport Layer*
  Differently from previous levels, this one works only with starting and end users. It is responsible of the flow of data exchanged between users and assigns logical addresses to the physical addresses that are used in the network layer. *TCP* and *UDP* are typical protocols example, both IP-based.

- *Level 5: Session Layer*
  This level guarantees that final users use the same session protocol, in order to establish a connection. The participants first have to negotiate a common protocol, which is then used in the communication. *Socks* is an example.

- *Level 6: Presentation Layer*
  This level defines how the information shall be formatted in order to make it understandable for the end user. If, for example, a string is transmitted, this level recognizes the information and interprets the bytes. So it manages the syntax of the frames transmitted. Conversion of data is covered here as well as optional encryption of information.

- *Level 7: Application Layer*
  Provides higher-layer services and interfaces to the application. This layer strongly depends on the application and operating system used. *SSH*, *IMAP* and *FTP* are typical protocols examples.

### TCP/IP suite

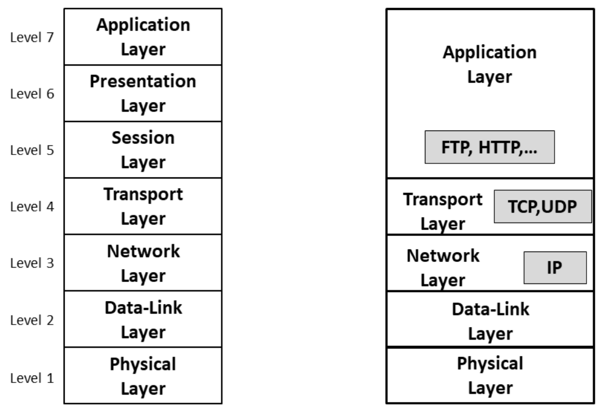


Figure 1.7: ISO/OSI and TCP/IP models

This layering model does not come from standard organization, but directly from internet communication protocols. TCP/IP suite is organized into five levels, four software and one hardware. *Figure 1.7* shows the differences between the OSI and the TCP/IP models. Here the levels are described:

- *Data-Link Layer and Physical Layer*: Protocols present in these lowest levels are related to the physical network employed (Ex: Ethernet). They accept and transmit IP frames over specific network;
- *Network Layer*: Internet layer handles the communication between two end-points. It encapsulates packets in a frame, uses routing algorithms to determine whether to deliver the datagram directly or send it to a router. Besides it checks the correctness of input packets and if they need to be processed internally or forwarded;
- *Transport Layer*: Regulates the information flow between two end-points, and it ensures that data reach destination without errors and in the correct order. This is implemented through acknowledge signals and retransmission of packets loss. TCP and UDP are available protocols;
- *Application Layer*: It is the highest level, it invokes application programs that allow to access TCP/IP services. The application program sends data in the required form to the transport layer. HTTP, FTP and DNS are typical protocols example.

## 1.2.5 Socket Programming

Focusing on Transport and Network Layer of the OSI Model (or also the TCP/IP Suite), *TCP* and *UDP*, over *IP*, are the most popular protocols adopted for network programming. The Internet Protocol (IP) is a Network Layer protocol that allows the exchange of data through an IP Address, typically on 32 bits. The Transport protocol can be implemented through *Sockets*. They are standard API and they are based on TCP or UDP. *Berkeley sockets* is an example.

### TCP - Transmission Control Protocol

Provides a reliable connection between Clients and Servers. It is a connection-oriented protocol, it divides the data transmitted in packets called *segments*. It is guaranteed that data reach the final destination and in the correct order. In case of losses or corrupted data they are retransmitted.

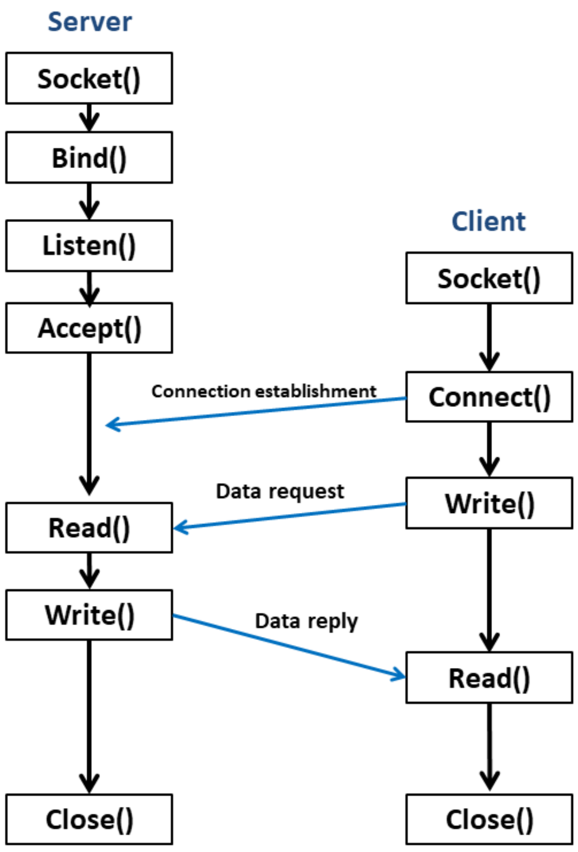


Figure 1.8: TCP socket programming

Before the data transfer, a connection between two endpoints is established. Server performs these steps:

1. Creates a socket with the function socket();
2. Through Bind function, it sets the port number and IP address;
3. The socket is in listening mode, waiting for a connection;
4. If the Client connects to the Server, it accepts the connection;

5. Exchanges segments through the Read and Write function;
6. Closes the socket.

On the other hand the Client:

1. Creates the socket;
2. Connects to the Server socket;
3. Exchanges data through Read and Write function;
4. Closes the socket.

### UDP - User Datagram Protocol

Provides a connection between two endpoints with a minimum overhead, because acknowledge and flow control of received data are not implemented. It is a connectionless protocol because there is not a constant connection between two processes. It is not guaranteed that packets reach the final destination and in the correct order. Data exchanged are called *datagrams*.

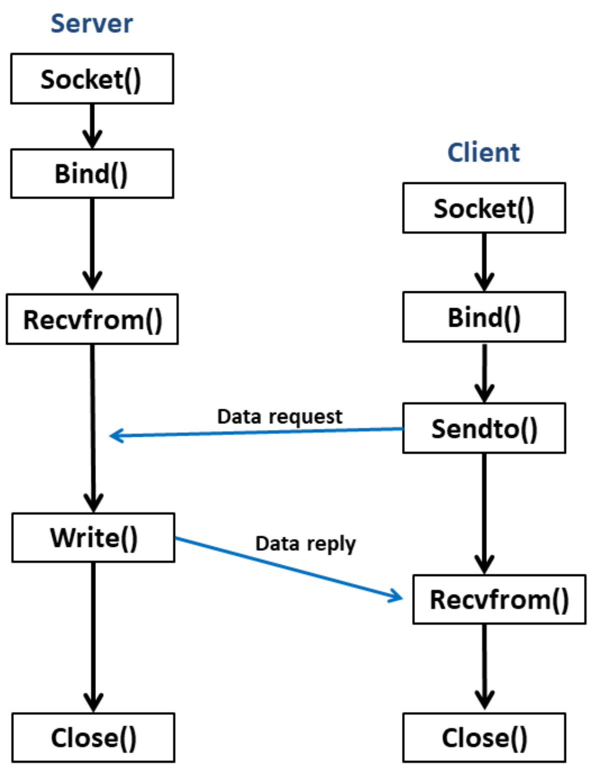


Figure 1.9: UDP socket programming

Server and Client create both new sockets, then with the Bind function they associate the sockets with an IP address and a port number. After this setup phase, Servers exchange data with Recvfrom and Write functions, while Clients use Send and Recvfrom. As the the *Figure 1.9* shows, there is no flow control between the datagrams exchanged.

# Chapter 2

# Communication protocols's overview

## 2.1 Modbus TCP/IP

*Modicon* created Modbus in 1979 and now the *Modbus Organization* follows the evolution of the protocol. This organization is a group of independent companies, that operate in the automation field, with the goal of updating the protocol and its variants. Besides they provide information to users and help them to implement Modbus applications.
Modbus is a *de facto* standard for industrial communication, it is continuing to grow and now can support more and more devices. One reason of this popularity is that this is a open source protocol, so everyone can obtain its specification just by simply following the instruction freely given by the manufacturer.
Today Modbus is present on different physical media and with gateways it is possible to connect different releases of the protocol:

- Modbus TCP/IP over Ethernet;
- Modbus Serial Line over EIA/TIA-232-E or EIA/TIA-485-A;
- Modbus Plus.

This chapter describes the main attributes of the Modbus TCP/IP, focusing on the application layer[4] and on the TCP/IP implementation[5].

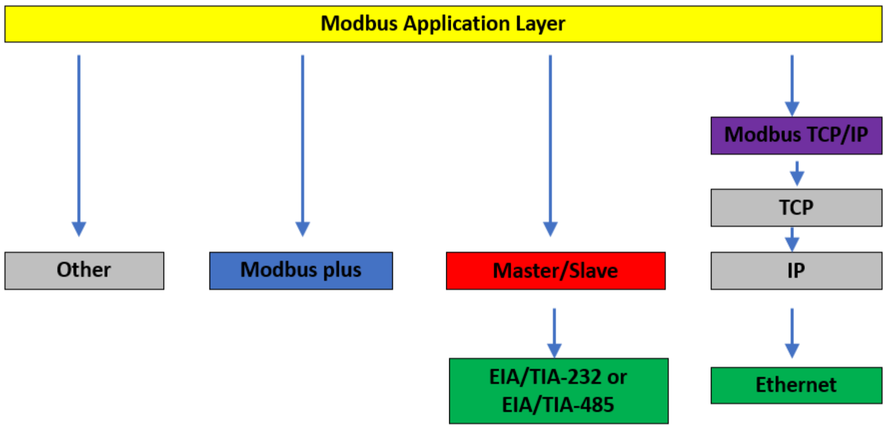


Figure 2.1: Modbus communication stack

Modbus can be defined as an application layer messaging protocol positioned at level 7 of OSI model. It allows the connection between different kind of devices, like PLC, HMI, Control Panel, Driver, Motion control and I/O Devices.

### 2.1.1 Protocol description

Modbus is a Client-Server protocol and exchanges information through the **registered port 502**. TCP frames are generally composed by a *Function code*, that indicates the actions to perform, and a *Data* field.
When the Client initiates the request, puts in the function code field what kind of action the Server should perform. The function code is represented on one byte and 0 is an invalid code. So valid data only if in the range 1 to 255, but range 128-255 are reserved. Specific functions require a *Sub-function code* to extend the previous field. The Data field of the message contains additional information that the Server requires for implementing the request, such as register addresses, number of elements to be handled or the number of data bytes. In some cases the data field is not required, depends on the function code.

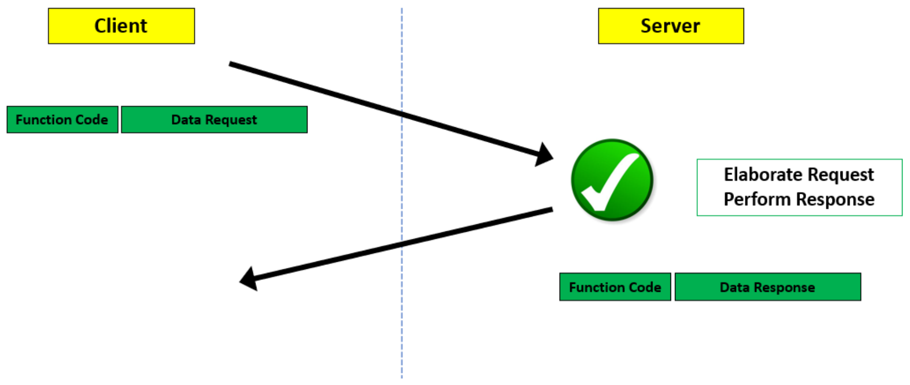


Figure 2.2: Modbus Transaction error-free

When the Server receives the frame, it elaborates the request, performs some action and creates the response. If no error occurs, the function field of the response echoes the code of the request, and the data field contains the information needed. Instead, if an error arises, the Server detects the problem and sends an Exception response. The function code contains the original code with the most significant bit (MSB) set to 1 and the data field contains an *Exception code* that indicates the cause of the error.

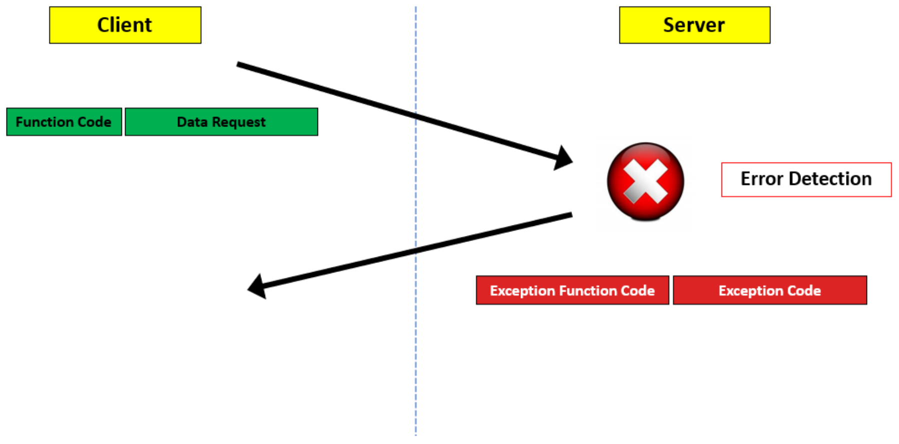


Figure 2.3: Modbus Exception response

*Figure 2.4* reports how the Server manage a Modbus transaction. Firstly it is listening port 502 for a Request. After that, it checks, in sequence, the correctness of Function Code, Data Address and Data Value. If it does not retrieve any errors, executes the function and sends a Modbus Response.

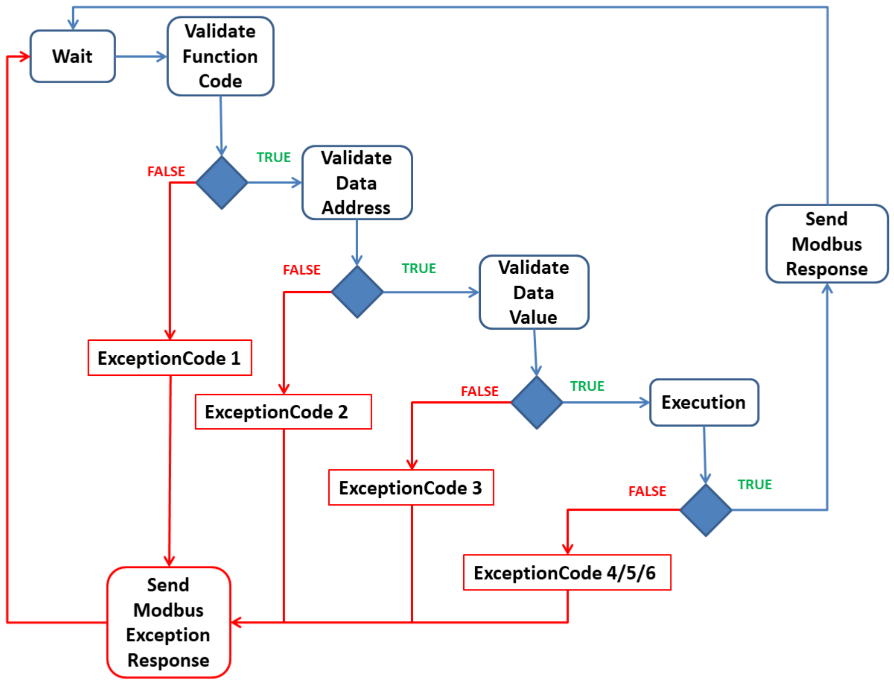


Figure 2.4: Modbus Transaction

### 2.1.2 Modbus Data model

#### Data Encoding

Data and addresses follow the *Big Endian* encoding format, so firstly the most significant byte is sent. For example, if the word 0x0123 needs to be send, first the byte 0x01 is sent and then the 0x23.

#### Memory Organization

Modbus provides a data model for Servers, that behaves as interface between an external request and the physical memory of the device. There are four distinct data references, listed in the table below. Two of them are organized on 16 bits and the remaining on 1 bit. Some data are Read-Only and can be provided by an I/O system, while others are Read-Write and can be modified by an application program.

| Name | Register Size | Access Permissions | Data Type |
|---|---|---|---|
| Discrete Inputs | 1 bit | Read-Only | Boolean |
| Coils | 1 bit | Read-Write | Boolean |
| Input Registers | 16 bits | Read-Only | Unsigned Word |
| Holding Registers | 16 bits | Read-Write | Unsigned Word |

Table 2.1: Modbus Memory Area

The protocol provides 65536 data items for each memory area. There is a mapping policy, totally vendor-specific, that allows to bound these data references to the physical address of the device. So a Client request, that want to access data of the Server device, needs to address one of this interface.

**Function Code**

It is a 1 Byte field, so can support up to 255 different codes. There are three categories:

- *Public Function Codes*: They are well defined codes, certificated by Modbus Organization and publicly documented. Users cannot change their order;
- *User-Defined Function Codes*: Users can implement their own function code but it is not supported by the specification and it is not guaranteed the uniqueness;
- *Reserved Function Codes*: They are reserved for companies, so not all users can access them.

| Name | Function Code | (hex) | Sub Code | Info |
|---|---|---|---|---|
| Read Coils | 01 | 01 | | Single bit data access |
| Read Discrete Inputs | 02 | 02 | | Single bit data access |
| Read Holding Registers | 03 | 03 | | 16 bits data access |
| Read Input Register | 04 | 04 | | 16 bits data access |
| Write Single Coil | 05 | 05 | | Single bit data access |
| Write Single Register | 06 | 06 | | 16 bits data access |
| Read Exception status | 07 | 07 | | Diagnostic |
| Diagnostic | 08 | 08 | 00-18, 20 | Diagnostic |
| Get Com event counter | 11 | 0B | | Diagnostic |
| Get Com Event Log | 12 | 0C | | Diagnostic |
| Write Multiple Coils | 15 | 0F | | Single bit data access |
| Write Multiple Registers | 16 | 10 | | 16 bits data access |
| Report Slave ID | 17 | 11 | | Diagnostic |
| Read File record | 20 | 14 | 6 | File record access |
| Write File record | 21 | 15 | 6 | File record access |
| Mask Write Register | 22 | 16 | | 16 bits data access |
| Read/Write Multiple Registers | 23 | 17 | | 16 bits data access |
| Read FIFO queue | 24 | 18 | | 16 bits data access |
| Read device Identification | 43 | 2B | 14 | Diagnostic |
| Encapsulated Interface Transport | 43 | 2B | 13,14 | Other |
| CANopen General Reference | 43 | 2B | 13 | Other |

Table 2.2: Modbus Function Code Description

### 2.1.3 Command descriptions

Function codes illustrated in the table above are all functions supported by Modbus protocol, but TCP/IP version supports only some of them.

- *0x01 - Read Coils*: Allows to read up to 2000 contiguous status of coils in a remote device;
- *0x02 - Read Discrete Inputs*: Allows to read up to 2000 contiguous status of discrete inputs in a remote device;
- *0x03 - Read Holding Registers*: Allows to read up to 125 contiguous blocks of holding registers in a remote device. Each register occupies two bytes, the first byte contains the high order bits and the second contains the low order bits;
- *0x04 - Read Input Register*: Allows to read up to 125 contiguous input registers in a remote device. Each register occupies two bytes, the first byte contains the high order bits and the second contains the low order bits;
- *0x05 - Write Single Coil*: Allows to set ON or OFF a single output in a remote device. Possible values in the data field are 0xFF00 for the ON state and 0x0000 for the OFF state;
- *0x06 - Write Single Register*: Allows to write a single holding register in a remote device;
- *0x0F - Write Multiple Coils*: Allows to set ON or OFF a sequence of contiguous coils in a remote device;
- *0x10 - Write Multiple Registers*: Writes a sequence of contiguous registers in a remote device, up to 120. Each register occupies two bytes, the first byte contains the high order bits and the second contains the low order bits;
- *0x14 - Read File record*: Allows to implement a read operation on a file record.
- *0x15 - Write File record*: Allows to implement a write operation on a file record.
- *0x16 - Mask Write Register*: Using an AND or OR mask, it allows to modify the content of a specific holding register. If the AND mask field is zero, the result is the logical OR between the mask and the current content of the register. If the OR mask field is zero, the result is the logical AND between the mask and the current content of the register;
- *0x17 - Read/Write Multiple Registers*: Performs a read and write operation on a sequence of contiguous holding registers. Each operation is performed up to 120 registers. The write operation is performed before the read;
- *0x18 - Read FIFO queue*: Allows to read the content of a FIFO queue, up to 32 registers.

### 2.1.4 Error Handling

Modbus provides features to manage transaction's errors. When Client sends a request message, if no errors occurs, the Server replies as expected. Unfortunately can happen that, due to a communication error, the Server does not receive the request or receives it but detects a parity or CRC error. Server is not able to return a response, so the Client can understand the critical situation only via timeout condition. It can happen that the Server receives the Request, no communication errors are present, but cannot manage it. So the Server returns an exception response indicating the cause of the error. In the exception response message, the function code field contains the function code of the original request but the MSB is set to 1. This makes the code with a value higher than 0x80, so the Client checking the MSB can recognize transaction error. Normal value of function code has MSB at 0. Data field contains the information required by the client, in normal situation. On the contrary, in a exception response, it contains an exception code showing the source of error. Here the exception errors are listed:

- *Illegal function*: The function code received is not correct. It can also indicate that the Server is not in the correct configuration to process a requests of this type;
- *Illegal data address*: The data address received points in a location that cannot be managed by the Server;
- *Illegal data value*: The data value received is not correct. Maybe its length does not correspond on what indicated in the Length field;
- *Slave device failure*: An unrecoverable error is occurred while the Server was attempting to perform the requested action;
- *Acknowledge*: A request is accepted, but it needs a long time to be processed. Server sends an acknowledge exception to inform the Client and avoid a timeout error;
- *Slave device busy*: The Server is processing other requests, therefore the Client needs to retransmit the message at a later time;
- *Memory parity error*: It can arise when there is a file record access. The Server wants to read a file, but detects a parity error in the memory;
- *Gateway path unavailable*: The gateway is not able to allocate a path from the input to the output for processing the request;
- *Gateway target device failed to response*: It indicates that no response was obtained from the target device, probably the device is not connected to the gateway.

## 2.2 EtherNet/IP

EtherNet/IP, introduced in 2000, is the Ethernet implementation of the CIP (Common Industrial Protocol) protocol[6][7][8]. *"IP"* stands for *"Industrial Protocol"*. This protocol

is managed by ODVA, that is a global association with the goal of developing communication technologies in industrial automation environments.
EtherNet/IP is an industrial communication system that allows industrial devices (sensors, actuators, PLCs and so on) to exchange information through TCP/IP networks. On *Figure 2.5* is showed the structure of the CIP protocol. It implements all the layers of the OSI model, and only the lowest four are network dependent. This because ODVA implements the CIP protocol also on other physical and transport layers such as *DeviceNet* on CAN technology, *ControlNet* on CTDMA technology and *CompoNet* on TDMA technology.

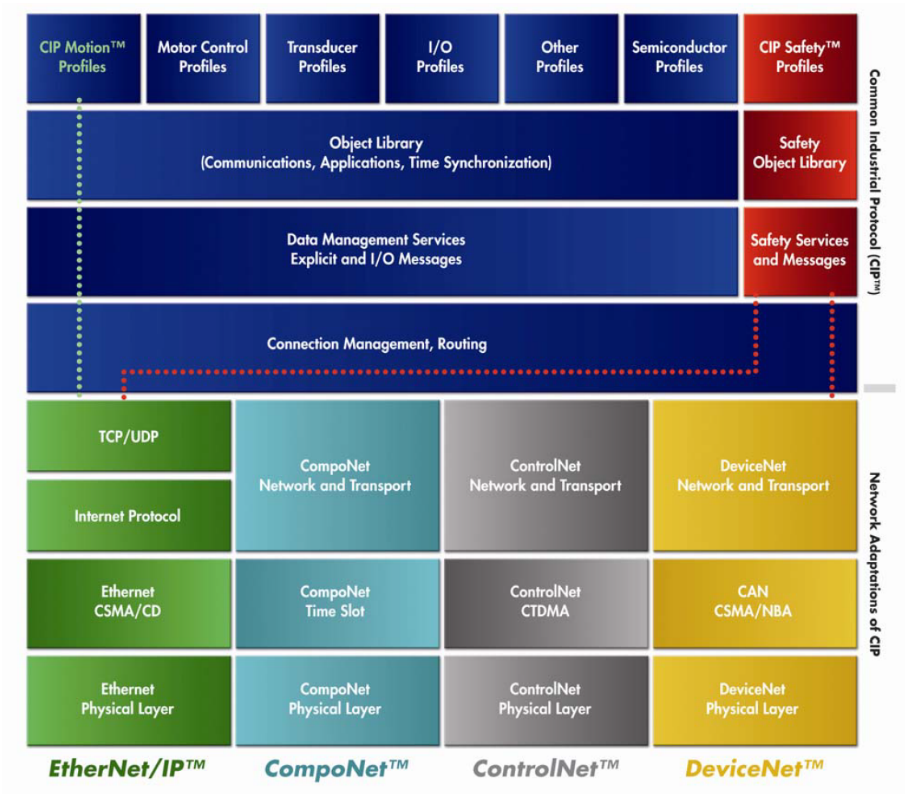

CIP Motion™ Profiles
Motor Control Profiles
Transducer Profiles
I/O Profiles
Other Profiles
Semiconductor Profiles
CIP Safety™ Profiles
Object Library (Communications, Applications, Time Synchronization)
Safety Object Library
Data Management Services Explicit and I/O Messages
Safety Services and Messages
Connection Management, Routing
Common Industrial Protocol (CIP™)
TCP/UDP
Internet Protocol
CompoNet Network and Transport
ControlNet Network and Transport
DeviceNet Network and Transport
Ethernet CSMA/CD
CompoNet Time Slot
ControlNet CTDMA
CAN CSMA/NBA
Ethernet Physical Layer
CompoNet Physical Layer
ControlNet Physical Layer
DeviceNet Physical Layer
Network Adaptations of CIP
EtherNet/IP™
CompoNet™
ControlNet™
DeviceNet™

Figure 2.5: CIP networks

### 2.2.1 CIP - Common Industrial Protocol

CIP protocol is based on object modelling. Each node behaviour (communication services, external interface and data exchanged) is described by an *Object*. It provides an abstract representation of a particular component within a product. Different Object instances of the same component in a node define a *Class*. They have the same set of attributes but different attribute values. *Attributes* indicate a status information, a description of an externally visible characteristic or feature of an Object. There are *Services* that allow to execute tasks.

In a node there are mandatory Objects, required from the CIP Network Specification, but also vendor-specific Objects can be defined. In this way, each manufacturer can customize its EtherNet/IP devices. There are addresses that indicate specific Objects in a node (*Figure 2.6*):

- *Node Address*: Integer value that identifies a node in a device. For EtherNet/IP is the IP address;
- *Class Identifier (Class ID)*: Integer value that identifies a Class in a device;
- *Instance Identifier (Instance ID)*: Integer value that distinguishes an Object instance in a Class;
- *Attribute Identifier (Attribute ID)*: Integer value that identifies an Attribute in an Object;
- *Service Code*: Integer value that identifies an Object instance function.

An external node that want to access a specific attribute needs to know all these addresses.

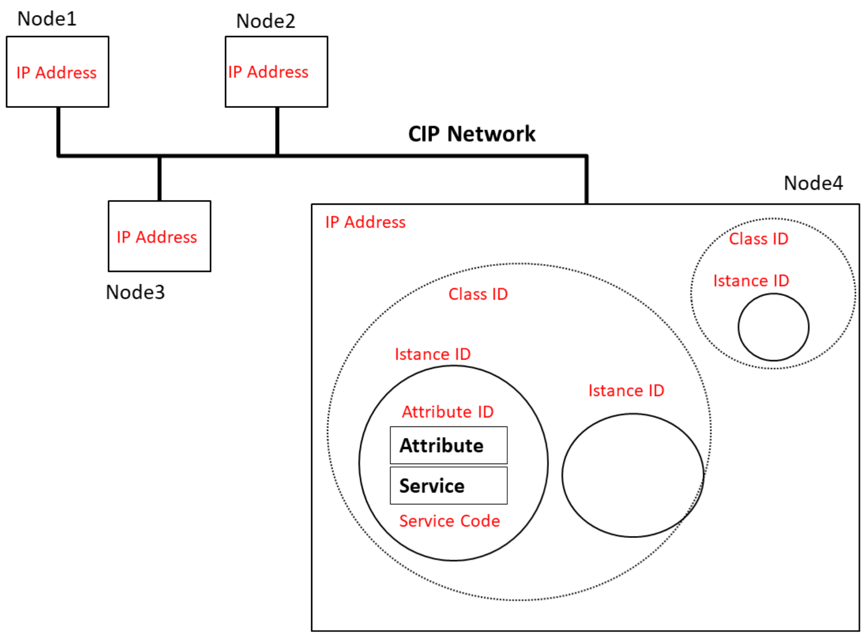


Figure 2.6: CIP Address Structure

The addresses described before can be divided in different categories:

- *Pubblic*: They are common to all CIP users;
- *Vendor-Specific*: Vendors manage this range of addresses for their devices;
- *Reserved for future use*: ODVA reserves some addresses for future use;
- *Object Class Specific*: Some Object classes require specific addresses.

### Messaging protocol

CIP is a connection-based protocol so the access to an internal Object is controlled by the Connection Object. Each connection is associated by a *Connection ID (CID)*. If the transmission is bidirectional, then two CIDs are assigned.

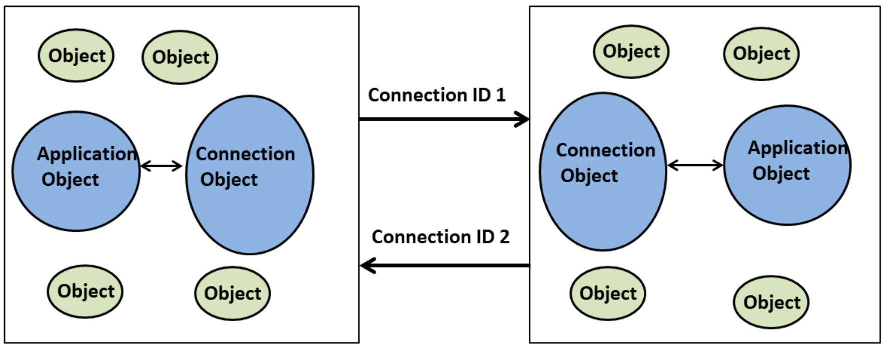


Figure 2.7: CIP Connection

To establish a connection between two devices, the process *Unconnected Message Manager (UCMM)* sends a *UCMM Forward_Open* request message. This service contains all the information required to create a connection between Provider and Consumer, in particular it contains: time-out information, network CID from originator to target, network CID from target to originator, information about the identity of the originator, maximum data size of the message, data segment, routing information and so on. After the setup phase, any message request is automatically routed to the destination device. Connection Objects distinguish two different kind of messages:

- *I/O connections*: It is a dedicated, special-purpose communication path between a producing application and a consuming one. The message is multicast, so it can be sent toward multiple devices. Data exchanged are Application-specific I/O. It is also called *Implicit Connection* because the meaning of the data transmitted is specified in the Connection ID;

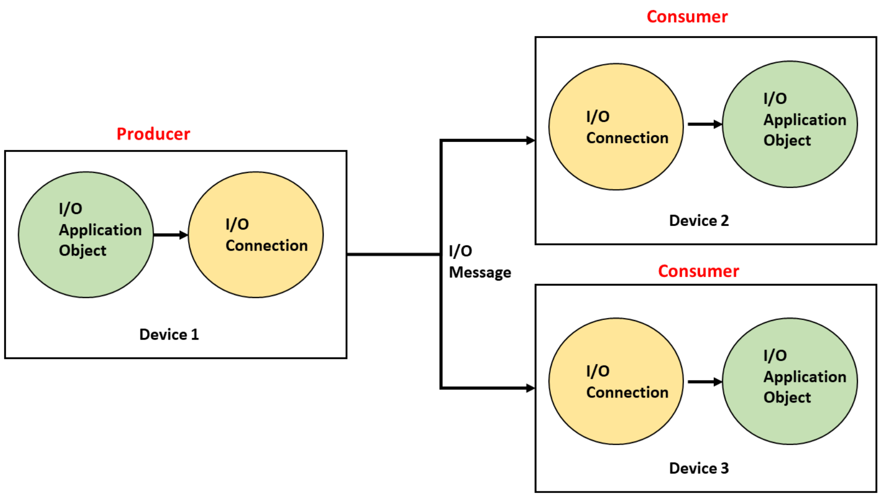


Figure 2.8: CIP I/O Message

- *Explicit message connections*: It is a generic, multi-purpose communication path between two end-points. Explicit Message implements request/response model. It is called "Explicit" because the content of the message explicitly indicates what action should be performed.

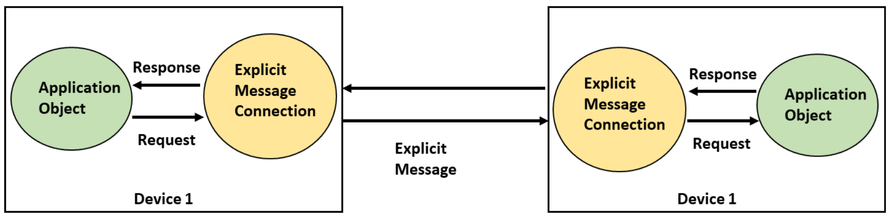


Figure 2.9: CIP Explicit Message

## 2.2.2 CIP Object Model

As said before, CIP is an object-based protocol. Its structure is composed of several connected objects that implement some specific tasks. Different objects are identified with an Object Id. A typical architecture can be seen on *Figure 2.10*:

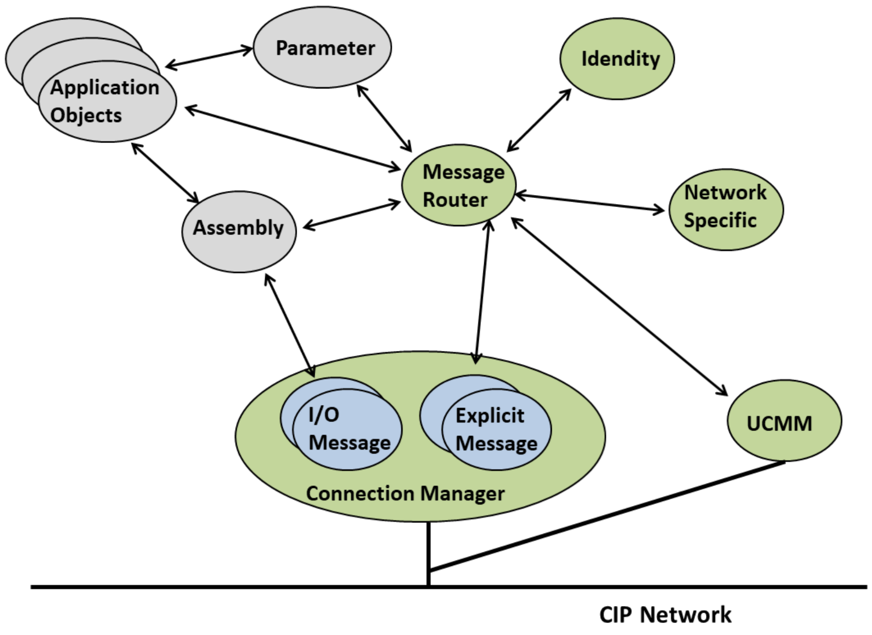


Figure 2.10: CIP Object Model

There are some objects mandatory for all CIP devices: *Message Router, Identity, Connection* and *Network Specific*. Here the most important objects usually present in a CIP device are listed.

- *Identity Object*: Provides an identification and general information about the device. ODVA registers different products of the same manufacturer with the same Vendor ID. Devices of the same vendor are identified with different device type id. Then there are identifiers that indicate the serial number and the revision number;
- *Message Router Object*: Provides a path in which an external device can access, through the Connection object, whatever object instantiated. The Message Router Object, firstly, receives the Explicit Message, then recognizes the Class Instance, routes a service to the specified object and finally routes the response to the correct source. It has a key role and all objects instantiated in a CIP node should be connected to it;
- *Assembly Object*: Allows to map attributes owned by different application object instances, into a single one. This binding is generally used by the I/O messages to maximize the performance of the data exchanged and to reduce the number of connection instances. The policy of mapping input and output data can be specified by the developers, in a fully custom way;
- *Connection Object Manager*: Provides all resources to manage I/O and Explicit Message Connection, for example: open connection, close connection, timeout setting and connection diagnosis;

- *Parameter Object*: There must be an instance of this object for each configurable device. It provides an interface that describes and defines all configuration parameters of a device, such as number of object instances and parameter descriptions.

### 2.2.3 Network adaption of CIP: EtherNetIP

EtherNetIP usually sends I/O Messages, over UDP/IP protocol, using port number **0x08AE** and Explicit Messages, over TCP/IP, using port **0xAF12**. But it is also possible to use UDP/IP for Explicit Messages.
CIP protocol needs to be extended by adding the *TCP/IP Interface Object* and *Ethernet Link Object*.

- TCP/IP Interface Object: Allows to configure a device's TCP/IP network interface. In particular it contains information on the status of interface, IP address, name of the server and network configuration capability supported by the device;
- Ethernet Link Object: Manages configuration parameters, error and status information of the Ethernet IEEE 802.3 communications interface.

The protocol provides a very long Ethernet frame (packet fragmentation is rarely used) and a multi-master structure allows to have no structure limitations. The frame specifies the commands used in the communication and the status information that identify possible errors. Available commands are:

- *NOP*: Used to check if the TCP connection is still open. No response should be provided and receiver should ignore the data content;
- *ListServices*: Determines which services the target device supports. The data field of the reply indicates the services supported;
- *ListIdentity*: The originator checks via UDP some potential target devices. The reply contains the target device code;
- *ListInterfaces*: Looks for non-CIP communication interfaces associated to a target;
- *RegisterSession*: The originator wants to initiate a session. The target device replies that the session is registered;
- *UnRegisterSession*: Originator or target device wants to terminate the session. No reply is admitted for this request;
- *SendRRData*: Used to send explicit unconnected messages, between target and originator. This kind of messages allocate generally limited resources, because the connection is not established;
- *SendUnitData*: Used to send explicit connected messages in both direction, from originator to target and vice versa. They require to set up a connection before the exchange of messages. This means allocate resources for all communications established and remain allocated as log as the connection is alive.

The table below illustrates the status field.

| Error Code | Meaning |
|---|---|
| 0x0000 | Success |
| 0x0001 | Invalid encapsulation command |
| 0x0002 | Insufficient memory resources |
| 0x0003 | Incorrect data in the Encapsulation Data field |
| 0x0004 – 0x0063 | Reserved |
| 0x0064 | Invalid Session Handle |
| 0x0065 | The message received has an invalid length |
| 0x0066 – 0x0068 | Reserved |
| 0x0069 | Unsupported protocol revision |
| 0x006A – 0xFFFF | Reserved for future expansion |

Table 2.3: EtherNet/IP Status field

Some kind of messages require an active session established between originator and target device.

1. *Establishing a Session*: The originator opens a TCP/IP connection to the target and sends a RegisterSession command. The target checks if can support the same protocol version of the originator. If the previous condition is satisfied, it assigns a unique Session ID and sends a RegisterSession reply to the originator;

2. *Maintaining a Session*: The session remains active until the originator or target closes the connection, one of them issues the UnRegisterSession command, or the TCP connection is interrupted;

3. *Terminating a Session*: The session is terminated if originator or target loses the TCP connection or one of them sends the UnRegister command.

## 2.3 OPC UA

OPC UA stands for *Open Platform Communication Unified Architecture*, published in 2008 from OPC Foundation. It is a platform independent, Service-Oriented Architecture (SOA) that allows secure and reliable exchange of data in the industrial automation environment. Different kind of systems and devices can communicate by sending *Messages* between Servers and Clients or *NetworkMessages* between Publisher and Subscribers. Here more attention is paid to Client/Server architecture. OPC UA can be employed in all automation pyramid levels, from actuators, sensors and PLCs to MES, ERP, HMI and SCADA devices. Furthermore it can be employed in industrial domains such as Industrial Internet of Things, Machine To Machine (M2M) and Industry 4.0.
Here a general description of the protocol is presented, for a more detailed explanation refer to OPC Fundation and its specification files.

### 2.3.1 System Description

To meet portability and efficiency requirements, OPC UA can be mapped into several communication protocols, such as *OPC UA TCP, HTTPS* and *WebSockets*, and data can be encoded in three different format: *XML, JSON* and *UA Binary*[9].
Security is implemented through different activities:

- Authentication of Clients, Servers and users: Clients and Servers are identify with digital *Certificate*. Users are authenticated only once, when the session is established;
- Session establishment to guarantee integrity of the Messages;
- Auditing, Encryption and signatures algorithms to guarantee security transport level.

OPC UA is an object-oriented protocol, the system includes the *AddressSpace*, where Servers make available to Clients a set of *Objects* and information such as Data, Events and Alarms. The AddressSpace is accessible only via the set of *Services* provided by the Servers. This field is organized internally in a set of *Nodes* that represent Objects. Each node is described in terms of *Attributes*, *Events* and *Methods*.

#### Client architecture

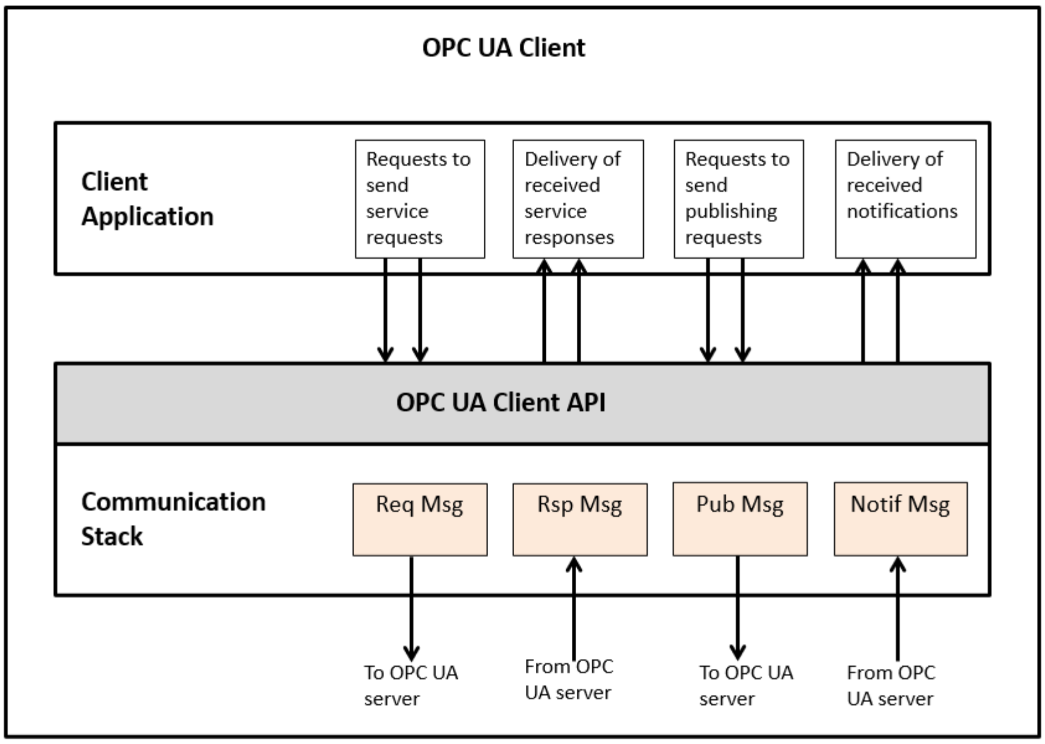


Figure 2.11: OPC_UA Client architecture

The Client architecture (*Figure 2.11*) is composed of several components:

- *Client-Application*: Implements the functions of the Client;
- *Client API*: Isolates the Client-Application from an OPC UA Communication stack. It sends service request/response to the Server;
- *Communication stack*: Converts Client API into Messages and sends them through the communication network. It also receives Messages response and Notification-Messages and delivers them to Client API.

**Server architecture**

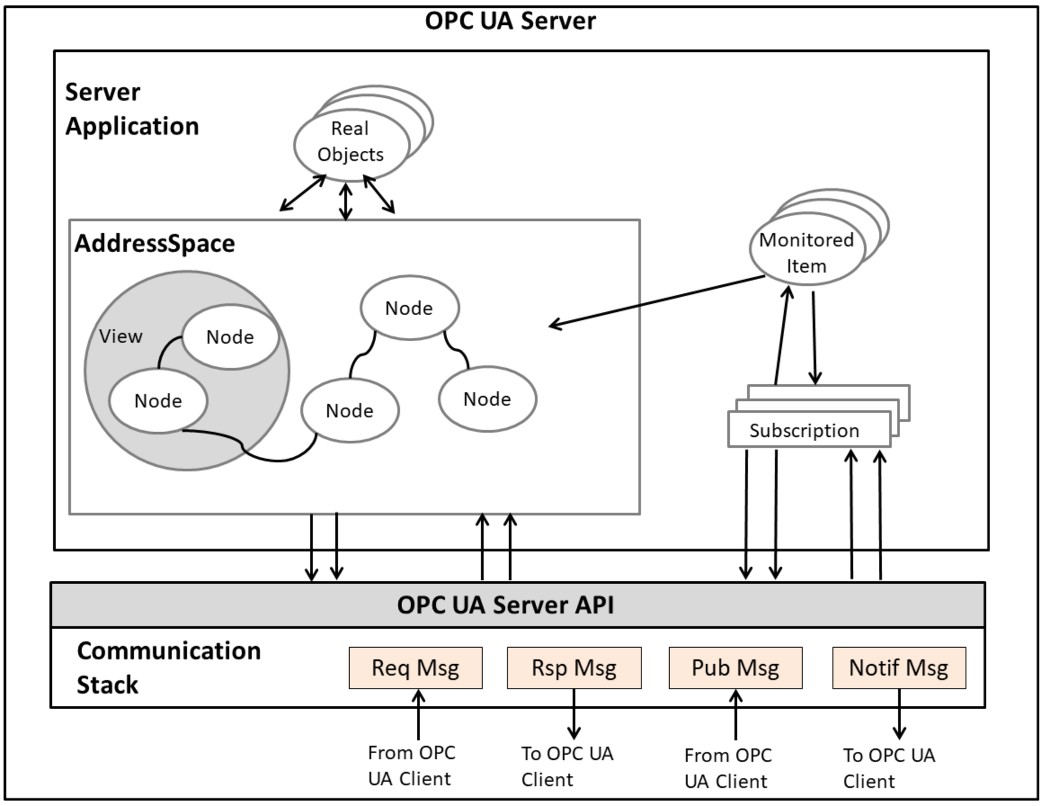


Figure 2.12: OPC_UA Server architecture

The Server architecture (*Figure 2.12*) is composed of several components:

- *Server-Application*: Implements the functions of the Server;
- *Real Objects*: Physical or software objects accessible from the Server-Application;
- *OPC UA AddressSpace*: Composed of several Nodes accessible by Clients using OPC UA Services (interfaces and methods). Nodes represent Objects, their definitions

and their connection to others. Servers are free to organize Nodes in a full custom way. A *View* is a subset of the AddressSpace. All Nodes inside a View are visible from the Client, so the remaining are hidden. For a Client request, the size of the AddressSpace is restricted to only visible Nodes. The default View is the entire AddressSpace. Servers may optionally define other Views;

- *Monitor Item*: Monitors the AddressSpace Nodes and when detects a data change or an event/alarm occurrence, it generates a Notification that is transferred to the Client by a Subscription;
- *Subscription*: It is an endpoint in the Server that publishes Notifications to Clients by sending *Publish Messages*;
- *Server API*: Isolates the Server-Application from an OPC UA communication stack. It sends and receives OPC UA Messages from Clients;
- *Communication stack*: Converts Server API into Messages and sends them through the communication network.

### 2.3.2 Security Model

OPC UA protocol is used at different industry levels, from management fields to the control of hardware devices (sensors, actuators, motors, etc.)[10]. The integration of OPC UA with ERP and MES systems, exposes the protocol to customers and suppliers. It may be an attractive target for industrial espionage or sabotage and causes financial losses. For these reasons, security in the communication protocols is a must for the industries that implement automation. OPC UA provides a set of goals that guarantee security in industrial automation systems:

- *Authentication*: Servers, Clients and users should provide their identity;
- *Authorization*: Read and write operations are allowed only for authorized entities;
- *Confidentiality*: Data is protected from external passive attacks by means of data encryption algorithms;
- *Integrity*: The same information reaches sender and receiver, without the data being changed during transmission;
- *Non-repudiation*: Guarantees that can not be rejected something marked as valid or true;
- *Auditability*: System records operations in order to track successful actions and user activities. Clients and Servers generate audit records of successful and unsuccessful connection attempts, results of security option negotiations, configuration changes, system changes, user interactions and Session rejections;
- *Availability*: It guarantees that the system works normally and no services have been damaged in such a way to become unavailable or severely degraded.

### Security architecture

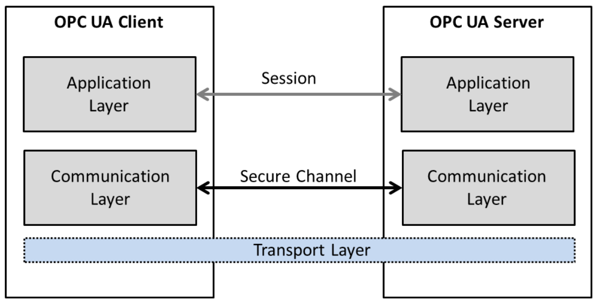


Figure 2.13: OPC_UA Security Architecture

Security architecture allows to implement security features in the OPC UA Client/Server Architectures and it is structured in an *Application Layer* and a *Communication Layer*, over the *Transport Layer*. A Session over a *Secure Channel* needs to be established in order to exchange informations, settings and commands between Client and Server. Session is implemented in the Application Layer, while Secure Channel in the Communication Layer. Data are generated in the Application layer and are passed to the Transport Layer for the transmission. During a Secure Channel establishment a Certificate is exchanged between the OPC UA Application Instances. During the communication, the receiver checks this Certificate and the result of this check determines if the request should be accepted or rejected.

### 2.3.3 AddressSpace Model

AddressSpace allows Servers to provide Objects to Clients[11].

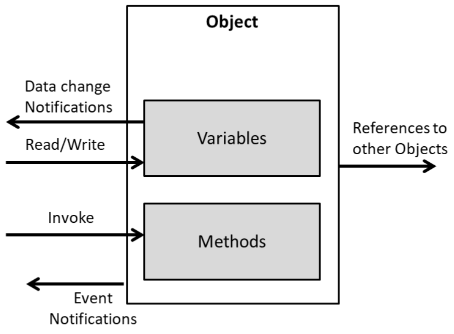


Figure 2.14: OPC_UA Object Model

Objects are defined in term of *Variables* and *Methods*. Variables represent Server-defined data and define the content of an Object, while Methods are limited functions

invoked through the Call service. Client calls Methods, they are executed on the Server and the result of the function is returned to the Client. During the browsing Services, Clients discover all Methods supported by Servers.
Different object elements are grouped in a *NodeClass*. Instances of NodeClass are called *Nodes*. Nodes are defined in term of *Attributes* and they are connected through *References*. Attributes are the data elements that describe Nodes. Clients can access Attribute values using *Read, Write, Query,* and *Subscription/MonitoredItem* Services. Some Nodes are used to organise the AddressSpace and others are used to represent real Objects.

### 2.3.4 Services

Client sends a Service Message request. Server decodes the request and detects all possible communication errors. If no errors arises, it accesses the operation identifier and performs the requested operation. Server includes in the response Message a success/failure code and any data requested. Services provided by Servers are grouped in Service Sets[12].

- *Discovery Service Set*: Clients are allowed to discover all endpoints implemented by Servers and read all the security configuration;
- *SecureChannel Service Set*: Clients are allowed to open a communication channel and ensure the confidentiality and integrity of all messages exchanged with the Server. Clients know the exact algorithms used to create a SecureChannel because they are described in the SecurityPolicy field of a given endpoint;
- *Session Service Set*: These services manage the establishment of a Session and allow the authentication of users;
- *NodeManagement Service Set*: Clients are allowed to add, modify and delete AddressSpace Nodes and References;
- *View Service Set*: Clients can navigate and manage the AddressSpace or the View;
- *Query Service Set*: Clients are allowed to directly obtain data from the AddressSpace or the View without any knowledge of the data organization;
- *Attribute Service Set*: Clients are allowed to read and write attributes of Nodes. They can also read and update historical data or events;
- *Method Service Set*: Manages the Methods call;
- *MonitoredItem Service Set*: MonitoredItems are created to monitor Attributes, Variables and Events. They generate a Notification when detect a changes;
- *Subscription Service Set*: Subscription are used to report Notification to the Client by sending NotificationMessages.

### 2.3.5 Mapping

All sections described before are technology independent. The goal of this section is to map the protocol specifications to the network technology[13]. This is done through the implementation of an OPC UA Stack (*Figure 2.15*).

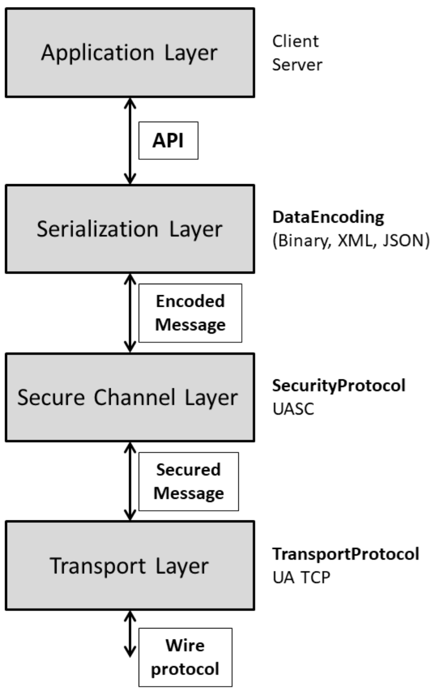


Figure 2.15: OPC_UA Stack

Mapping is implemented through three protocols: *DataEncoding*, *SecurityProtocol* and *TransportProtocol*.

#### DataEncoding

OPC UA defines three different standard for the data encoding. These standard define how Messages frames are constructed.

- *OPC UA Binary*: Designed for fast encoding and decoding, it allows to reach high speed performance;
- *OPC UA XML*: Data are encoded in XML format, following the XML Scheme;
- *OPC UA JSON*: Used to integrate OPC UA with web and enterprise software. The JSON format is described in RFC7159.

### SecurityProtocol

SecurityProtocols implement Services to establish a SecureChannel and apply security to Messages exchanged. Specifically they define the encryption algorithm and the Public Key (stored in Certificates) used to sent secure Messages over the SecureChannel.

### TransportProtocol

A full-duplex channel is established between a Client and a Server, through a TCP/IP or WebSockets. During the connection establishment, four different Messages types can be distinguished:

- *Hello Message*: Indicates that an endpoint wants to be connected to the server;
- *Acknowledge Message*: Server sends this signal to accept the connection request;
- *Error Message*;
- *ReverseHello Message*: The Server says to the Client to establish a SecureChannel using the socket created by the Server.

*Figure* 2.16 shows how a connection is established by a Client. If a Server wants to open the connection, it sends a ReverseHello Message after the OpenConnection. For closing a connection, Client sends a ClosesecureChannel Request and the Server replies with a CloseSocket.

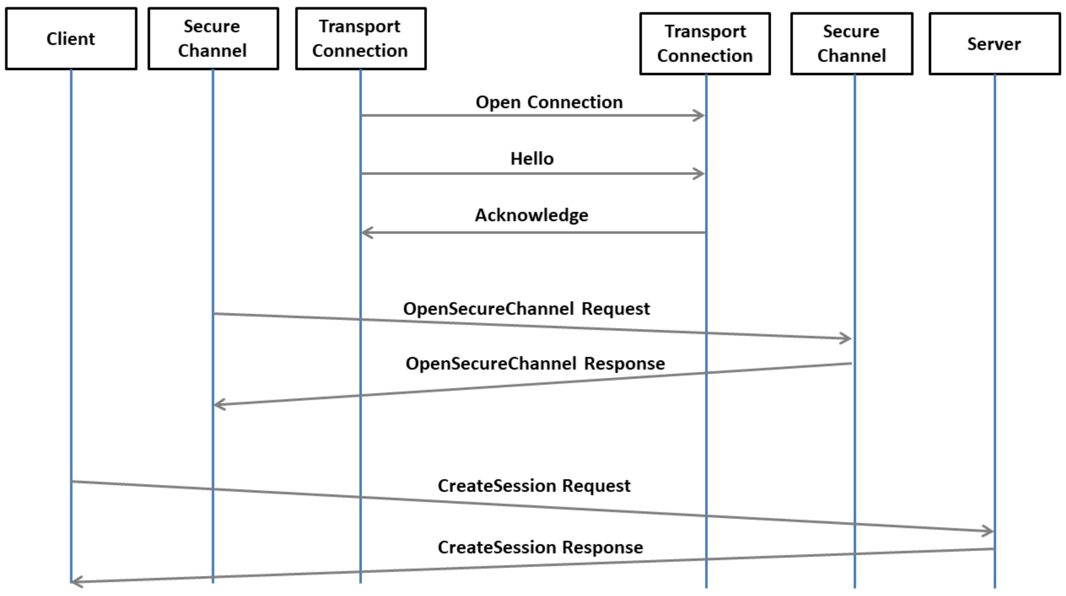


Figure 2.16: OPC_UA Connection

# Chapter 3

# Design choices's motivation

This chapter tries to compare the protocols previously presented and indicates the motivations of all design choices that make it possible to identify a winner of the competition. The first thing to say is that all protocols previously considered can be adopted for this system, they work well under the environment conditions. But there are differences that allow to make a choice. So considering the environment conditions and network requirements, the best communication protocol shall be identified. In this way all production machinery can exchange information with a central system and integration with MES systems is allowed. The metrics considered are listed below in order of importance:

1. *Interoperability*: Ability of a system to be easily interconnected with other devices without any restrictions. To reach high level of interoperability, it is required that devices speak a common protocol language. For this reason it is important that a varieties of devices can support the same protocol;

2. *Security*: Security policies, such as data encryption, protect system resources against external unauthorized access. They allow a secure exchange of data between devices;

3. *Vertical integration*: Indicates a protocol that can be integrated with the other levels of automation pyramid. In this application it is required a system that can be interfaced with field level devices and MES systems;

4. *Frame size*: Indicates the maximum length of the messages. Protocols implement encapsulation: in the application layer they create the frame with the data that need to be sent and then fields are added to adapt the frame to the transmission networks adopted. Protocols provide different frame size, an high value allows to send more data simultaneously;

5. *Communication model*: Indicates the relationships among connected devices in a network;

6. *Services provided*: Each communication protocol provides a set of custom services.

Delay and Jitter constraints are out of this list. The first indicates the time required for the exchange of messages between devices. The second is a variation of the delay and

the biggest absolute value is a measure of the average delay approximation. They are not considered as important metrics because a real time communication is not required, so delay requirements can be relaxed.
The central system does not need to continuously read the input parameters because the electric quantities and production progresses change slowly in time. So scanning the input parameters one or two times every seconds is enough. Similarly, the setting parameters of machinery and properties of processed material are initialized at the beginning and rarely they are modified during the entire production process. For these reasons there are not stringent delay requirements.

## 3.1 Interoperability

Interoperability is defined as the ability of two or more networks, systems or devices to exchange informations[14]. It can be obtained if they both speak a common language or there are standard interfaces that allow the communication between two different protocols. Interoperability is the most important metric, because it is essential connects the communication system with other devices from other vendors. It is also important to individuate a widely diffused protocol because, a priori, it is not known on which devices the system will be interfaced. So having a system that satisfies this requirement is a way to be a better competitor in the market, because there is a higher probability that the system will be integrated with customer existent devices.

- Modbus TCP: It is an open source protocol, freely available for users. The specifications for the protocol implementation are present in Modbus. It is widely diffused in automation industry;
- EtherNet/IP: It is a vendor-specific protocols. ODVA provides a list of all members and devices that support the protocol. In that list there are more than 300 companies, so it is quite diffused in industrial field, but interoperability is not always guaranteed. This because it offers a wide field devices integration but fewer with HMI and Server devices;
- OPC UA: It is a platform independent architecture with a huge list of hardware devices and operating systems that support the protocol. OPC UA provides the necessary infrastructure for interoperability from machine-to-machine, machine-to-enterprise and everything in between.

Interoperability in Modbus TCP and OPC UA protocols is better satisfied because the former is an open source protocol and the latter is cross-platform.

## 3.2 Security

Security requirements are becoming more and more important nowadays. Especially in automation fields, where the system is interfaced with management tools so it is exposed to external customers and suppliers. Protocols provide different policies to exchange secure data.

### Modbus

This protocol does not adopt any security policies. It detects possible communication errors and data corruption in the CRC (Cyclic Redundancy Code) field, but no protection of data is guaranteed.

### EtherNet/IP

CIP provides services to guarantee high integrity levels in the exchange of data. It is a network independent approach, in which safety functionalities are implemented directly into each device. So both standard and safety devices can operate in the same network. Transmission integrity is provided by:

- *Time Expectation*: Time stamp indications allow to detect transmission, data access and routing delays;
- *Production identifier*: Guarantees that messages reach the correct destination. It is obtained from an electronic key, the serial number of the device and the CIP connection serial number;
- *CRC*: Detects possible corruption in transmitted data;
- *Safety connection establishment*: It is established between two end points in which devices configuration, data configuration and connection type are well known;
- *Safety network number*: Provides a unique identifier for each network in the system;
- *Password protection*: Devices use password to protect their configurations;
- *Configuration Ownership*: Indicates who is allowed to configure a device.

### OPC UA

Here huge attention is paid to security. In particular the protocol implements:

- Authentication of Clients and Servers through the exchange of digital Certificates during the connection establishment;
- Encryption algorithms to protect data during the transmission;
- Auditability to record all actions and user activities.

### Summary

Security on EtherNet/IP and OPC UA has a key roles. Both protocols guarantee a secure transmission of data. On the other hand Modbus does not adopt any security policies, it only manages exceptions that arise during communications.
Until now not many customers require the employment of security protocols because textile industries have not yet reached an high-tech level, so this requirements is not decisive in the final choice. But considering a future prospective in which the concept

of automation pyramid will be integrated in textile industries, it is advisable to select a more secure protocol.

## 3.3 Vertical integration

An important requirement is the ability to cover different levels in the automation pyramid and integrate different communication protocols. The table below indicates which protocols can be integrated with the presented ones. Integration is done through gateways. As can be seen, OPC UA can be integrated with a large number of protocols.

| Protocol name | Protocol integration | Note |
|---|---|---|
| Modbus TCP | Modbus family protocols | Integration with protocols of its family |
| EtherNet/IP | Modbus TCP, CompoNet, DeviceNet, ControlNet | Integration with protocols of its family and Modbus TCP |
| OPC UA | Profinet, Modbus TCP, EtherNet/IP, etc. | Wide list of protocols |

Table 3.1: Protocols' integration

Vertical integration considers the data exchange between all automation levels inside a plant. In particular, it denotes the interconnection between production lines and offices, a linkage between field level devices and management applications. OPC UA provides Web Services and XML data encoding, so it allows a complete vertical integration from PLCs, sensors and actuators to HMI, ERP and MES systems. Modbus and EtherNet/IP do not provide any of these services. Consequently OPC UA plays a key role in modern industrial communication systems.

## 3.4 Frame Size

Here the protocols frame sizes are presented.

#### Modbus TCP

The Modbus frame, the *Figure 3.1* shows a generic one, is composed of two section. The former is called *Protocol Data Unit (PDU)* and contains the basic information exchanged (function code + data field). The total length is 253 Bytes. The latter, called *Application Data Unit (ADU)*, adds specific information fields and allows to map the PDU frame on the network used to implement the Modbus protocol.

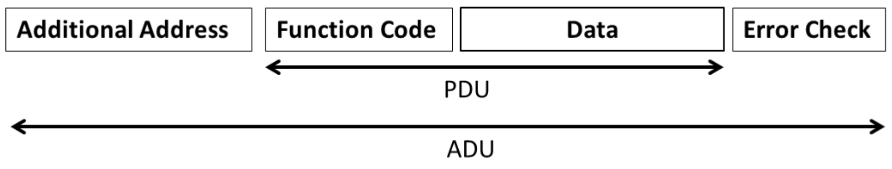


Figure 3.1: Modbus frame

The *Figure 3.2* shows how to encapsulate the Modbus request or response on a TCP/IP network. The ADU frame includes the PDU plus a specific header, the *MBAP header (Modbus Application Protocol header)*. Here are illustrated all the MBAP fields:

- *Transaction identifier*: 2 Bytes length. Identification of a Request/Response transaction. Initialized from the Client during the request transaction, the Server copies it in the response;
- *Protocol identifier*: 2 Bytes length. Identifies the protocol adopted, 0 for Modbus. Initialized from the Client during the request transaction, the Server copies it in the response;
- *Length*: 2 Bytes length. Counts the number of bytes contained in the following fields, including Unit Identifier and Data field. Client and Server fill this slot with the actual indication of the request and response frame;
- *Unit identifier*: 1 Byte length. Identifies the remote slave connected. The Client sets this field in the request and it is recopied by the Server in the response.

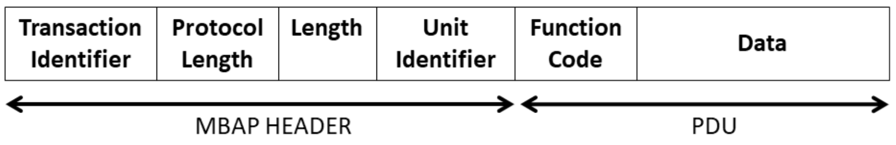


Figure 3.2: ModbusTCP/IP frame

Considering the length of the PDU, the total TCP frame's length:

$$ModbusTCPADU = PDU + MBAP = 253 + 7 = 260 bytes$$

**EtherNet/IP**

CIP messages, sent with the EtherNetIP protocol, need to be encapsulated in a Ethernet frame.

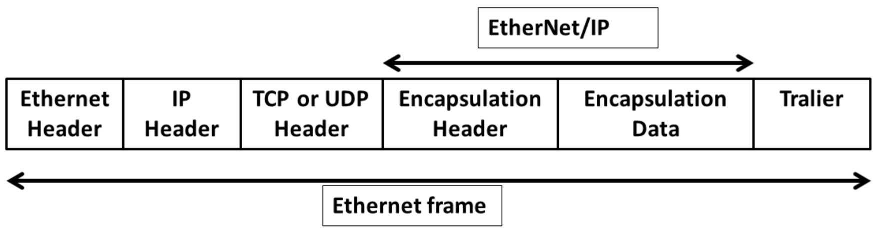


Figure 3.3: EtherNet/IP frame

The *Ethernet Header, IP Header, TCP or UDP Header and Trailer* are specific for the Ethernet frame. The *Encapsulation Header* is formed by several fields with a total length of 24 Bytes, followed by an optional data field. The *Encapsulation data*'s maximum length is 65511 Bytes, so the total EtherNet/IP maximum frame length is **65535 Bytes**. Here the description of all fields:

- *Encapsulation Header* contains:
  - *Command*: 2 Bytes. A session establishes a TCP/IP connection and allows to send commands;
  - *Length*: 2 Bytes. Indicates the length of the Encapsulation Data field;
  - *Session Handle*: 4 Bytes. It is created by the target device after the RegisterSession request. This code is inserted in the following packets exchanged, between target and originator devices, belonging to that session;
  - *Status*: 4 Bytes. Indicates if the receiver was able to execute the requested command. In all requests issued by the sender, the Status field shall contain zero. If the receiver receives a request with a non-zero Status field, the request shall be ignored and no reply shall be generated;
  - *Sender Context*: 8 Bytes. The sender places any value in this field. The receiver should return the same value. It is used for matching request and response messages;
  - *Options*: 4 Bytes. The originator sets it to zero. The target discards this packet if this field is not-zero. This field allows to modify the encapsulation commands.
- *Encapsulation Data* contains the *Command-Specific Data* field. The content and its length depend on the command field. Maximun length is 65511 Bytes.

## OPC UA

Messages are break into several packets, called *MessageChunks*, with a maximum length of **8 192 Bytes**. Thanks to this subdivision, security policies are applied to each individual MessageChunk and not to the entire Message.

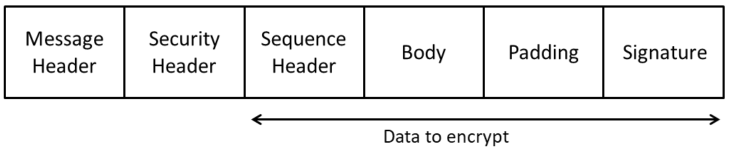


Figure 3.4: OPC_UA Message Chunks

MessageChunk is composed of:

- *Message Header*: Specifies the Message types, if it is the final chunks and the length in bytes of the MessageChunk;
- *Security Header*: Indicates the cryptography operations applied to the Message;
- *Sequence Header*: Ensures that the first encrypted block of every Message sent over a channel will start with different data;
- *Body*: Encoded with the OPC UA Binary encoding. This field is divided in multiple MessageChunks;
- *Padding*: Inserted at the end of Message, it ensures that the data length to encrypt is an integer multiple of the encryption block size;
- *Signature.*

**Summary**

The table below resumes the frame sizes of protocols:

| Protocol name | Frame size |
|---|---|
| Modbus TCP | 260 bytes |
| EtherNet/IP | 65 535 bytes |
| OPC UA | 8 192 bytes |

Table 3.2: Protocols' frame size

EtherNet/IP's frame size is the biggest, fragmentation of data rarely is required. Modbus has a shorter frame size, so requires more frame transmissions to exchanged the desired information. The goal is to reduce the quantity of frames transmitted in order to avoid large communication delays, so a protocol with an higher frame size is welcomed.

## 3.5 Communication model

Considering the communication model, so how devices exchange informations in a network, protocols adopt different models.

| Protocol name | Communication model |
|---|---|
| Modbus TCP | Client-Server |
| EtherNet/IP | Producer-Consumer |
| OPC UA | Client-Server |

Table 3.3: Protocols' communication model

### Client-Server

It is a *many-to-one* model in which a Server is connected to many Clients. It is based on four type of messages (*Figure 3.5*) and only the Client is allowed to start the transaction.

- *Request*: The Client starts the transaction sending a message on the network;
- *Indication*: The Request message is received by the Server side;
- *Response*: Based on the Request, the Server replies with a Response transaction;
- *Confirmation*: The Response message reaches the Client side.

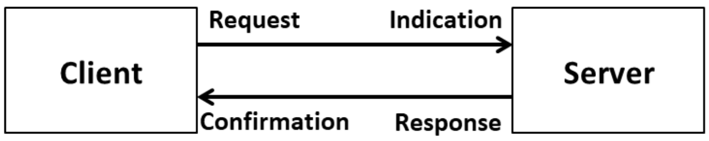


Figure 3.5: Client/Server model

So in this model, the Client asks for a service and the Server performs the task, providing the information required. The Server answers to requests coming from multiple Clients.

### Producer-Consumer

It is a *many-to-many* model in which Producers place messages on the network without having the visibility of the receivers. The messages do not have a direct destination address, but they have an identifier field. All Consumers receive the messages, but only interested devices process the information. So there is a shared bus where messages reach all the devices connected, then only the interested ones are allowed to read the information.

### Summary

The Client-Server model is adopted when information is centralized, because a single node provides all data required in the network. On the other hand, if multiple nodes generate information, the Producer-Consumer model is better. Since this project employs a centralized architecture, the best communication model is the Client-Server. The

Producer-Consumer model can also be adopted, considering a single producer node, but conceptually it is not the best choice.

## 3.6 Services provided

Protocols provide a set of services that simplify the management of connected devices. Here are listed the main services that protocols provide.

- Modbus TCP: Provides only services related to read/write data and diagnostics. It is a simple protocol but it does not provide flexibility and it is difficult to model devices and connect to system software;

- EtherNet/IP: Implements common and object-specific services that provide general tools for managing the communications between devices. Besides, it provides vendor-specific services that guarantee a lot of flexibility, but they may not be understood universally. CIP defines *Devices Profiles*, they are objects that represent groups of devices with similar functionality. This functionality allows to simplify the integration of devices, because services and attributes are already implemented. Developers must use a vendor ID to uniquely identify their product. Moreover, CIP provides *Electronic Datasheet (EDS)* that allow to configure easily a device. They provide a description of parameters, I/O connection and services supported by a device;

- OPC UA: Provides services that allow to search for endpoints and Servers on a network. Then it is possible to establish a secure connection, to read/write attributes and call methods. Machines and equipment are described with data structures and interfaces.

EtherNet/IP provides object-oriented services, so it is easy to integrate field devices. OPC UA does not focus the attention on modelling devices, but it allows to manage easily the elements connected in the networks. Both protocols implement a large set of services, OPC UA is better because it is not related to the hardware adopted. On the other hand, Modbus provides only tools to exchange data and too few services.

## 3.7 Protocols comparison

The table below resumes all the network performance metrics considered before. For Interoperability, Security and Vertical integration fields, *”yes”* means that the protocol pays attention to those arguments and implements solutions to satisfy the requirements. While *”no”* means that the protocol does not provides enough solutions.

| Network performance metrics | Modbus TCP | EtherNet/IP | OPC UA |
|---|---|---|---|
| Interoperability | yes | no | yes |
| Security | no | yes | yes |
| Vertical integration | no | no | yes |
| Frame size | 260 bytes | 65 535 bytes | 8 192 bytes |
| Communication model | Client-Server | Producer-Consumer | Client-Server |
| Services provided | read/write | EDS, Devices Profiles and common services | Server Browsing and Communication handling |

Table 3.4: Protocols' summary table

In particular:

- Modbus TCP:
  - *Interoperability*: "Yes" because it is an open source protocol and it is widely diffused in industries;
  - *Security*: "No" because there aren't services that monitor the data exchanged;
  - *Vertical integration*: "No" because it can be integrated only with protocols of Modbus family.
- EtherNet/IP:
  - *Interoperability*: "No" because it is an hardware dependent protocol and only registered devices can support the protocol;
  - *Security*: "Yes" because the protocol guarantees that messages reach the correct destination and they are not altered;
  - *Vertical integration*: "No" because it can be integrated with protocols of its family and Modbus TCP.
- OPC UA:
  - *Interoperability*: "Yes" because it is a cross-platform protocol;
  - *Security*: "Yes" because there are algorithms that implement data encryption and authentication of Clients and Servers;
  - *Vertical integration*: "Yes" because it can be integrated with Profinet, EtherNet/IP, Modbus and other communication protocols adopted in the automation fields.

Considering the availability of Eaton devices and the arguments previously considered it is possible to derive a final conclusion:

- Modbus TCP is easy to implement and open source, but it does not provide flexibility and security. Eaton provides libraries to easily implement this protocol but modern automation fields require more complex communication systems with more services provided. So it can be employed in simple applications;
- EtherNet/IP is a powerful protocol but it is strictly dependent on the hardware and there are few Eaton devices that support the protocol. So it is hard to implement a system with this protocol, considering that a priori it is not known on which devices the system is going to be interfaced;
- OPC UA provides a lot of services, security is crucial and it guarantees vertical integration and interoperability. Besides Eaton allows to easily implement the protocol.

The winner of the competition is OPC UA. It is the protocol adopted for this project.

# Chapter 4

# Implementation and evaluation

This chapter illustrates how the project is implemented and provides an evaluation of the implemented communication protocol. A typical Officine Gaudino's production line is composed by two or three machines. Previously they were independent of each other, so human effort was required to understand the general state of the production line. In particular, more than one person was needed to organize and monitor the production progresses. Now this figure is no more required because all machines are interconnected and can exchange information with a central system that acts as a Server, while other devices are the Clients of the communication.
*Figure 4.1* shows the system architecture. Machines are sketched with a PLC, connected to an HMI, that manages motors, sensors and actuators.

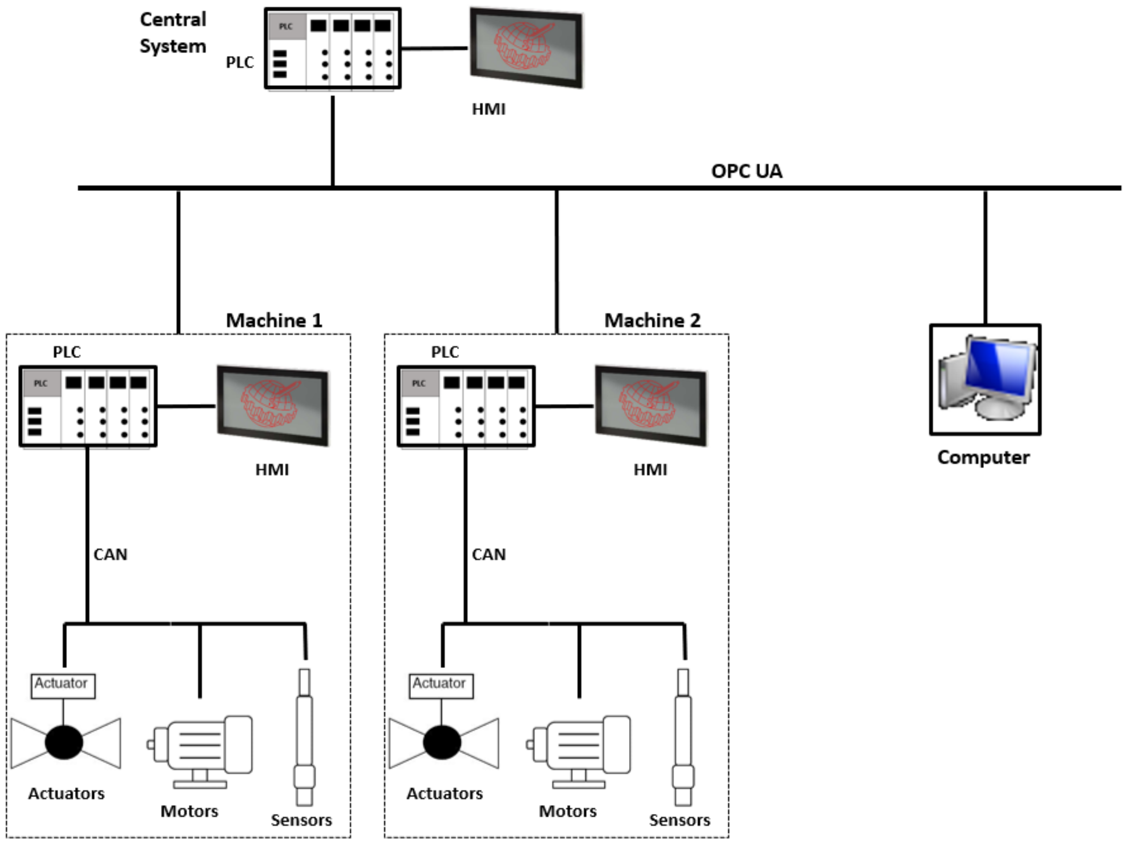


Figure 4.1: Project network

It is a common bus architecture where all devices share the same Ethernet bus, so all of them can access the Server AddressSpace and read or write variables. The central system is composed of a PLC connected to an HMI. It is placed in the office room and it has a complete overview on what is happening in the production line. Clients are PLCs placed on board of machines and they are connected via CAN protocol to motors, sensors and actuators. There is the possibility to connect a computer in the network to monitor the information exchanged and the network performance.
Server is identified by all Clients through a *Server URI* address. It is composed by the prefix *"opc.tcp://"* plus the IP address of the device. This address is used by Clients to establish the connection to the Server device.
The software presented in this chapter allow to exchange variables among Clients and Server through the OPC UA protocol. All exchanged variables can not be defined a priori, because Officine Gaudino provides the possibility to customize their machines, therefore some information exchanged with the central system are defined together with the customers. However a data model, suitable for all machines, should be derived to provides a general structure of the information that need to be exchanged between Client and Server.
Here the software adopted in this project are listed:

- *Codesys 3.5.16* for managing the communication system through PLCs. Codesys[1] is a software platform for industrial automation technology. It allows to program PLCs following the international standards such as the IEC-61131-3;
- *UA Expert Client 1.2* to implement an OPC UA Client on a computer. UA Expert[2] is a full-featured OPC UA Client. It is a cross-platform OPC UA test client programmed in C++ and available for Windows and Linux. It supports the following features:
  - UaExpert Common Framework: the basic framework includes general functionality like certificate handling, discovering OPC UA Servers, connecting with OPC UA Servers, browsing the information model, displaying attributes and references of particular OPC UA Nodes;
  - OPC UA DataAccess View: it allows to select multi-nodes in the Address Space window and drag-and-drop them into the DataAccess View. After that it is possible to monitor or write new values in the node;
  - OPC UA Alarms and Conditions View: it shows detailed information of an individually selected alarm, historical list of events and the current state of pending alarms;
  - OPC UA Historical Trend View: it represents the values of an attribute in a graphical trend view related to the requested time frame;
  - OPC UA Performance View: it measures the performances of the OPC UA Services selected.
- *Wireshark 3.4.9* to monitor the packets exchanged in the network. Wireshark[3] is a widely-used network protocol analyser. It is possible to see what's happening on the network at a microscopic level and it is the de facto standard across many commercial and non-profit enterprises, government agencies and educational institutions. Wireshark development thrives thanks to the volunteer contributions of networking experts around the globe and is the continuation of a project started by Gerald Combs in 1998. It offers a wide number of features such as: deep inspection of hundreds of protocols, live capture and offline analysis, multi-platform, network filters to analyse only a set of packets, live data read from Ethernet and export output to XML, CSV, or plain text.

## 4.1 Data model

This section focuses on the information that the central system should collect. A data model is implemented in order to identify which information should be exchanged. A

[1]https://www.codesys.com/

[2]https://www.unified-automation.com/products/development-tools/uaexpert.html

[3]https://www.wireshark.org/

production line can be composed of different machines, so each of them exchanges specific information. However a general data model is derived in order to organize the information common to all machines. Then, based on the kind of machines or customer requirements, the data model of each Client connected to the central system is modified by adding specific attributes.
In order to create the data structure, variables usually reported on the HMIs of Officine Gaudino machines are examinated. Therefore, a deep analysis of the textile machines developed by the company is required. In particular the user interfaces of ring spinning machines, doubling and twisting machines and cops winder machines are analysed. Then only the information, common to all machines, that allow to have a general overview of the system are considered and reported to the central system. The data model is divided into four arguments: *Alarms, Production control, Working recipe* and *Machine data.* These macro areas contain variables that allow to describe alarm conditions, production progresses, informations on the recipes processed and the status of machines connected in the network. Server fills the working recipe field with the information of the processed material and then reads the other fields. On the other hand, Clients report alarm conditions, production progresses and status information of the machines. The central system replicates this structure for each Clients connected. To avoid bandwidth occupation, only some of the variables inside the data model are exchanged at each program cycle, the others are exchanged only when required or when a change arises. In particular the information regarding the production progresses, the electrical absorptions and the machines status are always analysed.
Here the structure of the data model and the meaning of the variables is presented.

### Alarms

Clients report to Server the failure conditions. When an exception arises, the central system receives information on which machine has stopped and the cause of the error. In particular this field contains:

- *Time information*: it indicates when the failure arises;
- *Module in alarm*: it indicates which device (inverters, sensors, PLCs, etc.) is in the alarm condition;
- *Alarm code*: it allows to understand the cause of the error.

### Production control

Here all information of the production progresses are reported. This field allows the central system to have a complete overview on the production progresses so it is possible to organize the production in order to reduce dead times. Specifically there are:

- *Work shift indication*;
- *Lot*: it indicates the code of the material processed;
- *Bobbin start time*: it reports the start time of the bobbin;

- *Bobbin end time*: it predicts the end time of the bobbin;
- *Working time*: it is the time indication of the machine in running state;
- *Stopping time*: it is the time indication of the machine in stopping or alarm state;
- *Production*: it reports the yarn processed in kilometres or kilograms;
- *Number of bobbins completed*: it indicates the number of completed bobbins from the beginning of the work shift.

#### Working recipe

Recipes are files that contain the settings parameters of yarn processed. In particular they carry information on the characteristics of the material, yarn twist indications, yarn draft indications and speed of motors. In this way, users have no difficulty in processing different materials, they only need to change the recipe and all parameters are already configured. The central system should be able to read and change the recipes that are uploaded on the machines. The signals exchanged are:

- *Kind of material*: it indicates the name of the processed material;
- *Input count*: it is a number which indicates the mass per unit of length or the length per unit of mass of the yarn. It expresses whether the yarn is thick or thin;
- *Yarn twist*: from a technology point of view, the yarn needs to be twisted;
- *Yarn draft*: from a technology point of view, the yarn needs to be drafted;
- *Package shape*: it is possible to define the shape of the final bobbin (cylindrical, bottle, cops, etc.). This field describes in details all dimensions of the final package;
- *Speed of motors*: it influences the quality of the material processed;

#### Machine data

The last field reports information on the status of the system. The signals exchanged are:

- *Kind of machines*: it indicates the type of machine adopted in the system. Today Officine Gaudino develops ring spinning, cops winder, doubling-twisting, twisting-doubling-twisting and drafting machines;
- *Absorptions*: it indicates the electrical absorptions and torques developed of the main motors in the system;
- *User logged*: it reports information on the user logged in;
- *Machine state*: it indicates the working conditions. Here all possible machine states are listed:
  1. Machine stopped;

2. Machine running;
3. Machine stopping;
4. Alarm;
5. Initialization;
6. Machine starting;
7. End of bobbin: it indicates that completed bobbins should be replaced by empty ones in order to move forward the production;
8. Manual commands active: it indicates the possibility to move a singular motor at time, it is used for debug purposes.

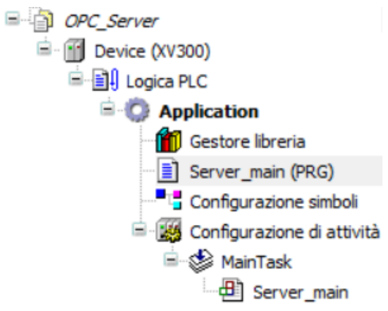
## 4.2 OPC UA Server

Codesys allows to easily implement an OPC UA Server that supports the following features:

- Browsing of data types and variables;
- Standard read/write services;
- Notification for value changes;
- Encrypted communication according to *"OPC UA standard (profile: Basic256SHA256)"*;
- Sending events according to the OPC UA standard.

Firstly, Codesys requires to setup the device, inserting the device type (Eaton XV300) and the IP address. There are some basic steps to perform in order to create an OPC UA Server[15]:

1. Create a new project;
2. Declare some variables of different types in the program;
3. Add a *Symbol Configuration* object.



```
OPC_Server
  Device (XV300)
    Logica PLC
      Application
        Gestore libreria
        Server_main (PRG)
        Configurazione simboli
        Configurazione di attività
          MainTask
            Server_main
```

Figure 4.2: Codesys Server's device tree

4. In the *Add Symbol Configuration* dialogue, select the *Support OPC UA Features* option;

5. Open the symbol configuration in the editor;

6. Click *Build*. The variables are shown in a tree structure;

7. Select the variables, previously defined, that want to be exchanged with an OPC UA Client. Specify the access rights;

8. Download the project to the controller.

Codesys offers the possibility to easily implement OPC UA Servers, because each variable, defined and configured as Symbols, is automatically implemented in the Address Space and visible to all Clients connected. So Clients need to navigate the AddressSpace, accessing all these objects to obtain the value of the variables.
There is also the possibility to create OPC UA Certificates in order to guarantee a secure exchange of information through data encryption and user identification. To do this, it is required to install the *Codesys Security Agent add-on* package and access the *Security screen* dialogue. The steps to follow in order to create a certificate are the following:

1. Install the *Codesys Security Agent add-on*;

2. Access the *Security Screen* through the *View* page;

3. Select the *Devices* tab and then the controller in the left view;

4. In the right view all certificates are displayed. Select the service *OPC UA Server*;

5. Click on create a new certificate and the *Certificate Settings* dialogue opens;

6. Define the certificate parameters, such as the length of the key, the validity period, the username and the password. After that, click *OK* to close the dialogue;

7. Restart the runtime system.

## 4.3 OPC UA Client

A Client device needs to implement an interface able to exchange data with the Server and support OPC UA Services. Here the software for implementing OPC UA Clients on PLCs, with Codesys, and on computer with UaExpert are described. There is this distinction because it is possible to connect a computer on the network and monitor the Server's data exchanged.

### 4.3.1 PLC OPC UA Client

In Codesys there are two ways to implement the OPC UA Client[15]:

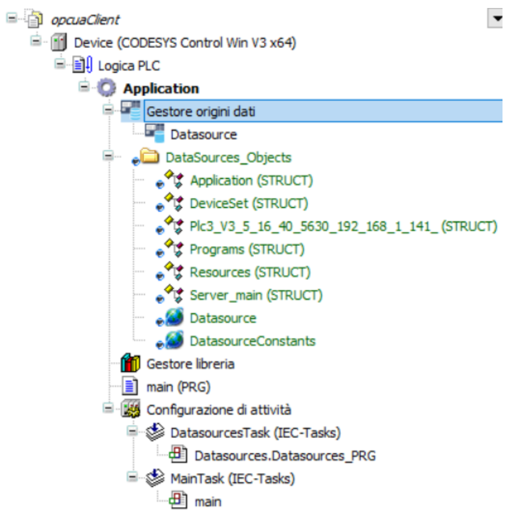
1. Using the *Data Source Manager*;
2. Using the *CmpOPCUAClient* and *CmpOPCUAStack* libraries.

In this project the first solution is used because managing the connection and the data exchange is easier. But an overview of the second version is also given because older Eaton devices support only the OPC UA libraries implementation.

#### Data Source Manager

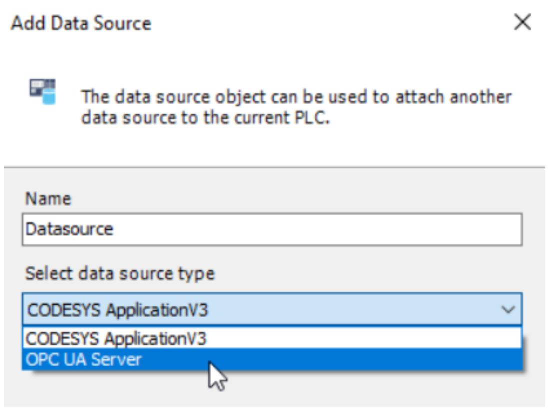
Data source manager, with one or more data source objects, permits to have read/write access to remote devices and their running applications. The functionality of the data source manager allows to establish connections to remote devices and make its data available through data source variables. Depending on the network where the controllers are located, a connection is established via different data source types and different connection types. Codesys requires the installation of the *Communication package*. This package offers extensive support of communication protocols for data exchange with other systems in the automation landscape. In particular, it implements interfaces to easily exchange information through the integrated communication protocols. The basic steps that allow to implement an OPC UA Client are the following:

1. Add *Data Source Manager* and a *Data Source Object* selecting OPC UA Server as data source type;



```
opcuaClient
  Device (CODESYS Control Win V3 x64)
    Logica PLC
      Application
        Gestore origini dati
          Datasource
        DataSources_Objects
          Application (STRUCT)
          DeviceSet (STRUCT)
          Plc3_V3_5_16_40_5630_192_168_1_141_ (STRUCT)
          Programs (STRUCT)
          Resources (STRUCT)
          Server_main (STRUCT)
          Datasource
          DatasourceConstants
        Gestore libreria
        main (PRG)
        Configurazione di attività
          DatasourcesTask (IEC-Tasks)
            Datasources.Datasources_PRG
          MainTask (IEC-Tasks)
            main
```

(a) *Client variables*



Add Data Source

The data source object can be used to attach another data source to the current PLC.

Name: Datasource

Select data source type: CODESYS ApplicationV3

CODESYS ApplicationV3
OPC UA Server

(b) *Client configuration*

Figure 4.3: Codesys Client's Data Source Object

2. Set the *Server URI* address;

3. Choose the nodes and the data that need to be exchanged;

4. Now all variables exchanged are visible, so implement a program that manages those variables. For accessing a variable, it is required to provide the AddressSpace path.

#### Codesys libraries

They provide functions that allow to configure a Client device and create the connection to the Server. Then it is possible to implement services like: Read data, Write data, Discovery Server on network, monitor item, event creation, implement Subscription and managing the View. Lastly, there are functions that disconnect the Client and eliminate the Session. For more details on the Codesys libraries adopted refer to CmpOPCUAClient and CmpOPCUAStack.
Here a description of the program flow that allows to establish a connection and exchange data:

1. *OPCUAClient_Create*: It creates a new OPC UA Client instance and it is the base of every further communication. It requires a Client description, in particular the device name and the hostname. It returns a session code;

2. *OPCUAClient_Connect*: It connects an OPC UA Client to a Server. The created session can be used to send services. It requires, for configuring the connection, the session code, session name, Server URL, timeout value, security policies, security level, security mode and server certificates. It returns a connection code;

3. *OPCUAClient_Read*: This function sends a Read request to the OPC UA Server. It requires the connection code and the description of the attribute required;

4. *OPCUAClient_Write*: This function sends a Write request to the OPC UA Server. It requires the connection code and the description of the attribute required;

5. *OPCUAClient_Disconnect and OPCUAClient_Delete*: They disconnect and delete an OPC UA Client. All resources related to the session and the connection are cleaned up.

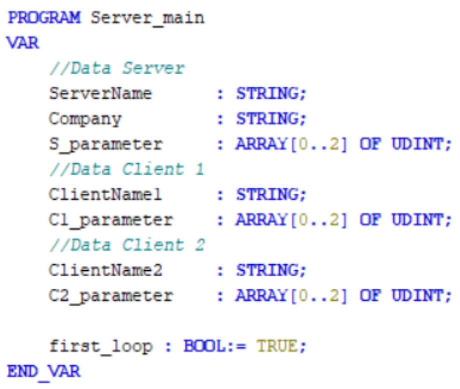
This solution is more complex than the previous one but it is supported by a large number of Eaton devices. After the creation of the Session and the connection to the Server, the Client can manage specific attributes. Programmers need to know the AddressSpace organization for managing the correct attribute. A suggestion is to run the UaExpert Client (see next section), retrieves the attribute description and fills all the fields required by the Codesys functions. In this way, programmers have a complete overview of the Server structure and can manage easily attributes and services.

### 4.3.2 Computer OPC UA Client

There is also the possibility to connect a computer on the network. Here the *UaExpert Client* is used: it is a free software that provides an easy connection to the Codesys OPC UA Server. There are a lot of open-source programs written in Python and C++ that implement the Client, such as Free OPC-UA, but here the UaExpert software is selected because it provides a clear graphic user interface, an easy connection mode and a complete attributes description. Firstly the Server URI address should be provided. After that, the Client scans the network searching the endpoint with that address. Then, if authentication is required, a Certificate or Password and Username must be provided. After this setup phase, the Address Space is completely available. At this point it is possible to read and modify some variables, create and monitor an event. On the left of the UAExpert interface, there is the possibility to see the AddressSpace created by the Server. In the central dialogue, variables can be inserted and their main attributes are shown. On the right dialogue there is a complete overview of all the attributes of a selected variable. There is the possibility to add a *Performance View* and perform some measurements on the delay of reading and writing variables. First it is necessary to select, from the AddressSpace, the Nodes that should be used for testing. Then the performance measurements must be configured, in particular selecting the services employed, the number of nodes and the duration of the measurement. The UaExpert will call the UA Service and measure the duration of each call.

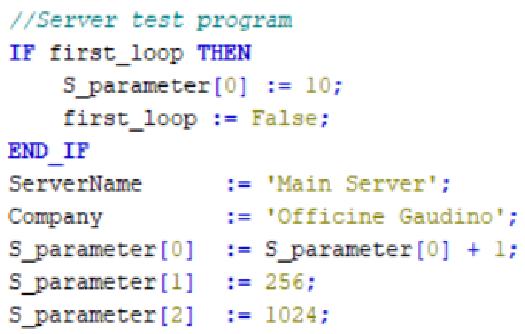
## 4.4 Protocol analysis

This section performs some tests that analyse the software implemented. The architecture of the system is composed by a Client connected to a Server. The Server consists of an Eaton XV300 device connected in the Ethernet network of the company and it is identified with the IP address *192.168.1.141*. A desktop PC simulates the Client and performs the analysis over the communication protocols. It is identified with the IP address *192.168.1.25*.
*Figure 4.4a* shows the test variables defined in the Server, they are inserted with the only scope to illustrate the data exchange. The Server program (*Figure 4.4b*) simply writes the content of some variables and increments one of them at each program cycle. The remaining variables are set by the Client. *Figure 4.4c* illustrates the Address Spaces just implemented.
In the Client program (*Figure 4.5*), simply, the first five rows are variables written by the Server, the others are written by the Client. The complete variables AddressSpace path must be provided in order to access their contents.



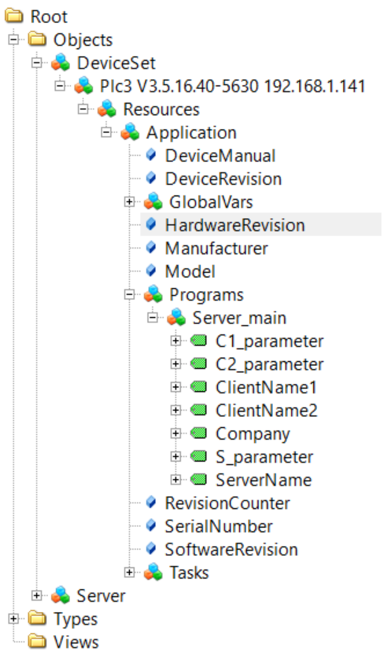
```
PROGRAM Server_main
VAR
    //Data Server
    ServerName      : STRING;
    Company         : STRING;
    S_parameter     : ARRAY[0..2] OF UDINT;
    //Data Client 1
    ClientName1     : STRING;
    C1_parameter    : ARRAY[0..2] OF UDINT;
    //Data Client 2
    ClientName2     : STRING;
    C2_parameter    : ARRAY[0..2] OF UDINT;

    first_loop : BOOL:= TRUE;
END_VAR
```

(a) *Server variables*



```
//Server test program
IF first_loop THEN
    S_parameter[0] := 10;
    first_loop := False;
END_IF
ServerName      := 'Main Server';
Company         := 'Officine Gaudino';
S_parameter[0]  := S_parameter[0] + 1;
S_parameter[1]  := 256;
S_parameter[2]  := 1024;
```

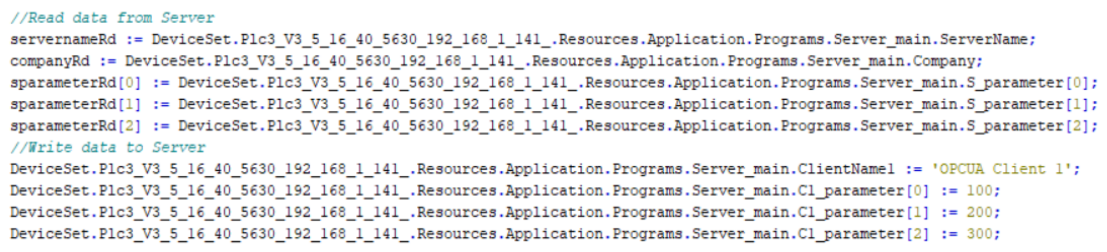
(b) *Server program*



```
Root
  Objects
    DeviceSet
      Plc3 V3.5.16.40-5630 192.168.1.141
        Resources
          Application
            DeviceManual
            DeviceRevision
            GlobalVars
            HardwareRevision
            Manufacturer
            Model
            Programs
              Server_main
                C1_parameter
                C2_parameter
                ClientName1
                ClientName2
                Company
                S_parameter
                ServerName
            RevisionCounter
            SerialNumber
            SoftwareRevision
            Tasks
    Server
  Types
  Views
```

(c) *Codesys Server's Address Space*

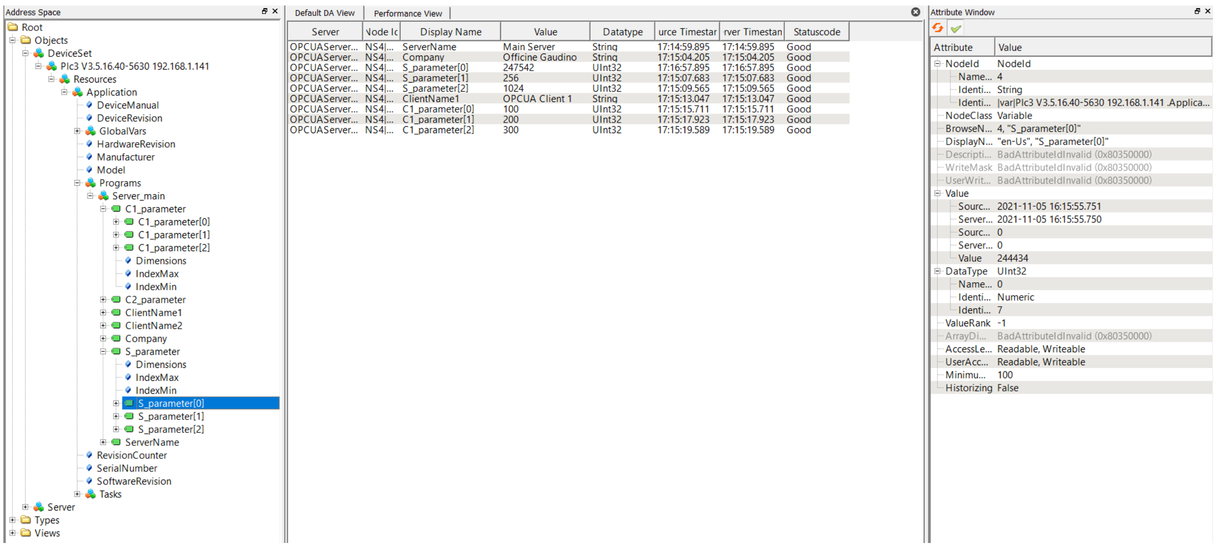
Figure 4.4: Codesys test Server's variables and program



```
//Read data from Server
servernameRd := DeviceSet.Plc3_V3_5_16_40_5630_192_168_1_141_.Resources.Application.Programs.Server_main.ServerName;
companyRd := DeviceSet.Plc3_V3_5_16_40_5630_192_168_1_141_.Resources.Application.Programs.Server_main.Company;
sparameterRd[0] := DeviceSet.Plc3_V3_5_16_40_5630_192_168_1_141_.Resources.Application.Programs.Server_main.S_parameter[0];
sparameterRd[1] := DeviceSet.Plc3_V3_5_16_40_5630_192_168_1_141_.Resources.Application.Programs.Server_main.S_parameter[1];
sparameterRd[2] := DeviceSet.Plc3_V3_5_16_40_5630_192_168_1_141_.Resources.Application.Programs.Server_main.S_parameter[2];
//Write data to Server
DeviceSet.Plc3_V3_5_16_40_5630_192_168_1_141_.Resources.Application.Programs.Server_main.ClientName1 := 'OPCUA Client 1';
DeviceSet.Plc3_V3_5_16_40_5630_192_168_1_141_.Resources.Application.Programs.Server_main.C1_parameter[0] := 100;
DeviceSet.Plc3_V3_5_16_40_5630_192_168_1_141_.Resources.Application.Programs.Server_main.C1_parameter[1] := 200;
DeviceSet.Plc3_V3_5_16_40_5630_192_168_1_141_.Resources.Application.Programs.Server_main.C1_parameter[2] := 300;
```

Figure 4.5: Codesys Client's program

Connecting the UAExpert Client to the Server implemented, it is possible to read the variables exchanged. On the left of *Figure 4.6* the AddressSpace created by the Server is visible. In the central dialogue variables can be inserted and their main attributes are showed. On the right dialogue there is a complete overview of all the attributes of a selected variable.



```
Root
  Objects
    DeviceSet
      Plc3 V3.5.16.40-5630 192.168.1.141
        Resources
          Application
            DeviceManual
            DeviceRevision
            GlobalVars
            HardwareRevision
            Manufacturer
            Model
            Programs
              Server_main
                C1_parameter
                  C1_parameter[0]
                  C1_parameter[1]
                  C1_parameter[2]
                  Dimensions
                  IndexMax
                  IndexMin
                C2_parameter
                ClientName1
                ClientName2
                Company
                S_parameter
                  Dimensions
                  IndexMax
                  IndexMin
                  S_parameter[0]
                  S_parameter[1]
                  S_parameter[2]
                ServerName
            RevisionCounter
            SerialNumber
            SoftwareRevision
            Tasks
    Server
  Types
  Views
```

| Server | Node Ic | Display Name | Value | Datatype | urce Timestar | rver Timestan | Statuscode |
|---|---|---|---|---|---|---|---|
| OPCUAServer... | NS4\|... | ServerName | Main Server | String | 17:14:59.895 | 17:14:59.895 | Good |
| OPCUAServer... | NS4\|... | Company | Officine Gaudino | String | 17:15:04.205 | 17:15:04.205 | Good |
| OPCUAServer... | NS4\|... | S_parameter[0] | 247542 | UInt32 | 17:16:57.895 | 17:16:57.895 | Good |
| OPCUAServer... | NS4\|... | S_parameter[1] | 256 | UInt32 | 17:15:07.683 | 17:15:07.683 | Good |
| OPCUAServer... | NS4\|... | S_parameter[2] | 1024 | UInt32 | 17:15:09.565 | 17:15:09.565 | Good |
| OPCUAServer... | NS4\|... | ClientName1 | OPCUA Client 1 | String | 17:15:13.047 | 17:15:13.047 | Good |
| OPCUAServer... | NS4\|... | C1_parameter[0] | 100 | UInt32 | 17:15:15.711 | 17:15:15.711 | Good |
| OPCUAServer... | NS4\|... | C1_parameter[1] | 200 | UInt32 | 17:15:17.923 | 17:15:17.923 | Good |
| OPCUAServer... | NS4\|... | C1_parameter[2] | 300 | UInt32 | 17:15:19.589 | 17:15:19.589 | Good |

| Attribute | Value |
|---|---|
| NodeId | NodeId |
| Name... | 4 |
| Identi... | String |
| Identi... | \|var\|Plc3 V3.5.16.40-5630 192.168.1.141 .Applica... |
| NodeClass | Variable |
| BrowseN... | 4, "S_parameter[0]" |
| DisplayN... | "en-Us", "S_parameter[0]" |
| Descripti... | BadAttributeIdInvalid (0x80350000) |
| WriteMask | BadAttributeIdInvalid (0x80350000) |
| UserWrit... | BadAttributeIdInvalid (0x80350000) |
| Value | |
| Sourc... | 2021-11-05 16:15:55.751 |
| Server... | 2021-11-05 16:15:55.750 |
| Sourc... | 0 |
| Server... | 0 |
| Value | 244434 |
| DataType | UInt32 |
| Name... | 0 |
| Identi... | Numeric |
| Identi... | 7 |
| ValueRank | -1 |
| ArrayDi... | BadAttributeIdInvalid (0x80350000) |
| AccessLe... | Readable, Writeable |
| UserAcc... | Readable, Writeable |
| Minimu... | 100 |
| Historizing | False |

Figure 4.6: UaExpert OPC UA Client

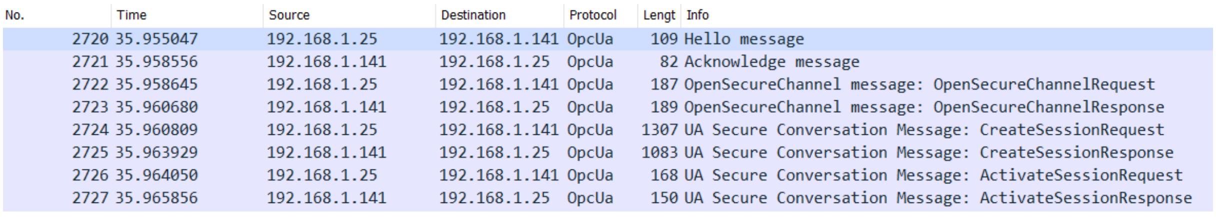
After writing these trivial software, detailed analysis of packets exchanged (with Wireshark) and a performance analysis (with UA Express) are implemented.

## 4.4.1 Packets exchanged analysis

The analysis of the packets received is performed with the Wireshark software. The goal of this test is to verify the correctness of the data exchanged. In particular three situations are analysed: the connection opening, the data exchange and the connection closing. The setup phase of Wireshark consists in identifying the network to be analysed and adding a filter in order to consider only the OPC UA packets. For each packet exchanged, Wireshark indicates the addresses of the source and destination devices, the time indication, the length of the frame and a description of the message transmitted. It is also possible to review the frames received, analysing the meaning of all frame fields. At the end of this section a packet lengths analysis is presented.

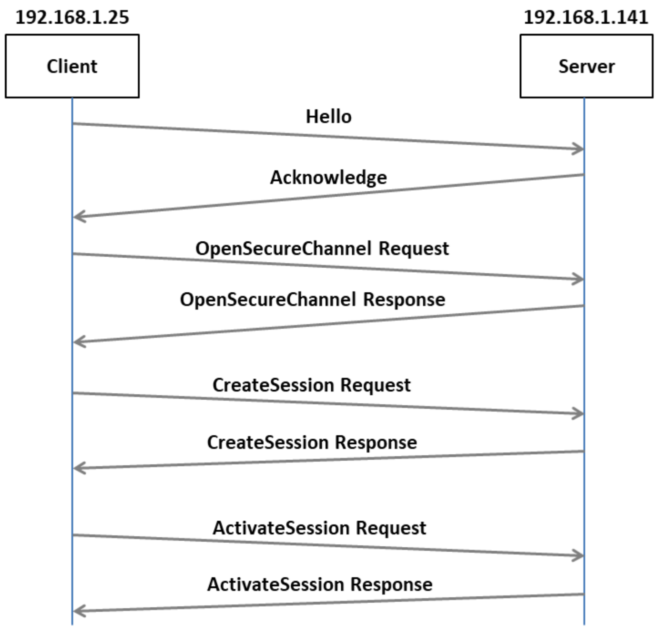
## Connection opening

*Figure 4.7a* illustrates the messages exchanged between Client and Server while establishing the OPC UA connection.



```
No.     Time       Source         Destination    Protocol Lengt Info
   2720 35.955047  192.168.1.25   192.168.1.141  OpcUa     109 Hello message
   2721 35.958556  192.168.1.141  192.168.1.25   OpcUa      82 Acknowledge message
   2722 35.958645  192.168.1.25   192.168.1.141  OpcUa     187 OpenSecureChannel message: OpenSecureChannelRequest
   2723 35.960680  192.168.1.141  192.168.1.25   OpcUa     189 OpenSecureChannel message: OpenSecureChannelResponse
   2724 35.960809  192.168.1.25   192.168.1.141  OpcUa    1307 UA Secure Conversation Message: CreateSessionRequest
   2725 35.963929  192.168.1.141  192.168.1.25   OpcUa    1083 UA Secure Conversation Message: CreateSessionResponse
   2726 35.964050  192.168.1.25   192.168.1.141  OpcUa     168 UA Secure Conversation Message: ActivateSessionRequest
   2727 35.965856  192.168.1.141  192.168.1.25   OpcUa     150 UA Secure Conversation Message: ActivateSessionResponse
```

(a) *Wireshark OPC UA open connection*



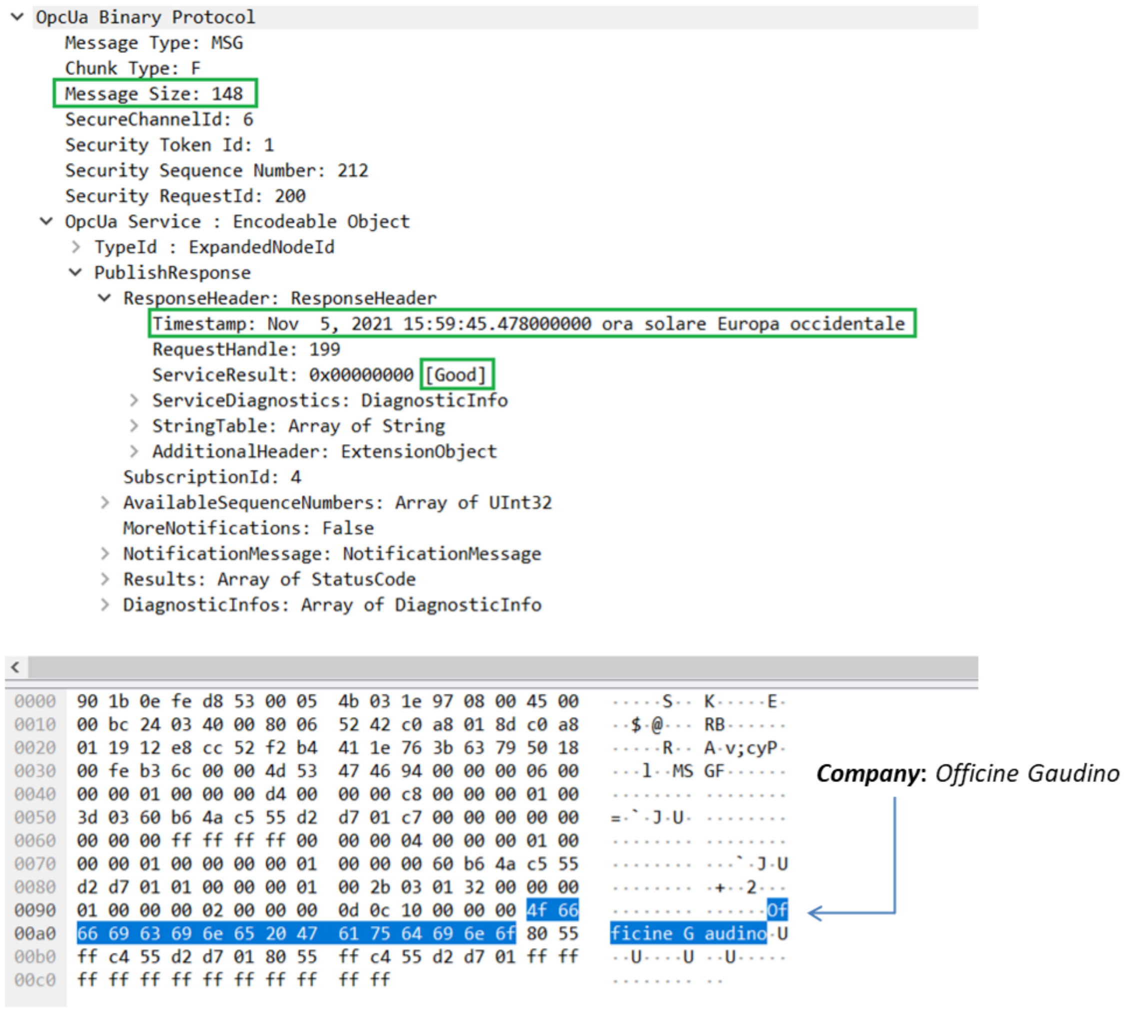
(b) *Open connection diagram*

Figure 4.7: OPC UA protocol analysis - open connection

Firstly the Client sends an *Hello* message to verify if the Server is able to listen to it. The Server, if it is not busy, replies with an *Acknowledge* message. After that, the Client sends an *OpenSecureChannelRequest* and waits for a *OpenSecureChannelResponse*, in this way a secure channel is created. Then the Client sends a *CreateSessionRequest* and the Server replies with a *CreateSessionResponse*. The session just created needs to be activated with *ActivateSessionRequest* and *ActivateSessionResponse* messages. After this setup phase the two devices are able to exchange messages.

### Data exchange

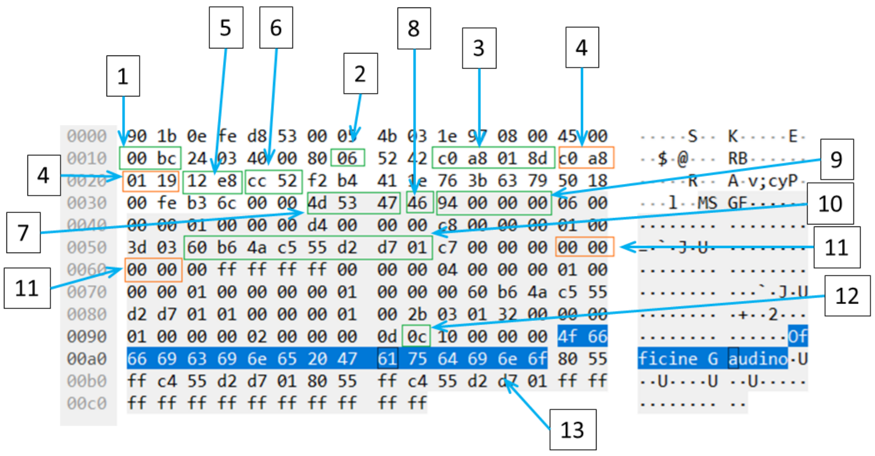
The analysis of the data received is performed over the *Company* variables which the Server fills with the string *"Officine Gaudino"*. At the bottom of the *figure 4.8* the entire frame received by the Client is shown. The content of the variable observed is highlighted. On the top of the figure there is a summary with the most relevant information such as the message size, the timestamp and the status of the packet received.



```
OpcUa Binary Protocol
    Message Type: MSG
    Chunk Type: F
    Message Size: 148
    SecureChannelId: 6
    Security Token Id: 1
    Security Sequence Number: 212
    Security RequestId: 200
  OpcUa Service : Encodeable Object
    > TypeId : ExpandedNodeId
      PublishResponse
        ResponseHeader: ResponseHeader
            Timestamp: Nov  5, 2021 15:59:45.478000000 ora solare Europa occidentale
            RequestHandle: 199
            ServiceResult: 0x00000000 [Good]
          > ServiceDiagnostics: DiagnosticInfo
          > StringTable: Array of String
          > AdditionalHeader: ExtensionObject
        SubscriptionId: 4
      > AvailableSequenceNumbers: Array of UInt32
        MoreNotifications: False
      > NotificationMessage: NotificationMessage
      > Results: Array of StatusCode
      > DiagnosticInfos: Array of DiagnosticInfo
```

```
0000  90 1b 0e fe d8 53 00 05  4b 03 1e 97 08 00 45 00   ·····S·· K·····E·
0010  00 bc 24 03 40 00 80 06  52 42 c0 a8 01 8d c0 a8   ··$·@··· RB······
0020  01 19 12 e8 cc 52 f2 b4  41 1e 76 3b 63 79 50 18   ·····R·· A·v;cyP·
0030  00 fe b3 6c 00 00 4d 53  47 46 94 00 00 00 06 00   ···l··MS GF······
0040  00 00 01 00 00 00 d4 00  00 00 c8 00 00 00 01 00   ········ ········
0050  3d 03 60 b6 4a c5 55 d2  d7 01 c7 00 00 00 00 00   =·`·J·U· ········
0060  00 00 00 ff ff ff ff 00  00 00 04 00 00 00 01 00   ········ ········
0070  00 00 01 00 00 00 00 01  00 00 00 60 b6 4a c5 55   ········ ···`·J·U
0080  d2 d7 01 01 00 00 00 01  00 2b 03 01 32 00 00 00   ········ ·+··2···
0090  01 00 00 00 02 00 00 00  0d 0c 10 00 00 00 4f 66   ········ ······Of
00a0  66 69 63 69 6e 65 20 47  61 75 64 69 6e 6f 80 55   ficine G audino·U
00b0  ff c4 55 d2 d7 01 80 55  ff c4 55 d2 d7 01 ff ff   ··U····U ··U·····
00c0  ff ff ff ff ff ff ff ff  ff ff                     ········ ··
```

Figure 4.8: Wireshark OPC UA data exchange



```
0000  90 1b 0e fe d8 53 00 05  4b 03 1e 97 08 00 45 00   ·····S·· K·····E·
0010  00 bc 24 03 40 00 80 06  52 42 c0 a8 01 8d c0 a8   ··$·@··· RB······
0020  01 19 12 e8 cc 52 f2 b4  41 1e 76 3b 63 79 50 18   ·····R·· A·v;cyP·
0030  00 fe b3 6c 00 00 4d 53  47 46 94 00 00 00 06 00   ···l··MS GF······
0040  00 00 01 00 00 00 d4 00  00 00 c8 00 00 00 01 00   ········ ········
0050  3d 03 60 b6 4a c5 55 d2  d7 01 c7 00 00 00 00 00   =·`·J·U· ········
0060  00 00 00 ff ff ff ff 00  00 00 04 00 00 00 01 00   ········ ········
0070  00 00 01 00 00 00 00 01  00 00 00 60 b6 4a c5 55   ········ ···`·J·U
0080  d2 d7 01 01 00 00 00 01  00 2b 03 01 32 00 00 00   ········ ·+··2···
0090  01 00 00 00 02 00 00 00  0d 0c 10 00 00 00 4f 66   ········ ······Of
00a0  66 69 63 69 6e 65 20 47  61 75 64 69 6e 6f 80 55   ficine G audino·U
00b0  ff c4 55 d2 d7 01 80 55  ff c4 55 d2 d7 01 ff ff   ··U····U ··U·····
00c0  ff ff ff ff ff ff ff ff  ff ff                     ········ ··
```

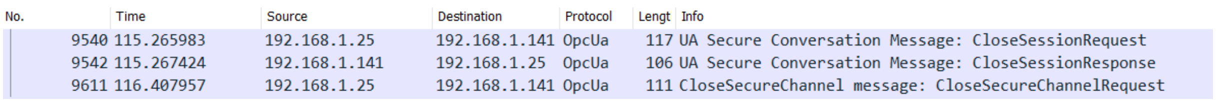
Figure 4.9: Wireshark OPC UA packet fields

*Figure 4.9* shows the Ethernet frame received by the Client after reading the Company variable. The main fields are explained below:

1. Total length: it indicates the total length of the frame, 188 bits (0xbc);
2. Protocol: it indicates the protocol adopted, 0x06 stands for TCP;
3. Source address: 192.168.1.141;
4. Destination address: 192.168.1.25;
5. Source port: 4840;
6. Destination port: 52306;
7. Message type: MSG;
8. Chunk type: "F" stands for final chunks. Since packets fragmentation was not required, this is the only chunk received;
9. Message size: it indicates the length of the message, 148 bits;
10. Timestamp: it provides a time indication;
11. Service result: it indicates the status of the transmission, 0x00 means that no error occurs;
12. Variant type: it indicates the type of data received, 0x0C stands for String;
13. Value: it provides the content of the data received. In this case, it contains the string "Officine Gaudino" as expected.

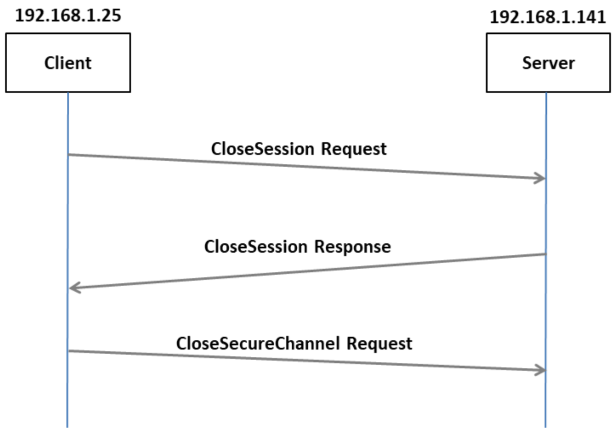
## Connection closing

*Figure 4.10a* illustrates the messages exchanged between Client and Server while closing the OPC UA connection.



| No. | Time | Source | Destination | Protocol | Lengt | Info |
|---|---|---|---|---|---|---|
| 9540 | 115.265983 | 192.168.1.25 | 192.168.1.141 | OpcUa | 117 | UA Secure Conversation Message: CloseSessionRequest |
| 9542 | 115.267424 | 192.168.1.141 | 192.168.1.25 | OpcUa | 106 | UA Secure Conversation Message: CloseSessionResponse |
| 9611 | 116.407957 | 192.168.1.25 | 192.168.1.141 | OpcUa | 111 | CloseSecureChannel message: CloseSecureChannelRequest |

(a) *Wireshark OPC UA close connection*



(b) *Close connection diagram*

Figure 4.10: OPC UA protocol analysis - close connection

The Client, firstly, closes the session with a *CloseSessionRequest*. After receiving the *CloseSessionResponse*, the Client sends a *CloseSecureChannelRequest* and the device is disconnected from the Server.

## Packet length

Wireshark provides the possibility to analyse the lengths of the packets exchanged. During this test, Server and Client were running the programs presented before and 1366 packets were collected. The services involved were: open connection, close connection, read and write data. The table below provides detailed results of the test. Overall, the average packets length is 189 bits, the shortest packet is 82 bits long while the longest is 1647 bits. There is an high percentage of packets exchanged with a length between 80 and 319 bits, this situation happens because the read and write services were the most adopted and they exchange integer or short string.

| Packet lengths [bits] | Count | Average | Max value | Min value | Percentage |
|---|---|---|---|---|---|
| 0-19 | 0 | - | - | - | 0% |
| 20-39 | 0 | - | - | - | 0% |
| 40-79 | 0 | - | - | - | 0% |
| 80-159 | 724 | 129,10 | 82 | 157 | 53,0% |
| 160-319 | 603 | 214,85 | 162 | 309 | 44,14% |
| 320-639 | 17 | 439,76 | 321 | 606 | 1,24% |
| 640-1279 | 12 | 1037,08 | 716 | 1221 | 0,88% |
| 1280-2559 | 10 | 1574,20 | 1307 | 1647 | 0,73% |
| 2560-5119 | 0 | - | - | - | 0% |
| 5120 and greater | 0 | - | - | - | 0% |
| Total | 1366 | 189,37 | 82 | 1647 | 100% |

Table 4.1: Packet length

### 4.4.2 Performance analysis

The goal of the test is to measure the delay of the reading data service. This delay is measured using the performance view of the UAExpert Client. The test is performed running the read service over the *S_parameter[1]* variable for 100 cycles. Then the response delay is measured for all service calls. *Figure 4.11* shows the result of the test. On the x-axis, the cycles indication is reported. While on the y-axis, all the delays measurements, expressed in milliseconds, are reported.
The results obtained in this test are the following:

- Average value: 3.07ms;
- Maximum value: 14.72ms;
- Minimum value: 1.27ms.

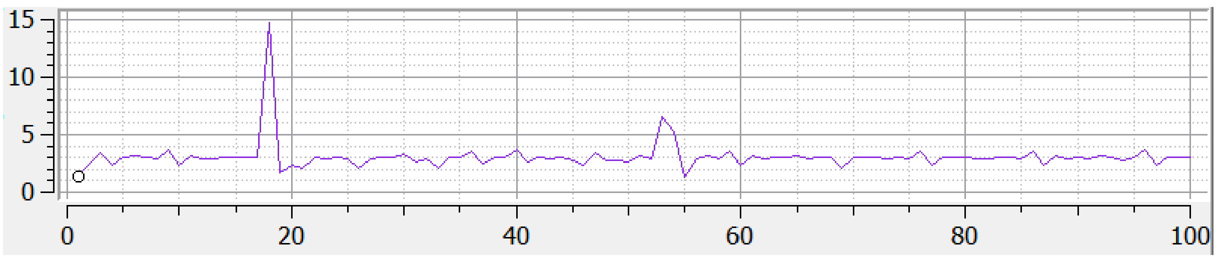


Figure 4.11: UaExpert test OPC UA Client performance

This tool is launched for all the variables defined in the AddressSpace and the results obtained are listed below:

| Variable | Type | Average value [ms] | Maximum value [ms] | Minimum value [ms] |
|---|---|---|---|---|
| ServerName | STRING | 2.77 | 3.76 | 1.52 |
| Company | STRING | 2.66 | 3.60 | 1.43 |
| S_parameter[0] | UDINT | 2.98 | 5.95 | 1.80 |
| S_parameter[1] | UDINT | 3.07 | 14.72 | 1.27 |
| S_parameter[2] | UDINT | 3.00 | 4.82 | 1.28 |
| ClientName1 | STRING | 2.75 | 3.83 | 1.44 |
| C1_parameter[0] | UDINT | 2.98 | 3.87 | 1.50 |
| C1_parameter[1] | UDINT | 2.99 | 3.72 | 1.17 |
| C1_parameter[2] | UDINT | 3.03 | 4.23 | 1.37 |

Table 4.2: UAExpert performance analysis

The data in the table shows that the read service has an average delay of 2.91ms. This result is strongly influenced by the noise of the company network, but the test demonstrates that packets exchanged between Client and Server are delivered in a reasonable amount of time.

# Chapter 5

# Conclusions

This last chapter presents a final analysis of the work done and the future developments of this project.
The connection of independent machines with a central system is a new topic for Officine Gaudino and it represents a starting point in the introduction of automation on its production machinery. It will be a long process because nowadays the larger volumes of yarn production are located in poor countries where the labour is cheaper and central monitoring systems are not necessary. On the other hand, in Europe and United States, companies need more sophisticated machines so these kinds of systems are already requested. For this reason, until now, only few customers require a centralized monitoring system but in future it will be more popular.
This kind of activity opens new opportunities for developing software that monitor the production progresses and status of machines. Besides it allows the company to be a better competitor in the market, because a system totally flexible with the customer requirements is provided and it can be interfaced with existent systems also from other vendors.

#### Activities performed

The steps performed in this activity and the difficulties encountered are described here below.

1. The first activity performed was to examine the environment conditions and the system requirements. An introduction of the textile theory concepts was required to understand the machines features. In particular, the parameters that affect the qualities of processed material, the meaning of yarn twist and draft were analysed. After that, the operating principles of Officine Gaudino machines were examined;

2. The following months were devoted to individuate the communication protocols that could be adopted. Starting from protocols supported by the existent hardware devices, their specifications have been studied and in the end three potential protocols have been individuated: Modbus TCP, EtherNet/IP and OPC UA;

3. The three protocols considered were compared and OPC UA was chosen as the best communication protocol. This choice was made considering network performance metrics such as interoperability, security, vertical integration, frame size, communication model and services provided. Besides, the protocol individuated must implement a flexible system that could be integrated with a wide number of automation devices. The difficulties encountered in this step were to individuate the crucial information that allow to make a comparison among protocols considered;

4. After selecting the communication protocol, the software that allow to exchange data among Clients and Server were developed. Many difficulties have been encountered here. Firstly, since OPC UA is a recent protocol, is not yet widely diffused on Eaton devices so it was difficult to find the necessary documents and practical examples for the protocol implementation. After that, it was necessary to update all software adopted by Officine Gaudino to a newer version. Therefore, it was necessary to verify the compatibility of hardware devices;

5. Then a data model has been developed. It took me some time because a deeper knowledge of the machines behaviour and basic theory concepts of textile technologies were required. This model allows to exchange data common to all machinery of the company, but it is possible to refine the model by adding specific variables;

6. In the final step, analysis of the software implemented are performed. These activities were carried out with Wireshark and UAExpert tools. The former tool monitors the packets exchanged while the latter allows to see the AddressSpace of the Server and to run a performance analysis. In this way it is possible to verify the correctness of the protocol implemented.

### Structural developments

A summary of the structural developments is presented here. Originally the system was composed of several independent machines. Human effort was required to understand the status of each machine and the overall production progresses. *Figure* 5.1 schematizes the original structure. Each machine is composed of a PLC, connected to an HMI, that communicates with motors, sensors and actuators through the CAN protocol.

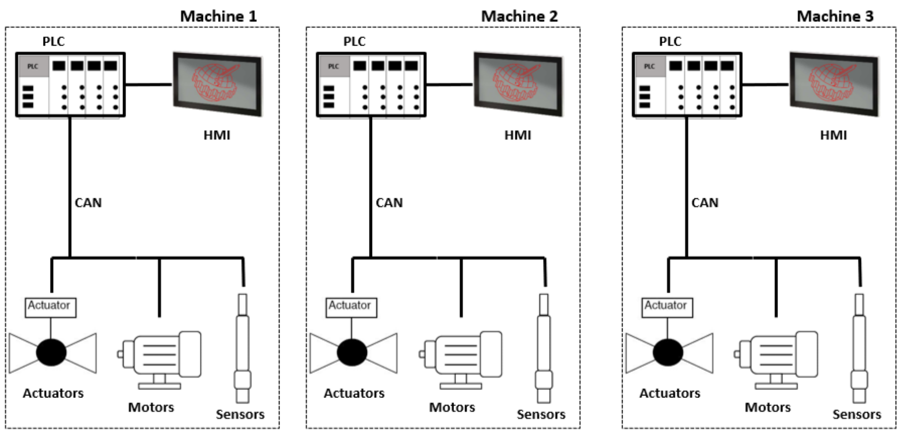


Figure 5.1: System before

*Figure* 5.2 shows the structural changes that this project has introduced. Now all machines are interconnected through a common Ethernet bus and they can exchange data with a central system. So from the central point is possible to organize and manage the production with the goal of reducing the manpower. There is also the possibility to insert a computer on the network to analyse the network performances and to exchange data with the Server.

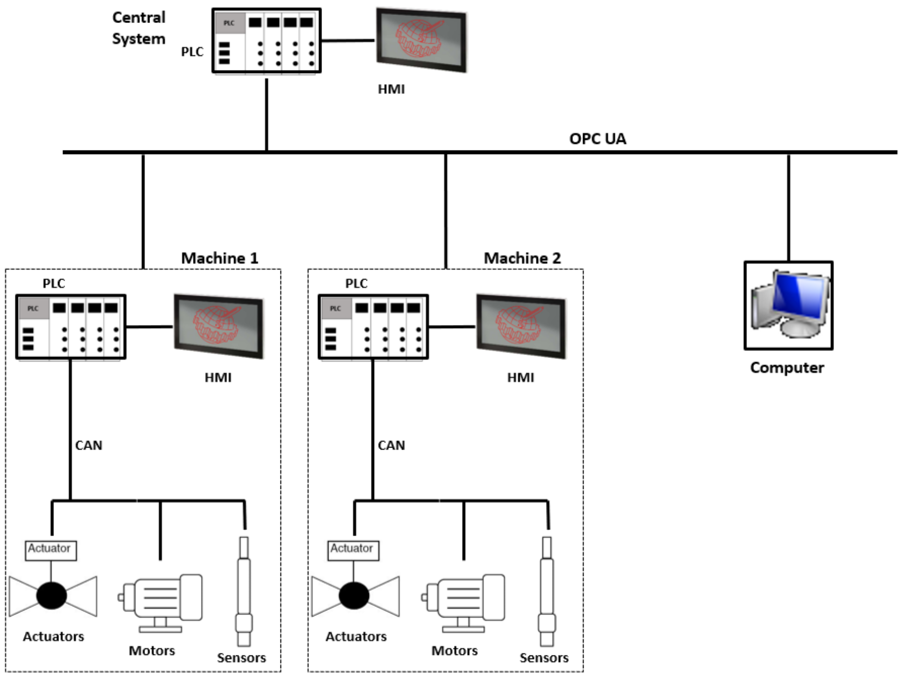


Figure 5.2: System now

*Figure* 5.3 illustrates the future developments of the system (see section 5.1).

### Personal considerations

Personally i can be pleased with the activity carried out because the project specifications have been satisfied. This project is a new topic for the company and it can have important future progresses (see section below). In the past the province where I live, Biella, was worldwide famous for textile industries, so I am proud to have had the possibility to study basic textile theory concepts and to work in a company that design textile machines. Besides this activity has greatly increased my professional knowledge.

## 5.1 Future work

This section illustrates the future system developments, specifying what the company has already in mind and what can be done in the next years. The main system improvements are listed here.

- Company will introduce *Predictive failure analysis* on its systems. They are methods intended to predict imminent failure of systems or components (software or hardware). They also enable mechanisms to avoid failure or recommend maintenance of systems prior to failure. The idea is to replace the CAN protocol with the IO-Link and implements these features. The predictive results are reported in the central system and a more detailed description of the status of each machine is obtained;

- Some customers already require an integration of the central system with MES and other management tools. In this way it is possible, following the Pyramid of Automation terminology, connect the control and field levels with the planning level. OPC UA allows to implement vertical integration of information from sensors, actuators and machines to ERP[16];

- Possibility to integrate the central system with cloud. Therefore a further solution for data and information transport is given, in order to choose the best mechanism for different scenarios.

The *Figure 5.3* schematizes a future prospective of the system.

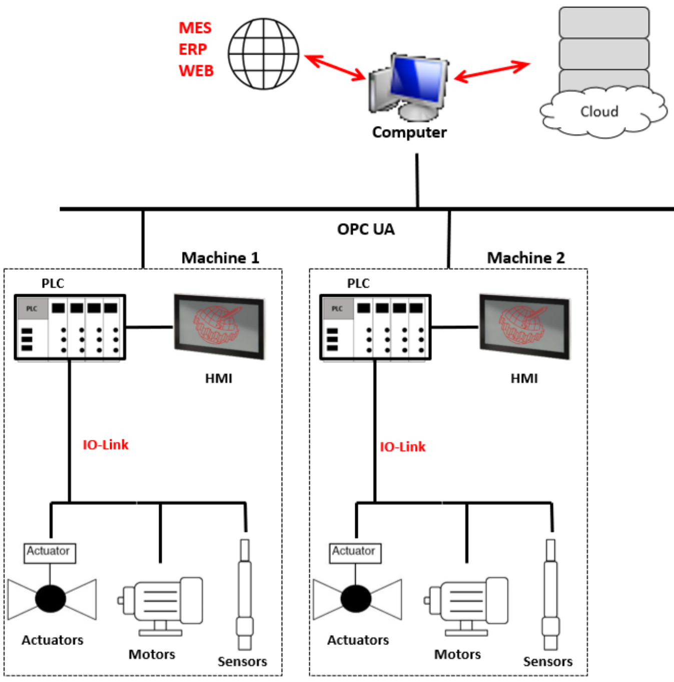


Figure 5.3: System future

# Bibliography

[1] M.P. Groover, *Automation, Production Systems, and Computer-Integrated Manufacturing*, Pearson.

[2] Bogdan M.Wilamowski, J.David Irwin, *Industrial Comunications Systems*, second edition, CR Press.

[3] D. Comer (2005), *Internetworking with TCP/IP Vol. 1: Principles, Protocols, and Architecture*, Prentice Hall.

[4] The Modbus Organization, *Modbus application protocol specification V1.1a*, https://modbus.org/specs.php.

[5] The Modbus Organization, *Modbus messaging on TCP/IP implementation guide V1.0b*, https://modbus.org/specs.php.

[6] ODVA, *The CIP network library, Volume1, Common Industrial protocol*.

[7] ODVA, *The CIP network library, Volume2, EtherNet/IP Adaption of CIP*.

[8] ODVA, *Common industrial protocol and the family of CIP networks*, https://www.odva.org/technology-standards/document-library.

[9] OPC Fundation (2017), *Overview and Concepts*, http://www.opcfoundation.org/UA/Part1.

[10] OPC Fundation (2017), *Security Model*, http://www.opcfoundation.org/UA/Part2.

[11] OPC Fundation (2017), *AddressSpace Model*, http://www.opcfoundation.org/UA/Part3.

[12] OPC Fundation (2017), *Services*, http://www.opcfoundation.org/UA/Part4.

[13] OPC Fundation (2017), *Mappings*, http://www.opcfoundation.org/UA/Part6.

[14] P.T. de Sousa, P. Stuckmann, *Telecommunication Network Interoperability*.

[15] Codesys Online Help, https://help.codesys.com/.

[16] P.Drahos, E.Kucera, O.Haffner, I.Klimo (2018), *Trends in Industrial Communication and OPC UA*.