Local and Remote Human Machine Interfaces for Programmable Logic Controllers

Rafael Rei, José Gaspar
Instituto Superior Técnico / UTL, Lisbon, Portugal
rafael.rei@tecnico.ulisboa.pt, jag@isr.tecnico.ulisboa.pt

ABSTRACT

Programmable Logic Controllers (PLCs) are essential components of Industrial Control Systems (ICSs). In recent years the security flaws of these devices have come under scrutiny, particularly since the cyber-attack Stuxnet and its worldwide repercussions. Trustworthy secure handling of sensitive data, such as product and production knowledge, is just as necessary as the protection against attacks on the networked systems. Local and remote secure communication is a central aspect approached in this report. Despite technological progress, the presence of human operators is still fundamental in production plants. Human Machine Interfaces (HMIs) are the main point of contact between the operator and the machine. The complexity of machines implies an increased complexity of HMIs. Thus, HMIs cannot be considered anymore an accessory to the machine and their improvement has become an important part of the design. This report presents solutions for intercommunication between PLCs and Personal Computers (PCs) with improved security. In addition, a methodology for security insertion based on Petri Nets is proposed. One case study is the automatic insertion of ON/OFF control of remote HMI. A specialized HMI panel, based on a Raspberry Pi is proposed as a low cost alternative to commercial systems and personal computers.

Keywords: PLC, HMI, Ethernet, Security

I. INTRODUCTION

Programmable Logic Controllers (PLCs) are the most common devices for integrating and controlling industrial processes. PLCs work 24/7 and many applications do not require more display than the Light Emitting Diodes (LEDs) of the PLCs. However, some applications would benefit from extra display of data. To display PLCs data, one may consider 7-segments digital displays, serial terminals, Ethernet displays, etc. Ethernet displays are the most flexible solution to display data. One of the focus of this report is the Ethernet-based communication for Human-Machine Interface (HMI) with PLCs and its benefits to the industry community. The vast diffusion of connected devices in the Industry 4.0 has created enormous demand for robust security in response to the growing demand of millions of connected devices and services worldwide. The number of threats is rising daily, and cyber-attacks have been on the increase in both number and complexity. Consequently, cyber-security is another goal of this report that must be studied.

A. Objectives and Challenges

Today’s devices and machines produce high-value data. However, the available data only becomes viable if it can be processed and used to improve a product, to offer a service, or to reduce the costs. A prerequisite for smart services is that devices, machines, and smart services securely exchange data. Otherwise, data, machines, and devices might be compromised or the value of the data might be monetarized by external parties. In the light of what has been stated before, the main objectives of the work are the following: Develop software for Ethernet-based communication with the PLC; Propose one hardware device, based on a Raspberry-pi, to display PLC data; Propose a software solution to send human-input-data to the PLC; Study security issues that may arise by using the Ethernet connection to the PLC; Implement a security measure that controls the data sent and received.

B. Report Structure

Section I introduces the problem to approach in the thesis. Section II presents the background and related work in the industrial networks as well the concept of Petri Net while chapter III describes an analysis of the case study. Section IV provides an overview of the different cases of HMI implemented to the case study and a way to make the communication between the devices more secure. Section V presents an implementation of a Petri Net in order to make the case study even more secure against attacks. Section VI presents all the experiments performed as well as its results. Section VII summarizes the work performed, highlights the main achievements and proposes further work to extend the activities described in this document.

II. BACKGROUND AND RELATED WORK

A. Communications with PLCs

A complex automated industrial system usually needs a Distributed Control System (DCS) — an organized hierarchy of controller systems — to function. In this hierarchy, there is usually a HMI at the top, where an operator can monitor or operate the system. This is typically linked to a middle layer of PLC via a non-time-critical communications system, such as Ethernet. At the bottom of the control chain is the
Field Bus that links the PLCs to the components that do the work, such as sensors and actuators. The origin of the Field Bus was to replace any point-to-point links between the field devices and their controllers by a digital single link on which all the information is transmitted serially and multiplexed in time. A set of rules, Protocol [1], must be defined to accomplish data transfer between the units along the bus. Nowadays, the most used Field Bus protocols are Controller Area Network (CAN) BUS, Process Field Bus (PROFIBUS) and MODBUS. In this report, one will use the MODBUS Transmission Control Protocol/Internet Protocol (TCP/IP) protocol, as it is the Hardware/Software (HW/SW) available in the Laboratory PLCs to use.

The MODBUS Field Bus is a serial communications protocol originally published by Schneider Electric in 1979 for use with its PLCs. MODBUS is often used to connect a supervisory computer with Supervisory Control and Data Acquisition (SCADA) systems. MODBUS TCP/IP is often referred to as MODBUS over Ethernet. For the most part, MODBUS TCP/IP is simply MODBUS packets encapsulated in standard Transmission Control Protocol (TCP) packets. This enables MODBUS TCP/IP devices to immediately and easily connect and communicate over existing Ethernet networks. MODBUS TCP/IP allows the use of multiple Masters, speeds in the gigabit range and the ability to have as many slaves as the physical layer can handle [7].

B. Open Platform Communication

FieldBus is not enough, for Industry 4.0, to ensure the safety of a factory. To make such a statement, the concept of Open Platform Communication (OPC) was created. Open Platform Communication is the interoperability standard for the secure and reliable exchange of data in the industrial automation space and other industries [8]. The OPC standard is a series of specifications developed by industry vendors, end-users and software developers. When the standard was first released in 1996, its purpose was to abstract PLC specific protocols (such as MODBUS, PROFIBUS, etc.) into a standardized interface allowing HMI/SCADA systems to interface with a “middleman” who would convert generic-OPC read/write requests into device-specific requests and vice-versa. Many types of OPC software were developed over the years, being the Open Platform Communication Unified Architecture (OPC UA) one of the most used. OPC UA allows secure communication between devices, by encrypting and signing messages as well as assigning users and passwords. However, internal security must also be applied, to the system to be able to defend itself against cyber attacks, e.g. password phishing. Consequently, the analysis of Petri Nets, a representation of discrete event systems, is necessary for such internal security to be constructed.

C. Petri Nets, Representation of Discrete Event Systems

A Discrete Event System (DES) is a discrete-state, event-driven system, that is, its state evolution depends entirely on the occurrence of asynchronous discrete events over time. DESs are present in multiple systems and can be implemented on controllers like PLCs. A Petri Net (PN) is a tool that allows representing DESs.

Definition. (Petri net graph) A Petri net graph is a weighted bipartite graph \((P, T, A, \omega)\) where: \(P\) is the finite set of places (one type of node in the graph); \(T\) is the finite set of transitions (the other type of node in the graph); \(A \subseteq (P \times T) \cup (T \times P)\) is the set of arcs from places to transitions and from transitions to places in the graph; \(\omega: A \rightarrow \{1, 2, 3, \ldots\}\) is the weight function on the arcs. The set of places will be normally represented by \(P = \{p_1, p_2, \ldots, p_n\}\), and the set of transitions by \(T = \{t_1, t_2, \ldots, t_m\}\). A typical arc is of the form \((p_i, t_j)\) or \((t_j, p_i)\), and the weight related to an arc is a positive integer.

A marked Petri Net is a five-tuple \((P, T, A, \omega, \mu)\) where \((P, T, A, \omega)\) is a PN graph and \(\mu\) is a marking of the set of places \(P; \mu = [\mu(p_1), \mu(p_2), \ldots, \mu(p_n)] \in \mathbb{N}^n\) is the row vector associated with \(\mu\). For simplicity, marked Petri nets will be referred as just Petri nets.

A transition \(t_j \in T\) in a PN is said to be enabled if

\[\mu(p_i) \geq \omega(p_i, t_j) \quad \text{for all} \quad p_i \in I(t_j)\]

In words, transition \(t_j\) in the PN is enabled when the number of tokens in \(p_i\) is at least as large as the weight of the arc connecting \(p_i\) to \(t_j\), for all places \(p_i\) that are input to transition \(t_j\).

The incidence matrix, \(D\), of a Petri net, is a \(N \times M\) matrix - where \(N\) is the number of places and \(M\) the number of transitions - whose \((j, i)\) entries are of the form:

\[d_{ji} = \omega(t_j, p_i) - \omega(p_i, t_j)\]

The incidence matrix, \(D\), can also be written using the Pre-Condition, \(D^+\), and Post-Condition matrices, \(D^-\), in the form \(D = D^+ - D^-\), whose \((j, i)\) entries can be presented as \(d^+_{ji} = \omega(p_i, t_j)\) and \(d^-_{ji} = \omega(t_j, p_i)\), respectively. PN models can be characterized by several properties.

Property 1. Reachability According with the Method of Matrix Equations:

\[Dq(k) = \mu(k + 1) - \mu(k)\]

if there is not a \(q(k)\) (firing vector) that satisfies this condition, then \(\mu(k + 1)\) is not reachable.

Property 2. Liveness PN Liveness can be classified in 5 different levels: Level 0 - a particular transition can never fire (Dead Transition); Level 1 - live (potentially fireable), iff it may fire, i.e. it is in some firing sequence; Level 2 - live if it can fire arbitrarily often, i.e. if for every positive integer \(k\), it occurs at least \(k\) times in some firing sequence; Level 3 - live if it can fire infinitely often, i.e. if for every positive integer \(k\), it occurs at least \(k\) times; Level 4 - live if it may always fire (Live Transitions).

Property 3. Temporal Invariance A necessary condition to prove that the PN is time invariant is the existence of a matrix \(q\) such that:

\[Dq(k) = 0\]

Property 4. Boundedness Given a PN: \(C = (P, T, A, \omega, \mu_0)\), a place, \(p_i \in P\), is \(k\)-bounded if \(\mu_0^i \leq k\). For
all the \( \mu' = (\mu'_1, ..., \mu'_i, ..., \mu'_n) \in R(C, \mu_0) \) (Reachable Set), a PN is \( k \)-bounded if all places are \( k \)-bounded.

**Property 5. Conservation** A PN \( C = (P, T, A, \omega, \mu_0) \) is strictly conservative if, for all \( \mu' \in R(C, \mu_0) \):

\[
\sum_{p_i \in P} \mu'_i(p_i) = \sum_{p_i \in P} \mu(p_i)
\]

**Property 6. Safeness** A PN is safe if all points of the PN are safe. A PN point is said to be safe if:

\[
\forall \mu' \in R(C, \mu_0) : \mu'_i \leq 1
\]

**Property 7. Coverability** Given a PN \( C = (P, T, A, \omega, \mu_0) \) with a initial mark \( \mu_0 \) and states \( \mu, \mu' \in R(C, \mu_0) \), then one have that \( \mu' \) has coverage of \( \mu \) case \( \mu'_i \leq \mu_i \), for all points \( p_i \in P \).

### III. PLC INTERFACE AND CASE STUDY

This section analyzes the hardware components of the PLC used in the experiments and details the case study that will be considered for developing HMI communications and to study the respective security.

#### A. Hardware and Software Setup

The model of PLC that will be used in the experimental section of this report is Schneider’s TSX P57 Premium. This model has a rack with five slots. The Central Processing Unit (CPU) model (P57 2634M) and the Ethernet module (ETY Port) are plugged into slots '0' and '1', respectively, although physically they are a single hardware piece. The Ethernet module accepts a connection to a TCP/IP network with X-WAY MODBUS messaging facility on a TCP/IP profile. The Inputs module (DEY 16D2) is plugged into slot '2' and the Outputs module (DSY 16T2) is plugged into slot '4'. The figure 1 show the laboratory PLC attach with a hardwired HMI.

![PLC: Schneider PS7 Terminal](image)

Fig. 1. PLC TSX P57 connected to a Hardwired HMI terminal.

#### B. Case Study, Alarm System

This subsection III-B describes the specification of aims an intrusion detection alarm system, for a restricted space. The alarm system has external inputs and outputs, namely three inputs (switches), and four outputs (three LEDs and one Buzzer). The alarm also has a keyboard formed by a set of push buttons arranged in a 4x3 array. The keyboard lines are connected to the PLC input module and the columns are connected to the PLC output module. To perform the reading of a key it is necessary to feed the columns alternately, i.e cyclically. The 60ms timers, more specifically Timers ON Delay (TONs), were used. The TON ensures that the column is powered for a short time, and simultaneously the other 2 columns are OFF. When the key is pressed, the circuit is closed and the signal is sent to one of the PLC inputs, i.e to one of the lines, thus identifying which key was pressed. The Alarm System has five main modes of operation, namely:

- **(Mode 0) OFF**: This mode deactivates the alarm completely. In this mode all outputs are OFF.
- **(Mode 1) PRESENCE DETECTOR**: This mode is activated, first by turning ON the Presence Detector Switch. If the window switch is then turn ON, the Yellow LED is signalized as well the buzzer on the panel. The Yellow LED turns ON and after 5 seconds and an acoustic signal with the duration of 1 second should be emitted.
- **(Mode 2) ALARM**: This mode is activated, first by turning ON the Alarm Switch. In this mode, the alarm is to be used. The alarm only starts after the user insert the correct password. A fixed time (30 seconds) is set so that the user can leave the room. After this time, the Green LED is activated, which allows the user to know that the alarm is set. If the window is open, the Yellow LED is set for 5 seconds followed by the Red LED and a periodic acoustic signal of the buzzer that is ON for 1 second and OFF for 2 seconds. The only way to stop the Buzzer, the LEDs as well the Alarm Mode is by introducing the correct password in the Keyboard.
- **(Mode 3) HARDWARE TEST**: Mode that allows the user to check if all the outputs are functional.
- **(Mode 4) CHANGE OF PASSWORD**: Mode that allows the user to change the current password. When entering this mode, one needs, first, to introduce the correct current password, follow by the new one.

When the PLC is initialized, the initial step OFF MODE is activated. Next, to enter the HARDWARE TEST it is necessary to press the "#" key, and automatically, the tests are made to the hardware. After exiting the HARDWARE TEST, one is in an intermediate step, the DECISION STEP. From here, the operator of the alarm can choose which way to enter: the PD MODE, ALARM MODE or the CODE MODE.

### IV. INDUSTRIAL PROCESSES LOCAL AND REMOTE HMI

In this section are presented different hardware and software setups for a human to interface with a PLC controlling an industrial process.

#### A. HMI Scenarios based on Communications

**Case 0. Hardwired HMI Terminal** - Direct communication between the PLC and a hardwired HMI terminal. In this case there are no communications via TCP or MODBUS messages. There are only input/output cables that guarantee the system operation. See figure 1. This case has been addressed in section III.

**Case 1. Local HMI** - Communication inside the factory, via Ethernet. This case can be divided into two sub-cases, using personal computers or specialized devices for HMI. See
Fig. 2. Case 1, Local HMI. HMI communications inside the factory. (a) case 1.1, PC with an HMI that can be developed in e.g. MATLAB, (b) case 1.2, Raspberry-Pi with a HMI that can be developed in e.g. Python.

figures 2(a) and 2(b). The HMI, developed in the local area networks, does not allow bridges to the outside of the factory.

**Case 2, Remote HMI - Communication from/to the outside the factory network.** See figure 6. Typically, one uses a local router bridging one or more local area networks to a wide area network.

### B. Local HMI - HMI on a PC

As one moves into the Industry 4.0 era, it is evident that the creation of virtual HMIs is overtaking the installation of the hardwired HMIs. Designers are adding Graphical User Interfaces (GUIs), moving from physical buttons to virtual buttons on the GUI, increasing the number of tasks the HMI can perform.

The first part of this subsection proposes a HMI based in the MATLAB programming environment, considering the network configuration in 2(a). The objective is the construction of a virtual HMI terminal, similar to the terminal presented in figure 1.

**Fig. 3.** Graphical User Interface developed in MATLAB.

Using the tool Graphical User Interface Design Environment (GUIDE) [5], from the Instrument Control Toolbox of MATLAB, one can construct a GUI as shown in the figures above. This GUI allows a user to interact with the PLC without any hardware, using only virtual push buttons, text boxes and MODBUS TCP/IP messages. The MODBUS TCP/IP messages are created using the MATLAB function modbus [6].

In figure 3, the system is in alarm mode, where the window is open. One can observe that the user gets some feedback like LEDs, the buzzer, the Wrong Password Bit, the actual mode of the system and even the keys that the user already introduced. One must give particular importance to these last three outputs: the **Wrong Password Bit**, the **actual mode** and the **keys already pressed** because the physical terminal does not present such components. Saying this, with only that hardwired terminal, one cannot know if the password introduced was correct, how many keys one already pressed (to activate/deactivate the alarm) and in which mode the alarm system is at the time. In sum, the development of such HMI is an upgrade of the terminal available in the Laboratory.

### C. Local HMI - Specialized HMI Panel

In this second part of the local HMI development, a Raspberry Pi is used instead of a PC. This choice enables an easy end-user human-machine-interface and convenient data log accessibility. The figure 4 shows the hardware setup, where one can see that the Raspberry Pi is used both as a server and client. In order for the PLC to know that the messages reaching him are MODBUS messages, one must establish X-WAY addresses in the Unity Pro configurations.

**Fig. 4.** Hardware Setup of the Raspberry Pi and the PLC, with the configured addresses.

The program, built in Python 3.4, running in the Raspberry Pi terminal displayed on an LCD allows user input of alarm modes and the writing of **Memory Bit** (%M) and **Memory Word** (%MW) while displaying updated system information, such as the PLC outputs. The Client sends TCP/IP messages to the server and according to the order received, this last one sends a MODBUS message to the PLC that, in correct operation, must send back some sort of feedback.

**Fig. 5.** Raspberry Pi Client View.

Figure 5 shows the Raspberry Pi terminal, that corresponds to the HMI that the client has at his disposal in the device.
The client can view the terminal in a display that is attached to the Raspberry Pi through an HDMI cable. The development of the Python program can be seen in section VI.

D. Remote HMI, Encrypted Communication with OPC UA

The purpose of this subsection is to access the PLC from different places and different devices. In the previous subsection has created a server that was capable of receive information, from a client, process it, and send it to the PLC. The server IP Address, running in Raspberry Pi, is the only one attached to an X-Way Address, that it is registered in the PLC program. In other terms, this server, implemented in Raspberry Pi, is the only one allowed to communicate with the PLC. The figure 6 shows the setup to make such a statement happened. The figure 6 allow communication with the PLC from 2 different ways: Through the Local Area Network (LAN), from two devices that will be the users, marked by (a); Through the Wide Area Network (WAN), from an Android Phone client marked by (b).

In order to implement the remote HMI in an Android, one used the application Pydroid 3, available in the Play Store. Pydroid 3 is a Python 3 Integrated Development Environment (IDE) for Android that will serve as the tool to actuate as a client with the Raspberry Pi. The client program, created using the Python language, can be reused in the application Pydroid 3. The access of the server, via Android, from any place at any time, is one more factor to point out that extra security is needed. That way one must consider an OPC UA as necessary security for the communication between Client and Server.

Encrypted Communication - OPC UA

OPC UA complements the existing OPC industrial standard by adding essential new properties such as platform independence, high availability and Internet capability. The encryption process regards the communication between a PC/Android (the OPC Client) and a Raspberry Pi (the OPC Server). The encryption process is made possible thanks to the open source implementation, FreeOpcUa. The main steps of the implementation are described in section VI-C.

V. SYSTEM DEVELOPMENT AND SECURITY INSERTION BASED ON PETRI NETS

The aim of this section is the design and implementation of a system based on a Petri Net that stops the remote HMI whenever the operator finds a defective mode of operation possibly resulting from non-legitimate remote actions. The challenge is incorporating in the tool-chain a procedure allowing the change during operation of the system behavior with respect to specific inputs.

The subsections V-A and V-C focus is the study of a Petri Net based on presence detector and alarm modes, where is only considered a hardwired HMI as human-machine interaction, while in the subsection V-E is considered the addition of the remote/local HMI to the alarm system, running in parallel with the hardwired terminal.

A. Petri Net Alarm Graph and Properties

This subsection aims to the development of a Petri Net that describes some of the events and the state evolution of the alarm system. From the PN, one can obtain the incidence matrix. Using the Petri Net editor, PIPE2, one is able to construct a Petri Net of the alarm system, such the one described in the figure 7.

Figure 8 shows the incidence matrix, $D_I$, corresponding to the Petri Net shown in figure 7. The incidence matrix is obtained automatically from a Petri net graph created with PIPE2.

Fig. 6. Remote/Local HMI Setup, with the configured addresses.

Fig. 7. Petri Net Alarm Graph.

Fig. 8. Incidence Matrix of the Petri Net Alarm.
The MATLAB toolbox tpn5 [9] [2] was used to obtain the reachability tree and the reachable set of the Petri net alarm. The toolbox returned a finite reachable set. By inspection, it is possible to state that the transitions are all live (Liveness of level 4) since one finds always a sequence of transitions that allows firing a specific transition under liveness study. From each state, one can reach any other state of the reachable set, which indicates that the Petri Net is time invariant. The analysis of the reachable set also allows one to conclude that the Petri Net is 1-bounded and it is conservative as no places have more than one mark and all states contain a summation of marks equal to one. Since the Petri Net is 1-bounded, one can assure that it is also safe. No state covers another state and therefore the Petri Net does not present the coverability property.

**B. Petri nets Complemented with IO**

In order to interact with real systems, one has to add Inputs/Outputs (I/O) to the Petri nets, creating Interpreted Petri nets. In this report are used the Input-Output Place-Transition (IOPT) tools, a GUI that allow adding inputs and outputs to building Petri nets defined in accordance with typical hardware used for creating embedded devices. In section V-C the IOPT tools are used to design the alarm system and then a toolchain is presented to convert automatically the net to a PLC program.

a) Interpreted Petri nets: Interpreted Petri Nets (IPNs) [4] add I/Os to the standard Petri nets. Figure 9(b) shows the Petri net has a number of places associated with outputs, that turn ON/OFF actuators, and a number of transitions associated with inputs, that fire according to sensors. Interpreted Petri Nets can generalized logic functions like I10 +/OR I11, as well as activate multiple outputs in the same place, as seen in Sequential function Chart (SFC) programming language.

**C. IOPT Tools**

Systems are specified using IOPT Nets, a class of Petri nets extended with input and output capabilities necessary to communicate with the external world, to allow reading sensors, manipulate actuators, communicate with other systems and implement user interfaces. With the IOPT Tools, one is able to define input signals, output signals, input events and output events. Input signals are used to obtain information from the external world, for instance, to read the state of sensors, read user interface buttons, or read signals from other systems. Output signals may be used to manipulate mechanical actuators, illuminate LEDs, or send information to other systems. Input Events are triggered by changes in input signals and output events will cause changes in output signals.

In figure 10 is represented the alarm system without the keyboard, obtained by using the IOPT tools. This model contains 10 places shown as yellow circles, 20 transitions (14 untimed and 6 timed transitions) drawn as blue rectangles, 7 input signals (3 switches in ON position + 3 switches in OFF position + 1 timed input) depicted as blue rectangles, 7 input events drawn as blue triangles, 4 output signal (3 LEDs + 1 Buzzer) represented as green rectangles and 4 output events presented as green triangles. The arrow lines connecting Places and Transitions are Arcs.

**D. Conversion of Petri nets to Structured Text**

In order to generate the Structured Text (ST) - a non-graphical, text-based programming language - program that later is used in the Laboratory PLC controller, an ST compiler was developed in MATLAB, based on [3]. Figure 11 shows, in a very simple way, the necessary steps to acquire an ST code.
made, namely the algorithm of data acquisition (the extracted file from PIPE2 is an Extensible Markup Language (XML) file while the one from IOPT tools is a Petri Net Markup Language (PNML) file) as well as the automatic insertion of inputs/outputs in the Compiler (PIPE2 only allows PN construction, and so the insertion of I/Os must be made by hand, while IOPT tools have the option of already adding and declaring the I/Os IDs).

E. Insertion of Remote Access Control into the Petri net

The data encryption and password based login are necessary, but leave out the attacks based on e.g. password phishing. Given the system password, the attacker would have full access to perform changes in the alarm system. This way, the virtual HMI must have some sort of access control.

Separation of Inputs Graphical Representation

Being an event in the same transition prevents the control of just one of them. One needs to split the commands of the virtual HMI, memory bits, from the ones of the hardwired HMI, physical inputs. The separation of two alternative inputs, just one of them. One needs to split the commands of the two inputs in the same transition prevents the control of access to the remote HMI, where

$$D_{IM} = \begin{bmatrix} D_I & D_M \end{bmatrix}$$  \hspace{1cm} (5)

The incidence matrix $D_{IM}$ in equation 5 is a selection of the incidence matrix 8 that is duplicated. More in detail, the number of transitions is duplicated, with the same values in each column, while maintaining the same number of places. In this context, $D_I = D_M$.

$$D_{IP}^{N_P \times M_P} = \begin{bmatrix} D_{IM}^{N_{IP} \times M_{IM}} & 0^{N_{IM} \times 2} \end{bmatrix} \begin{bmatrix} \nu_P & 1 \end{bmatrix}$$  \hspace{1cm} (6)

The incidence matrix $D_P$ in equation 6 shows the incidence matrix of the IPN Alarm with separate inputs in parallel with the incidence matrix of the IPN subsystem, $D_S$ that controls the access to the remote HMI, where $D_S = \begin{bmatrix} -1 & 1 \\ 1 & -1 \end{bmatrix}$. Matrix $D_{IP}^{N_P \times M_P}$ has zero anti-diagonal block matrices, as there are no arcs connecting the graphs represented by $D_{IM}$ and $D_S$. However one must correlate these two IPNs in order to the control remote access interact with the IPN alarm. To do this, one must create a supervisor.

Supervision to Enable/Disable Inputs

In this report, the supervisor is used to constrain some activities - like the remote control of the alarm system. This constrain can be represented by mathematical equations, like the one seen in equation 7.

$$\sum_M q_M + \mu_{S2} \leq 1 \quad M \in \text{MemoryTransitions}$$  \hspace{1cm} (7)

This mathematical constrain translates the following condition: the firing of any $\% M$ transition ($q_M$) cannot occur if there is a marking in the place of Refuse Remote HMI ($\mu_{S2}$). Knowing the condition, one can solve the Generalized Linear Constraint:

$$L \mu_P + F q_P + C \nu_P \leq b$$  \hspace{1cm} (8)

where $\mu_P$ is the marking vector for system $P$; $q_P$ is the firing vector since $l_0$; $\nu_P$ is the number of transitions (firing) that can occur. $\mu_P \in N^0$, $\nu_P \in N^0$, $q_P \in N^0$, $L \in Z^{1 \times N_P}$, $F \in Z^{1 \times M_P}$, $C \in Z^{1 \times M_P}$, $b \in Z$. Let $M_M$ be the number of columns in the matrix $D_M$ and $M_I$ be the number of columns in the matrix $D_I$. The vectors $L,F,C$, as well the value of $b$ can be seen below.

$$L = \begin{bmatrix} 0^{1 \times N_{IM}} & 0 \end{bmatrix} \quad F = \begin{bmatrix} 0^{1 \times M_I} & 1^{1 \times M_M} & 0 & 0 \end{bmatrix} \quad C = \begin{bmatrix} 1 \end{bmatrix} \quad b = 1$$

The supervisor Pre-Conditions incidence matrix ($D^-_C$) is computed as:

$$D^-_C = \max(0, L \mu_P + C, F) = \begin{bmatrix} 0^{1 \times M_I} & 1^{1 \times M_M} & 0 & 1 \end{bmatrix}$$  \hspace{1cm} (9)

The supervisor Post-Conditions incidence matrix ($D^+_C$) is computed as:

$$D^+_C = \max(0, F - \max(0, L \mu_P + C)) - \min(0, L \mu_P + C) = \begin{bmatrix} 0^{1 \times M_I} & 1^{1 \times M_M} & 0 & 1 \end{bmatrix}$$  \hspace{1cm} (10)

The initial marking ($\mu_{C_0}$) is:

$$\mu_{C_0} = b - L \mu_{P_0} - C \nu_{P_0} = 1$$  \hspace{1cm} (11)

Fig. 12. Separating inputs and memory flag. (a) Interpreted Petri Net with Separate Physical and Memory Inputs (b) Proposed IPN Subsystem that enables/disables the remote access.

Fig. 13. IPN with Separate inputs connected to the proposed Subsystem through a supervisor $P_C$. Note that $P_{S1}$ is marked and unmarked by events also acting on $P_C$. Place $P_C$ is acted by additional events.
The solutions in equations 9, 10 and 11 allow the determination of the full incidence matrix, $D_T$ that includes the supervisor incidence matrix, $D_C = D_N^S - D_N^C$, and the incidence matrix $D_P$. The figure 13 is a graphical representation of the incidence matrix $D_T$.

**Case Study** The supervisor implemented in the previous section can be removed and their arcs can be added directly to the place $P_{S1}$. This way, the system becomes clearer, the incidence matrix becomes equal to that of equation 6 while maintaining the planned constraints. Figure 14 shows the final system that will be analyzed and implemented in this part of the report.

![Fig. 14. IPN with Separate inputs connected to the proposed Subsystem without a supervisor.](image)

Although figure 14 has an incidence matrix equal to that of 6, its graphical representation is different. The Pre-Conditions and Post-Conditions matrices of this new IPN are not the same since place $P_{S1}$ receives and consumes a token when the transition $T_{B2}$ fires. Thus, the construction of a PN must always be done from the Pre-Conditions and Post-Conditions matrices and not from the incidence matrix. The incidence matrix, $D_P$, of equation 6 could be used to the alarm system if one did not consider timed transitions.

![Fig. 15. IOPT net alarm section connected to the Subsystem.](image)

The solutions in equations 9, 10 and 11 allow the determination of the full incidence matrix, $D_T$ that includes the supervisor incidence matrix, $D_C = D_N^S - D_N^C$, and the incidence matrix $D_P$. The figure 13 is a graphical representation of the incidence matrix $D_T$.

**Case Study** The supervisor implemented in the previous section can be removed and their arcs can be added directly to the place $P_{S1}$. This way, the system becomes clearer, the incidence matrix becomes equal to that of equation 6 while maintaining the planned constraints. Figure 14 shows the final system that will be analyzed and implemented in this part of the report.

The solutions in equations 9, 10 and 11 allow the determination of the full incidence matrix, $D_T$ that includes the supervisor incidence matrix, $D_C = D_N^S - D_N^C$, and the incidence matrix $D_P$. The figure 13 is a graphical representation of the incidence matrix $D_T$.

**Case Study** The supervisor implemented in the previous section can be removed and their arcs can be added directly to the place $P_{S1}$. This way, the system becomes clearer, the incidence matrix becomes equal to that of equation 6 while maintaining the planned constraints. Figure 14 shows the final system that will be analyzed and implemented in this part of the report.

![Fig. 14. IPN with Separate inputs connected to the proposed Subsystem without a supervisor.](image)

Although figure 14 has an incidence matrix equal to that of 6, its graphical representation is different. The Pre-Conditions and Post-Conditions matrices of this new IPN are not the same since place $P_{S1}$ receives and consumes a token when the transition $T_{B2}$ fires. Thus, the construction of a PN must always be done from the Pre-Conditions and Post-Conditions matrices and not from the incidence matrix. The incidence matrix, $D_P$, of equation 6 could be used to the alarm system if one did not consider timed transitions.

![Fig. 15. IOPT net alarm section connected to the Subsystem.](image)

The incidence matrix $D_T$, shown in 16, represents the real representation of the IPN alarm with the subsystem implemented. The number of places in $D_M$ is the same at $D_I$, but the number of transitions is not. This can be seen more clearly in figure 15, where only in places between a transition that fires with physical inputs (like switches and buttons) can have a new transition that fires with a memory bit. The timed transitions, as the name says, cannot be controlled by a memory bit. Therefore if $D_I$ is a matrix $N \times M$, where $N$ is the number of places and $M$ is the number of transitions, $D_M$ will be a matrix $N \times (M - P)$, where $P$ is the number of timed transitions. The total incidence matrix, $D_T$, of the IPN alarm, with the subsystem, will be a matrix $(N+2) \times (M+(M-P)+2)$. Since the IPN has 10 Places, 14 untimed/physical transitions and 6 timed transitions, as detailed in subsection V-C, matrix $D_T$ will have the form $12 \times 36$.

### VI. EXPERIMENTS AND RESULTS

To validate the methodologies and techniques presented in sections III, IV and V, some experiments were performed.

#### A. Alarm Program based on SFC, ST and LD

In this subsection is presented the results, in datalogs, of the implementation of the alarm system in the Laboratory PLC, where the connection between the PLC and the HMI terminal is purely by hard-wire I/O cables. The languages used to created such alarm system were Structured Text, Ladder Diagram and SFC.

**Hardware Test** In the graph of figure 17 one can observe the beginning of the developed program. To be able to go to any functionality of the implemented program you have to always perform, first a test of the hardware. To do this, simply press the '#' key. As soon as the key is pressed, it activates the buzzer and the various LEDs alternately, to check if there is any problem with the hardware.

![Fig. 17. Hardware Test Mode Datalog.](image)

**Presence Detector Mode**

After performing the hardware test, one can operate in two different ways. One is the presence detector function, which is activated by placing the switch in the presence detector position. As soon as this mode is activated, the green LED will be active to indicate that it is operating. The window switch simulates the presence detector, and as soon as it is active,
the yellow LED lights up and then the buzzer is activated only once for each detected presence. The presence detector is activated only when it recognizes a rising edge of the signal. In fig 18 one can verify that, in the presence detector mode, the window was opened twice, so the buzzer and the yellow LED were active in the same way twice.

**Alarm Mode** Looking at figure 19 one can verify that the first attempt to activate the alarm was rejected, since the code entered was incorrect (as seen in the dashed red rectangle), also evidenced by the **Memory Bit Wrong Password**. After that, the first dashed green rectangle contains the correct code to activate the alarm, followed by 30 seconds of delay until the alarm is operational. If the green LED turns ON it means that the alarm is now ON. Due to a presence - window switch is ON - the yellow LED, followed by the red LED and the buzzer is energized, being this last one an acoustic periodic signal. The second dashed green rectangle contains the correct code, introduced by the user, that deactivates the alarm mode.

**B. HMI Setup based on the Raspberry Pi**

In this subsection is presented the implementation of the virtual HMI terminal in a Raspberry Pi. The server created has eleven options of service request being them referred in figure 5. In the next paragraph is described the construction of one of them: option 6 (Turn ON the Alarm).

Fig. 20. Activation of the alarm mode using the Raspberry Pi terminal.

**Option 6 - Turn ON the Alarm** After the user enters the number of the desirable option, the server sends a response message, via TCP/IP, asking for the correct password to activate the alarm mode of the system. Meanwhile, the server will send a request MODBUS to the PLC, asking them for the actual correct password of the alarm system. The message MODBUS send to the PLC is the array: \[0, 1, 0, 0, 0, 6, 1, 03, 1, 44, 0, 9\]. Each number represent a byte and the first seven bytes represent the MODBUS Application Protocol (MBAP) header. 03 corresponds to the function **Read Holding Registers**, 1 and 44 correspond to the address of first register (Memory Word) to read - \(\%\text{MW}(1 \times 256 + 44 \times 1) = \%\text{MW}300\). The last two bytes 0 and 9 correspond to the number of memory words to read - \(0 \times 256 + 9 \times 1 = 9\) registers. These nine \(\%\text{MW}\) correspond to the keyboard password, currently saved in the PLC memory. The server will then make a comparison between the password inserted in the client terminal. If the password inserted is the correct one, the server, first, will send, to the PLC, the message: \[0, 1, 0, 0, 0, 6, 1, 05, 1, 246, 255, 0\], necessary to change the value of the memory bit responsible for triggering of the alarm mode. In order to do such task, this message must have, as function code, the value 05 - **Write Single Coil** - and as data, the address of coil (\(\%\text{M}(1 \times 256 + 246 \times 1) = \%\text{M}502\)) and the value to write (in order to set the \(%\text{M}\) to 1, one must present in the last 2 bytes the values 255 and 0). Being turned ON the memory bit of the alarm mode, a message of feedback is sent to the client, as seen in figure 20.

**C. Secure HMI based on OPC UA**

The implementation of the OPC UA in the virtual HMI, constructed in Python, can be seen below step by step.

**Step 1** The server can be started by establishing an endpoint with the IP Address of the Raspberry Pi, in port 4840 - the typical OPC UA TCP Protocol port.

**Step 2** Next, the server certificate and private key must be loaded, in order to enable endpoints with signing and encryption. Without this step the messages sent/received between the Server and Clients could be easily read.

**Step 3** Now that the server is created, one must add writable variables. In order to do so, nodes in the server must be created. Each node can have multiple variables. It is with the variables inserted in this created nodes that the communication Client-Server, using the OPC UA, is possible.
**Step 4** It takes only two nodes, each one with one variable (a string) to make the communication a success. The client sends to the server his command (input key) and the client’s terminal prints the text every time the variable in the node Text is modified. Two more variables were also added: the User and the Password, being these two variables not writable.

**Step 5** Setting all the variables, the server can be started.

In figure 21, one can see the OPC UA Client, where in the red rectangle, the client entered a user/password not existing on the server, so the connection was declined and disconnected. On the other hand, in the green rectangle a new user connected with success.

![Image](image_url)

**Fig. 21.** Client View of the correct login in the OPC UA Server (in the green rectangle). Incorrect login in the OPC UA Server (in the red rectangle).

### D. Remote Access Turn OFF/ON for Security Control

Since the Petri Net chosen in section V only presents the alarm mode and the presence detector mode, one must exchange the server code so that when the user selects option 6 (Turn ON the alarm) or 7 (Turn OFF the alarm), the introduction of a password is not requested. The datalog of figure 22 was obtained thanks to the implementation of the converter of IOPT nets into a ST program.

The datalog of the alarm with remote access control, seen in figure 22, can be divided into three parts.

![Image](image_url)

**Fig. 22.** Alarm mode datalog of the ST program generated from the IOPT net alarm with the subsystem that enables/disables the remote HMI.

The first part corresponds to the normal alarm mode, triggered by remote/local HMI, seen in the left green rectangle of the graph. The alarm mode is turned ON, using option 6 (Turn ON the alarm) of the virtual HMI. The alarm mode is finalized by choosing option 7 (Turn OFF the alarm) of the virtual terminal.

The second part, seen inside the red rectangle, corresponds to a scenario where the memory bit responsible for controlling the remote/local HMI changes its value to 1. In this case, the remote/local HMI is denied. The user tries to turn ON the alarm via virtual HMI (marked by the orange rectangle) but the green LED does not lights up after 30 seconds. The only way to turn ON the alarm mode is by switching the physical alarm switch.

In the third part, signaled by the right green rectangle, the memory bit of the remote HMI is turned OFF, and so the normal activity is resumed, where the user can interact with the alarm system using both virtual and hardwired HMI terminals.

### VII. Conclusion and Future Work

The work described in this thesis is focused on the study of Ethernet-based communication with the PLC. Two main technical constraints were originally identified in this work, (i) security issues by using the Ethernet connection to the PLC and (ii) HMI to send human-input-data from the display to the Raspberry Pi and from the Raspberry Pi to the PLC. In particular one has studied the OPC UA architecture in order to improve the security in different levels (direct interaction with PLC, inside the factory and interaction from outside the factory). Previous works provide good starting points for the thesis, such as MATLAB to PLC MODBUS communication, PLC programming based on Petri-nets and created hardware devices for directly interfacing with the PLC digital inputs and outputs. Furthermore, it was proposed a methodology to insert security, namely control of the remote HMI, based on Petri Nets. Future work in this subject includes the specification of the triggering of the memory bit responsible for the remote/local HMI control. This memory bit could be turned ON by too many failed remote login attempts or by the attempt of trying firing another transition of the alarm mode sequence that is not the true one.

## References


