UNIVERSITY OF VERONA

---

Department of COMPUTER SCIENCE

Master's Degree in
COMPUTER SCIENCE AND ENGINEERING

Master Thesis

# Towards Process Comprehension of Industrial Control Systems: a Framework for Analyzing Industrial Systems

*Candidate:*
**Marco OLIANI**
VR457249

*Supervisor:*
Prof. **Massimo MERRO**

*Co-supervisor:*
Prof. **Ruggero LANOTTE**
University of Insubria

---

Academic Year 2022/2023

*"If you spend more on coffee than on IT security, you will be hacked. What's more, you deserve to be hacked"*
(Richard Clarke)

# Abstract

Bla bla bla

# Contents

**1 Introduction** **1**
- 1.1 Contribution . . . . . . . . 1
- 1.2 Outline . . . . . . . . 1

**2 Background** **3**
- 2.1 Industrial Control Systems in a nutshell . . . . . . . . 3
- 2.2 ICS components . . . . . . . . 4
  - 2.2.1 SCADA systems . . . . . . . . 4
    - 2.2.1.1 SCADA architecture . . . . . . . . 5
  - 2.2.2 Field devices . . . . . . . . 7
  - 2.2.3 Programmable Logic Controllers . . . . . . . . 7
    - 2.2.3.1 PLC Architecture . . . . . . . . 8
    - 2.2.3.2 PLC Programming . . . . . . . . 9
    - 2.2.3.3 PLC Security . . . . . . . . 10
  - 2.2.4 Remote Terminal Units . . . . . . . . 11
  - 2.2.5 Human-Machine Interface . . . . . . . . 12
  - 2.2.6 Cybersecurity components . . . . . . . . 12
- 2.3 Communication Networks . . . . . . . . 13
  - 2.3.1 ICS Communication Protocols . . . . . . . . 13
    - 2.3.1.1 Modbus . . . . . . . . 13
    - 2.3.1.2 EtherNet/IP . . . . . . . . 17
    - 2.3.1.3 Common Industrial Protocol (CIP) . . . . . 19
    - 2.3.1.4 Other Protocols . . . . . . . . 20

3 State of the Art 21
- 3.1 Literature on Process Comprehension . . . . . . . . . . . . . 22
- 3.2 Ceccato et al.'s methodology for analyzing water-tank systems . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 24
  - 3.2.1 Testbed . . . . . . . . . . . . . . . . . . . . . . . . . . . 25
  - 3.2.2 Scanning of the System and Data Pre-processing . . . 28
  - 3.2.3 Graphs and Statistical Analysis . . . . . . . . . . . . . 30
  - 3.2.4 Invariants Inference and Analysis . . . . . . . . . . . 31
  - 3.2.5 Businness Process Mining and Analysis . . . . . . . . 33
  - 3.2.6 Application . . . . . . . . . . . . . . . . . . . . . . . . 34
  - 3.2.7 Limitations . . . . . . . . . . . . . . . . . . . . . . . . 40

4 A framework to improve Ceccato et al.'s work. 49
- 4.1 Phase 0: General Improvements . . . . . . . . . . . . . . . . . 50
  - 4.1.1 Single Programming Language . . . . . . . . . . . . . 50
  - 4.1.2 System Independence . . . . . . . . . . . . . . . . . . 51
  - 4.1.3 Flexibility and Ease on Use . . . . . . . . . . . . . . . 52
- 4.2 Phase 1: Data Pre-processing . . . . . . . . . . . . . . . . . . 52
- 4.3 Phase 2: Graphs and Statistical Analysis . . . . . . . . . . . . 52
- 4.4 Phase 3: Invariants Analysis . . . . . . . . . . . . . . . . . . . 52
  - 4.4.1 Invariants Generation . . . . . . . . . . . . . . . . . . 52
- 4.5 Phase 4: Businness Process Analysis . . . . . . . . . . . . . . 52
- 4.6 Extra Information on the Physics . . . . . . . . . . . . . . . . 52

5 Case study: the iTrust SWaT System 53

6 Our framework at work: reverse engineering of the SWaT system 55
- 6.1 Pre-processing . . . . . . . . . . . . . . . . . . . . . . . . . . 55
- 6.2 Graph Analysis . . . . . . . . . . . . . . . . . . . . . . . . . . 55
  - 6.2.1 Conjectures About the System . . . . . . . . . . . . . 55
- 6.3 Invariants Analysis . . . . . . . . . . . . . . . . . . . . . . . . 55
  - 6.3.1 Actuators Detection . . . . . . . . . . . . . . . . . . . 55
  - 6.3.2 Daikon Analysis and Results Comparing . . . . . . . 55
- 6.4 Extra information on the Physics . . . . . . . . . . . . . . . . 55

6.5 Businness Process Analysis . . . . . . . . . . . . . . . . . . . . 55

**7 Conclusions** **57**
7.1 Discussions . . . . . . . . . . . . . . . . . . . . . . . . . . . . 57
7.2 Guidelines . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 57
7.3 Future work . . . . . . . . . . . . . . . . . . . . . . . . . . . . 57

**List of Figures** **59**

**List of Tables** **61**

**References** **63**

# Chapter 1

## Introduction

LOREM ipsum dolor bla bla bla. Ma dove metto l'abstract? Prova di interlinea che direi posso anche andare bene, ma bisogna poi vedere il tutto come si incastra alla fine, in modo da ottenere un bel risultato alla vista.

## 1.1 Contribution

Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged. It was popularised in the 1960s with the release of Letraset sheets containing Lorem Ipsum passages, and more recently with desktop publishing software like Aldus PageMaker including versions of Lorem Ipsum.

## 1.2 Outline

The thesis is structured as follows:

**Chapter 2:** provides background on the topics covered in this thesis: Industrial Control Systems (ICSs), Supervisory Control And Data Acquisition (SCADA), Programmable Logic Controllers (PLCs) and other devices, industrial communication protocols.

**Chapter 3:**

**Chapter 4:**

**Chapter 5:**

**Chapter 6:**

**Chapter 7:**

# Chapter 2

# Background

## 2.1 Industrial Control Systems in a nutshell

INDUSTRIAL CONTROL SYSTEMS (ICSs) are information systems used to control industrial processes such as manufacturing, product handling, production, and distribution [1].

ICSs are often found in critical infrastructure facilities such as power plants, oil and gas refineries, and chemical plants.

ICSs are different from traditional IT systems in several key ways. Firstly, ICSs are designed to control physical processes, whereas IT systems are designed to process and store data. This means that ICSs have different requirements for availability, reliability, and performance. Secondly, ICSs are typically deployed in environments that are harsh and have limited resources, such as extreme temperatures and limited power. Thirdly, the protocols and hardware used in ICSs are often proprietary and not widely used outside of the industrial sector.

ICSs are becoming increasingly connected to the internet and other networks, which has led to increased concerns about their security. Industrial systems were not originally designed with security in mind, and many of them have known vulnerabilities that could be exploited by attackers. Additionally, the use of legacy systems and equipment can make it difficult to implement security measures. As a result, ICSs are increasingly seen as a potential target for cyber attacks, which could have serious consequences for the safe and reliable operation of critical infrastructure.

The increasing connectivity of ICSs and the associated security risks have led to a growing interest in the field of ICS security. Researchers and practitioners are working to develop new security technologies, standards, and best practices to protect ICSs from cyber attacks. This includes efforts to improve the security of ICS networks and devices, as well as the development of new monitoring and detection techniques to identify and respond to cyber attacks.

## 2.2 ICS components

*Industrial control systems* (ICSs) are composed of several different components that work together to monitor and control industrial processes.

### 2.2.1 SCADA systems

*Supervisory Control And Data Acquisition* (**SCADA**) is a system of software and hardware elements that allows industrial organizations to [2]:

- Control industrial processes locally or at remote locations
- Monitor, gather, and process real-time data
- Directly interact with devices such as sensors, valves, pumps, motors, and more through human-machine interface (HMI) software
- Record events into a log file

The SCADA software processes, distributes, and displays the data, helping operators and other employees analyze the data and make important decisions.

SCADA systems are known for their ability to monitor and control large-scale industrial processes, and for their ability to operate over long distances. This makes them well-suited for use in remote locations or for controlling processes that are spread out over a wide area. However, the same features that make SCADA systems so useful also make them vulnerable to cyber attacks.

SCADA systems were not originally designed with security in mind, and many of them have known vulnerabilities that could be exploited by attackers. Additionally, the use of legacy systems and equipment can make it difficult to implement security measures. As a result, SCADA systems are increasingly seen as a potential target for cyber attacks, which could have serious consequences for the safe and reliable operation of critical infrastructure.

To secure SCADA systems, it is important to implement security measures such as network segmentation, secure communication protocols, and access control. Additionally, it is important to monitor SCADA systems for unusual activity and to implement incident response procedures to quickly detect and respond to any security breaches.

#### 2.2.1.1 SCADA architecture

According to the *Purdue Enterprise Reference Architecture* (PERA), or simply **Purdue Model**, SCADA architecture consists in **six levels** each representing a functionality [3], as shown in Figure 2.1:

- Level 0 (**Processes**): contains **field devices** (2.2.2), or *sensors*.
- Level 1 (**Intlligent Devices**): includes **local or remote controllers** that sense, monitor and control the physical process, such as **PLCs** (2.2.3) and **RTUs** (2.2.4). Controllers interface directly to the field devices reading data from sensors and sending commands to actuators.
- Level 2 (**Control Systems**): contains computer systems used to supervising and monitoring the physical process: they provide a **Human-Machine Interface** (*HMI*, 2.2.5) and *Engineering Workstations* (EW) for operator control.

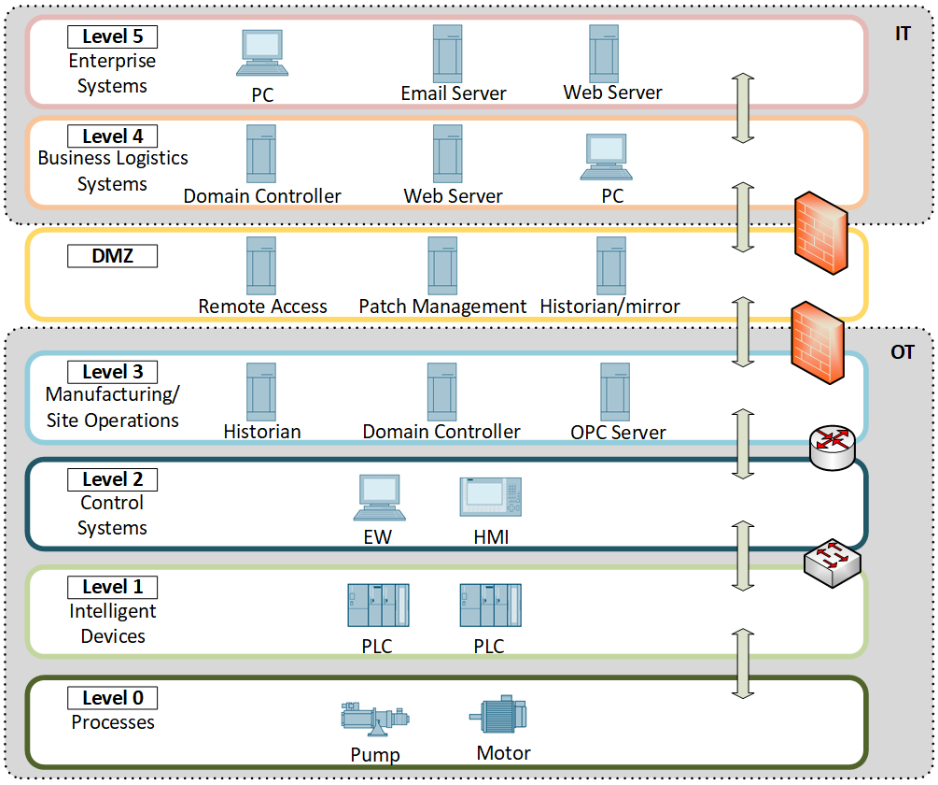

IT
Level 5
Enterprise Systems
PC
Email Server
Web Server
Level 4
Business Logistics Systems
Domain Controller
Web Server
PC
DMZ
Remote Access
Patch Management
Historian/mirror
OT
Level 3
Manufacturing/ Site Operations
Historian
Domain Controller
OPC Server
Level 2
Control Systems
EW
HMI
Level 1
Intelligent Devices
PLC
PLC
Level 0
Processes
Pump
Motor

***Figure 2.1:*** *SCADA architecture schema*

- Level 3 (**Manufactoring/Site Operations**): comprises systems used to manage the production workflow for plant-wide control: they collate informations from the previous levels and store them in Data Historian servers.

- **Industrial Demilitarized Zone** (DMZ): intermediate level that connects the *Operational Technology* (OT) part (levels 0-3) with the *Information Technology* (IT) part of the system (levels 4 and 5). Communication takes place indirectly through services such as *proxy servers* and *remote access servers*, which act as intermediaries between the two environments.

- Level 4 (**Business Logistics Systems**): collect and aggregates data from the Manufactoring/Site Operations level overseeing the IT-related activities to generate **reporting** to the Enterprise System layer. At this layer we can find application and e-mail servers, and *Enterprise Resource Planning* (ERP) systems.

- Level 5 (**Enterprise Systems**): represents the enterprise network, used for the business-to-business activities and for business-to-client purpose services. At Enteprise Systems level are typical IT services such as mail servers, web servers and all the systems used to manage the ongoing process.

### 2.2.2 Field devices

*Field devices* are the **sensors** and **actuators** that are used to collect data from the process and control it. Examples of field devices include temperature sensors, pressure sensors, valves and pumps.

### 2.2.3 Programmable Logic Controllers

A *Programmable Logic Controller* (PLC) is a **small and specialized industrial computer** having the capability of controlling complex industrial and manifacturing processes [4].

Compared to relay systems and personal computers, PLCs are optimized for control tasks and industrial environments: they are rugged and designed to withdraw harsh conditions such as dust, vibrations, humidity and temperature: they have more reliability than personal computers, which are more prone to crash, and they are more compact a require less maintenance than a relay system. Furthermore, I/O interfaces are already on the controller, so PLCs are easier to expand with additional I/O modules (if in a rack format) to manage more inputs and ouputs, without reconfiguring hardware as in relay systems when a reconfiguration occours.

PLCs are more *user-friendly*: they are not intended (only) for computer programmers, but designed for engineers with a limited knowledge in programming languages: control program can be entered with a simple and intuitive language based on logic and switching operations instead of a general-purpose programming language (*i.e.* C, C++, ...).

#### 2.2.3.1 PLC Architecture

The basic hardware architecture of a PLC consists of these elements [5]:

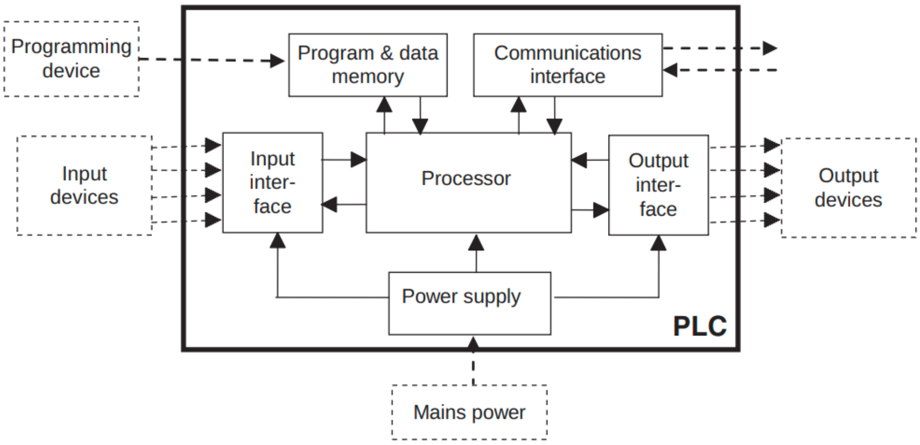

Programming device
Program & data memory
Communications interface
Input devices
Input inter-face
Processor
Output inter-face
Output devices
Power supply
PLC
Mains power

*Figure 2.2: PLC architecture*

- **Processor unit (CPU)**: contains the microprocessor. This unit interpretes the input signals from I/O modules, executes the control program stored in the Memory Unit and sends the output signals to the I/O Modules. The processor unit also sends data to the Communication interface, for the communication with additional devices.

- **Power supply unit:** converts AC voltage to low DC voltage.

- **Programming device:** is used to store the required program into the memory unit.

- **Memory Unit:** consists in RAM memory and ROM memory. RAM memory is used for storing data from inputs, ROM memory for storing operating system, firmware and user program to be executed by the CPU.

- **I/O modules:** provide interface between sensors and final control elements (actuators).
- **Communications interface:** used to send and receive data on a network from/to other PLCs.

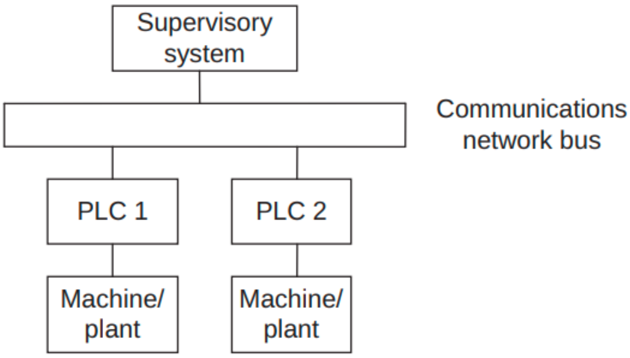


***Figure 2.3:*** *PLC communication schema*

#### 2.2.3.2 PLC Programming

Two different programs are executed in a PLC: the **operating system** and the **user program**.

The operating system tasks include executing the user program, managing memory areas and the *process image table* (memory registers where inputs from sensors and outputs for actuators are stored).

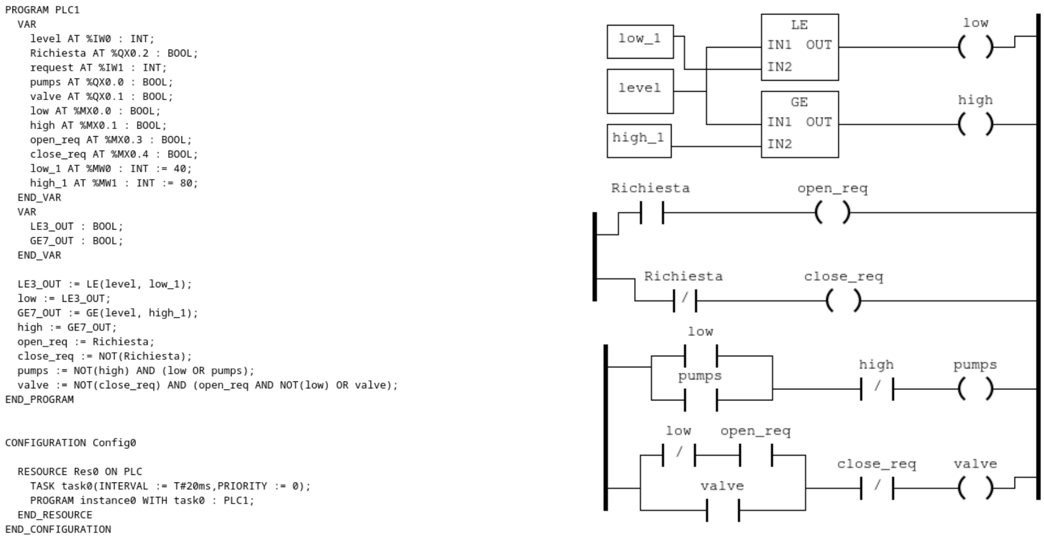
The user program needs to be uploaded on the PLC via the programming device and runs on the process image table in *scan cycles*: each scan is made up of three phases [6]:

1. reading inputs from the process images table
2. execution of the control code and computing the physical process evolution

3. writing output to the process image table to have an effect on the physical process. At the end of the cycle, the process image table is refreshed by the CPU

Standard PLCs **programming languages** are basically of two types: **textuals** and **graphicals**. Textual languages include languages such as *Instruction List* (IL) and *Structured Text* (ST), while *Ladder Diagrams* (LD), *Function Block Diagram* (FBD) and *Sequential Function Chart* (SFC) belong to the graphical languages.

Graphical languages are more simple and immediate comparing to the textual ones and are preferred by programmers because of their features and simplicity, in particular the **Ladder Logic programming** (see Figure 2.4 for a comparison).



```
PROGRAM PLC1
  VAR
    level AT %IW0 : INT;
    Richiesta AT %QX0.2 : BOOL;
    request AT %IW1 : INT;
    pumps AT %QX0.0 : BOOL;
    valve AT %QX0.1 : BOOL;
    low AT %MX0.0 : BOOL;
    high AT %MX0.1 : BOOL;
    open_req AT %MX0.3 : BOOL;
    close_req AT %MX0.4 : BOOL;
    low_1 AT %MW0 : INT := 40;
    high_1 AT %MW1 : INT := 80;
  END_VAR
  VAR
    LE3_OUT : BOOL;
    GE7_OUT : BOOL;
  END_VAR

  LE3_OUT := LE(level, low_1);
  low := LE3_OUT;
  GE7_OUT := GE(level, high_1);
  high := GE7_OUT;
  open_req := Richiesta;
  close_req := NOT(Richiesta);
  pumps := NOT(high) AND (low OR pumps);
  valve := NOT(close_req) AND (open_req AND NOT(low) OR valve);
END_PROGRAM


CONFIGURATION Config0

  RESOURCE Res0 ON PLC
    TASK task0(INTERVAL := T#20ms,PRIORITY := 0);
    PROGRAM instance0 WITH task0 : PLC1;
  END_RESOURCE
END_CONFIGURATION
```

***(a)*** *Example of ST programming* ***(b)*** *Example of Ladder Logic*

***Figure 2.4:*** *Comparison between ST language and Ladder Logic*

#### 2.2.3.3 PLC Security

PLCs were originally designed to operate as closed systems, not connected and exposed to the outside world via communication networks:

the question of the safety of these systems, therefore, was not a primary aspect. The advent of Internet has brought undoubted advantages, but has introduced problems relating to the safety and protection of PLCs from external attacks and vulnerabilities.

Indeed, a variety of different communication protocols used in ICSs are designed to be efficient in communications, but do not provide any security measure i.e. confidentiality, authentication and data integrity, which makes these protocols vulnerable against many of the IT classic attacks such as *Replay Attack* or *Man in the Middle Attack*.

Countermeasures to enhance security in PLC systems may include [7]:

- protocol modifications implementing **data integrity**, **authentication** and **protection** against *Replay Attacks*
- use of *Intrusion Detection and Prevention Systems* (IDP)
- creation of *Demilitarized Zones* (DMZ) on the network

In addition to this, keeping the process network and Internet separated, limiting the use of USB devices among users to reduce the risks of infections, and using strong account management and maintenance policies are best practices to prevent attacks and threats and to avoid potential damages.

### 2.2.4 Remote Terminal Units

*Remote Terminal Units* (RTUs) are computers with radio interfacing similar to PLCs: they transmit telemetry data to the control center or to the PLCs and use messages from the master supervisory system to control connected objects [8].

The purpose of RTUs is to operate efficiently in remote and isolated locations by utilizing wireless connections. In contrast, PLCs are designed for local use and rely on high-speed wired connections. This key difference allows RTUs to conserve energy by operating in low-power mode for extended periods using batteries or solar panels. As a result, RTUs consume less energy than PLCs, making them a more sustainable and cost-effective option for remote operations.

Industries that require RTUs often operate in areas without reliable access to the power grid or require monitoring and control substations in remote locations. These include telecommunications, railways, and utilities that manage critical infrastructure such as power grids, pipelines, and water treatment facilities. The advanced technology of RTUs allows these industries to maintain essential services, even in challenging environments or under adverse weather conditions.

### 2.2.5 Human-Machine Interface

The *Human-Machine Interface* (HMI) is the hardware and software interface that operators use to monitor the processes and interact with the ICS.
An HMI shows the operator and authorized users information about system status and history; it also allows them to configure parameters on the ICS such as set points and, send commands and make control decisions [9].

The HMI can be in the form of a physical panel, with buttons and indicator lights, or PC software.

### 2.2.6 Cybersecurity components

*Cybersecurity components*, as seen in section 2.2.3.3 about PLCs security, are used to protect ICSs from cyber threats and vulnerabilities. They can include firewalls, *Intrusion Detection and Prevention systems* (IDP), and *Security Information and Event Management* (SIEM) systems.

## 2.3 Communication Networks

*Communication Networks* are the networks that are used to connect the different components of the ICS and allow them to communicate with each other. Communication networks can include wired and wireless networks, such as Ethernet/IP, Modbus, DNP3 and others.

### 2.3.1 ICS Communication Protocols

As mentioned in Section 2.1, industrial systems differ from classical IT systems in the purpose for which they are designed: controlling physical processes the former, processing and storing data the latter. For this reason, ICSs require different communication protocols than traditional IT systems for real time communications and data transfer.

A wide variety of industrial protocols exists: this is because originally each vendor developed and used its own proprietary protocol. However, these protocols were often incompatible with each other, resulting in devices from different vendors being unable to communicate with each other.
To solve this problem, standards were defined with a view to allowing these otherwise incompatible device to intercommunicates.

Among all the various protocols, some have risen to prominence as widely accepted standards. These *de facto* protocols are commonly utilized in industrial systems due to their proven reliability and effectiveness. In the following sections, we will provide a brief overview of some of the most prevalent and widely used protocols in the industry.

#### 2.3.1.1 Modbus

*Modbus* is a serial communication protocol developed by Modicon (now Schneider Electric) in 1979 for use with its PLCs [10] and designed expressly for industrial use: it facilitates interoperability of different devices connected to the same network (sensors, PLCs, HMIs, ...) and it is also often used to connect RTUs to SCADA acquisition systems.

Modbus is the most widely used communication protocol among industrial systems because it has several advantages:

- simplicity of implementation and debugging
- it moves raw bits and words, letting the individual vendor to represent the data as it prefers
- it is, nowadays, an **open** and ***royalty-free*** protocol: there is no need to sustain licensing costs for implementation and use by industrial device vendors

Modbus is a **request/response** (or *master/slave*) protocol: this makes it independent of the transport layer used.

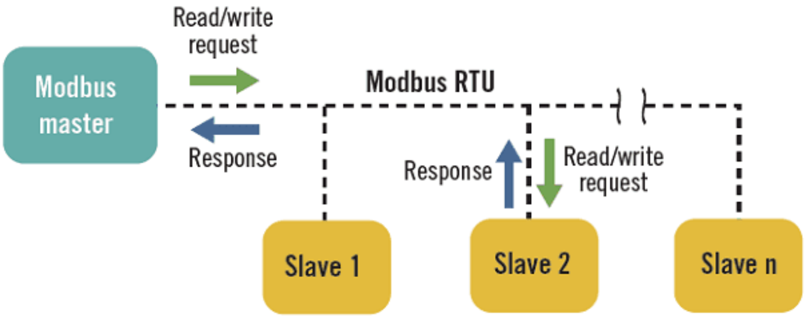


***Figure 2.5:*** *Modbus Request/Response schema*

In this kind of architecture, a single device (master) can send requests to other devices (slaves), either individually or in broadcast: these slave devices (usually peripherals such as actuators) will respond to the master by providing data or performing the action requested by the master using the Modbus protocol. Slave devices cannot generate requests to the master [11].

There are several variants of Modbus, of which the most popular and widely used are Modbus RTU (used in serial port connections) and Modbus TCP (which instead uses TCP/IP as the transport layer). Modbus TCP embeds a standard Modbus frame in a TCP frame (see Figure 2.6): both masters and slaves listen and receive data via TCP port 502.

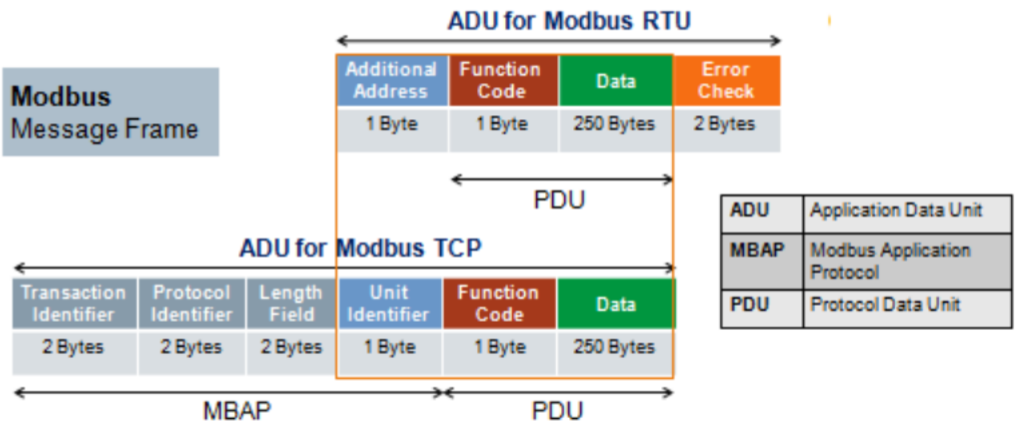

ADU for Modbus RTU
Modbus Message Frame
Additional Address
Function Code
Data
Error Check
1 Byte
1 Byte
250 Bytes
2 Bytes
PDU
ADU for Modbus TCP
Transaction Identifier
Protocol Identifier
Length Field
Unit Identifier
Function Code
Data
2 Bytes
2 Bytes
2 Bytes
1 Byte
1 Byte
250 Bytes
MBAP
PDU
ADU Application Data Unit
MBAP Modbus Application Protocol
PDU Protocol Data Unit

*Figure 2.6: Modbus RTU frame and Modbus TCP frame*

**Modbus registers** Modbus provides four object types, which map the data accessed by master and slave to the PLC memory:

- *Coil*: binary type, read/write accessible by both masters and slaves
- *Discrete Input*: binary type, accessible in read-only mode by masters and in read/write mode by slaves
- *Analog Input*: 16 bits in size (word), are accessible in read-only mode by masters and in read/write mode by slaves
- *Holding Register*: 16 bits in size (word), accessible in read/write mode by both masters and slaves. Holding Registers are the most commonly used registers for output and as general memory registers.

**Modbus Function Codes** *Modbus Function Codes* are specific codes used by the Modbus master within a request frame (see Figure 2.6) to tell the

Modbus slave device which register type to access and which action to perform on it.

Two types of Function Codes exists: for data access and for diagnostic Function Codes list for data access are listed in Table 2.1:

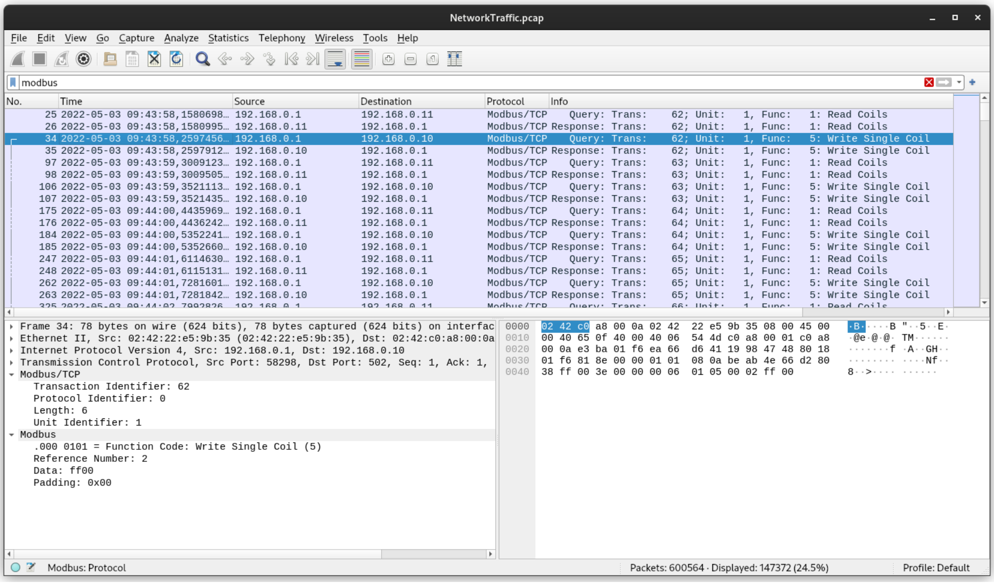
| **Function Code** | **Description** |
|---|---|
| FC01 | Read Coils |
| FC02 | Read Discrete Input |
| FC03 | Read Holding Registers |
| FC04 | Read Analog Input Registers |
| FC05 | Write/Force Single Coil |
| FC06 | Write/Force Single Holding Register |
| FC15 | Write/Force Multiple Coils |
| FC16 | Write/Force Multiple Holding Registers |

***Table 2.1:*** *Modbus Function Codes list*

**Modbus Security Issues** Despite its simplicity and widespread use, the Modbus protocol does not have any security feature, which exposes it to vulnerabilities and attacks.

Data in Modbus are transmitted unencrypted (*lack of confidentiality*), with no data integrity controls (*lack of integrity*) and authentication checks (*lack of authentication*), in addition to the *lack of session*. Hence, the protocol is vulnerable to a variety of attacks, such as Denial of Services (DoS), buffer overflows and reconnaissance activities.

The easiest attack to bring to the Modbus protocol, however, is **packet sniffing**: since, as mentioned earlier, network traffic is unencrypted and the data transmitted is in cleartext, it is sufficient to use a packet sniffer to capture the network traffic, read the packets and thus gather informations about the system such as ip addresses, function codes of requests and to modify the operation of the devices.



```
No.   Time                        Source        Destination   Protocol   Info
   25 2022-05-03 09:43:58,1580698… 192.168.0.1   192.168.0.11  Modbus/TCP    Query: Trans:    62; Unit:   1, Func:   1: Read Coils
   26 2022-05-03 09:43:58,1580995… 192.168.0.11  192.168.0.1   Modbus/TCP Response: Trans:    62; Unit:   1, Func:   1: Read Coils
   34 2022-05-03 09:43:58,2597456… 192.168.0.1   192.168.0.10  Modbus/TCP    Query: Trans:    62; Unit:   1, Func:   5: Write Single Coil
   35 2022-05-03 09:43:58,2597912… 192.168.0.10  192.168.0.1   Modbus/TCP Response: Trans:    62; Unit:   1, Func:   5: Write Single Coil
   97 2022-05-03 09:43:59,3009123… 192.168.0.1   192.168.0.11  Modbus/TCP    Query: Trans:    63; Unit:   1, Func:   1: Read Coils
   98 2022-05-03 09:43:59,3009505… 192.168.0.11  192.168.0.1   Modbus/TCP Response: Trans:    63; Unit:   1, Func:   1: Read Coils
  106 2022-05-03 09:43:59,3521113… 192.168.0.1   192.168.0.10  Modbus/TCP    Query: Trans:    63; Unit:   1, Func:   5: Write Single Coil
  107 2022-05-03 09:43:59,3521435… 192.168.0.10  192.168.0.1   Modbus/TCP Response: Trans:    63; Unit:   1, Func:   5: Write Single Coil
  175 2022-05-03 09:44:00,4435969… 192.168.0.1   192.168.0.11  Modbus/TCP    Query: Trans:    64; Unit:   1, Func:   1: Read Coils
  176 2022-05-03 09:44:00,4436242… 192.168.0.11  192.168.0.1   Modbus/TCP Response: Trans:    64; Unit:   1, Func:   1: Read Coils
  184 2022-05-03 09:44:00,5352241… 192.168.0.1   192.168.0.10  Modbus/TCP    Query: Trans:    64; Unit:   1, Func:   5: Write Single Coil
  185 2022-05-03 09:44:00,5352660… 192.168.0.10  192.168.0.1   Modbus/TCP Response: Trans:    64; Unit:   1, Func:   5: Write Single Coil
  247 2022-05-03 09:44:01,6114630… 192.168.0.1   192.168.0.11  Modbus/TCP    Query: Trans:    65; Unit:   1, Func:   1: Read Coils
  248 2022-05-03 09:44:01,6115131… 192.168.0.11  192.168.0.1   Modbus/TCP Response: Trans:    65; Unit:   1, Func:   1: Read Coils
  262 2022-05-03 09:44:01,7281601… 192.168.0.1   192.168.0.10  Modbus/TCP    Query: Trans:    65; Unit:   1, Func:   5: Write Single Coil
  263 2022-05-03 09:44:01,7281842… 192.168.0.10  192.168.0.1   Modbus/TCP Response: Trans:    65; Unit:   1, Func:   5: Write Single Coil
  325 2022-05-03 09:44:02,7992826… 192.168.0.1   192.168.0.11  Modbus/TCP    Query: Trans:    66; Unit:   1, Func:   1: Read Coils
```

```
› Frame 34: 78 bytes on wire (624 bits), 78 bytes captured (624 bits) on interfac
› Ethernet II, Src: 02:42:22:e5:9b:35 (02:42:22:e5:9b:35), Dst: 02:42:c0:a8:00:0a
› Internet Protocol Version 4, Src: 192.168.0.1, Dst: 192.168.0.10
› Transmission Control Protocol, Src Port: 58298, Dst Port: 502, Seq: 1, Ack: 1,
⌄ Modbus/TCP
    Transaction Identifier: 62
    Protocol Identifier: 0
    Length: 6
    Unit Identifier: 1
⌄ Modbus
    .000 0101 = Function Code: Write Single Coil (5)
    Reference Number: 2
    Data: ff00
    Padding: 0x00
```

```
0000  02 42 c0 a8 00 0a 02 42  22 e5 9b 35 08 00 45 00   ·B·····B "··5··E·
0010  00 40 65 0f 40 00 40 06  54 4d c0 a8 00 01 c0 a8   ·@e·@·@· TM······
0020  00 0a e3 ba 01 f6 ea 66  d6 41 19 98 47 48 80 18   ·······f ·A··GH··
0030  01 f6 81 8e 00 00 01 01  08 0a be ab 4e 66 d2 80   ········ ····Nf··
0040  38 ff 00 3e 00 00 00 06  01 05 00 02 ff 00         8··>···· ······
```

Modbus: Protocol · Packets: 600564 · Displayed: 147372 (24.5%) · Profile: Default

*Figure 2.7: Example of packet sniffing on the Modbus protocol*

To make the Modbus protocol more secure, an encapsulated version was developed within the *Transport Security Layer* (TLS) cryptographic protocol, also using mutual authentication. This version of the Modbus protocol is called **Secure Modbus** or **Modbus TLS**. In addition to this, Secure Modbus also includes X.509-type certificates to define permissions and authorisations [12].

#### 2.3.1.2 EtherNet/IP

*EtherNet/IP* (where IP stands for *Industrial Protocol*) is an open industrial protocol that allows the *Common Industrial Protocol* (CIP) to run on a typical Ethernet network [13]. It is supported by ODVA [14].

EtherNet/IP uses the major Ethernet standards, such as IEEE 802.3 and the TCP/IP suite, and implements the CIP protocol stack at the upper layers of the OSI stack (see Figure 2.8). It is furthermore compatible with the main Internet standard protocols, such as SNMP, HTTP, FTP and DHCP, and other industrial protocols for data access and exchange such as *Open Platform Communication* (OPC).

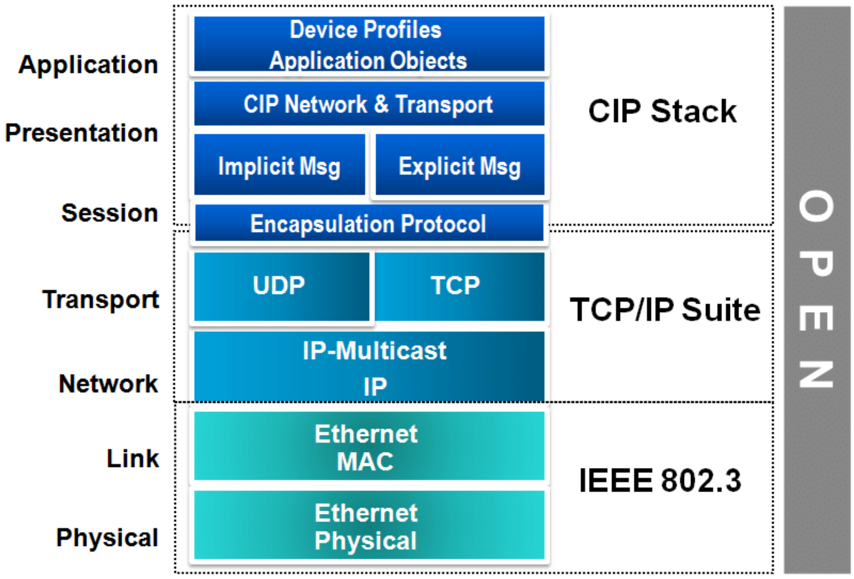

Application
Presentation
Session
Transport
Network
Link
Physical
Device Profiles
Application Objects
CIP Network & Transport
Implicit Msg
Explicit Msg
Encapsulation Protocol
UDP
TCP
IP-Multicast
IP
Ethernet MAC
Ethernet Physical
CIP Stack
TCP/IP Suite
IEEE 802.3
OPEN

***Figure 2.8:*** *OSI model for EtherNet/IP stack*

**Physical and Data Link layer** The use of the IEEE 802.3 standard allows EtherNet/IP to flexibly adopt different network topologies (star, linear, ring, etc.) over different connections (copper, fibre optic, wireless, etc.), as well as the possibility to choose the speed of network devices. IEEE 802.3 in addition defines at Data Link layer the *Carrier Sense Multiple Access - Collision Detection* (CSMA/CD) protocol, which controls access to the communication channel and prevents collisions.

**Transport layer** At the transport level, EtherNet/IP encapsulates messages from the CIP stack into an Ethernet message, so that messages can be transmitted from one node to another on the network using the TCP/IP protocol. EtherNet/IP uses two forms of messaging, as defined by CIP standard [13][15]:

- **unconnected messaging:** used during the connection establishment phase and for infrequent, low priority, explicit messages. Unconnected messaging uses TCP/IP to transmit messages across the network asking for connection resource each time from the *Unconnected*

*Message Manager* (UCMM).

- **connected messaging:** used for frequent message transactions or for real-time I/O data transfers. Connection resources are reserved and configured using communications services available via the UCMM.

  EtherNet/IP has two types of message connection [13]:

  - **explicit messaging:** *point-to-point* connections to facilitate *request-response* transactions between two nodes. These connections use TCP/IP service on port 44818 to transmit messages over Ethernet.
  - **implicit messaging:** this kind of connection moves application-specific **real-time I/O data** at regular intervals. It uses multicast *producer-consumer* model in contrast to the traditional *source-destination* model and UDP/IP service (which has lower protocol overhead and smaller packet size than TCP/IP) on port 2222 to transfer data over Ethernet.

**Session, Presentation and Application layer** At the upper layers, EtherNet/IP implements the CIP protocol stack. We will discuss this protocol more in detail in Section 2.3.1.3.

#### 2.3.1.3 Common Industrial Protocol (CIP)

The *Common Industrial Protocol* (CIP) is an open industrial automation protocol supported by ODVA. It is a **media independent** (or *transport independent*) protocol using a *producer-consumer* communication model and providing a **unified architecture** throughout the manufacturing enterprise [16][17].
CIP has been adapted in different types of network:

- **EtherNet/IP**, adaptation to *Transmission Control Protocol* (TCP) technologies

- **ControlNet**, adaptation to *Concurrent Time Domain Multiple Access* (CTDMA) technologies
- **DeviceNet**, adaptation to *Controller Area Network* (CAN) technologies
- **CompoNet**, adaptation to *Time Division Multiple Access* (TDMA) technologies

**CIP objects** CIP is a *strictly object oriented* protocol at the upper layers: each object of CIP has **attributes** (data), **services** (commands), **connections**, and **behaviors** (relationship between values and services of attributes) which are defined in the **CIP object library**. The object library supports many common automation devices and functions, such as analog and digital I/O, valves, motion systems, sensors, and actuators. So if the same object is implemented in two or more devices, it will behave the same way in each device [18].

**Security** [19] In EtherNet/IP implementation, security issues are the same as in traditional Ethernet, such as network traffic sniffing and spoofing. The use of the UDP protocol also exposes CIP to transmission route manipulation attacks using the *Internet Group Management Protocol* (IGMP) and malicious traffic injection.

Regardless of the implementation used, it is recommended that certain basic measures be implemented on the CIP network to ensure a high level of security, such as *integrity*, *authentication* and *authorization*.

#### 2.3.1.4 Other Protocols

# Chapter 3

# State of the Art

IN TRADITIONAL IT, an attacker aims to understand the behavior of a program through various techniques so as to bring attacks aimed at changing its execution flow, functionalities or bypassing limits imposed by the licensing of such software. These attack techniques include a **preliminary study** of the program: a *static analysis* (i.e., a preliminary analysis of the software without it running) and a *dynamic analysis* (i.e., an analysis performed with the program running).
The result of these two preliminary investigation techniques is a **reverse engineering** of the software, which is useful for identifying any weaknesses or bugs and therefore planning an attack.

In the OT context, however, the concept of *reverse engineering* is also associated with that of ***process comprehension***, a term coined by Green et al.'s [20] to describe the understanding of the characteristics of the system and the physical elements of within it, that are responsible for its proper functioning.

Not much knowledge exists in the literature regarding the collection and analysis of information concerning the understanding and operation of an ICS: in Section 3.1 we will look at a quick overview of some of the existing literature on the subject and in the following sections we will focus in particular on one of the papers exposed.

## 3.1 Literature on Process Comprehension

***Keliris and Maniatikos*** The first approach presented in this section is by Keliris and Maniatakos [21]: they present a methodology for automating the reverse engineering of ICS binaries based on a *modular framework* (called ICSREF) that can reverse binaries compiled with CODESYS, one of the most popular and widely used PLC compilers, irrespective of the language used.

***Yuan et al.*** Yuan et al. [22] propose a *data-driven* approach to discovering cyber-physical systems from data directly: to achieve this goal, they have implemented a framework whose purpose is to identify physical systems and transition logic inference, and to seek to understand the mechanisms underlying these cyber-physical systems, making furthermore predictions concerning their state trajectories based on the discovered models.

***Feng et al.*** Feng et al. [23] developed a framework that can generate system *invariant rules* based on machine learning and data mining techniques from ICS operational data log. These invariants are then selected by systems engineers to derive IDS systems from them.

The experiment results on two different testbeds, the *Water Distribution system* (WaDi) and the *Secure Water Treatment system* (SWaT), both located at the iTrust - Center for Research in Cyber Security at the University of Singapore [24], show that under the same false positive rate invariant-based IDSs have a higher efficiency in detecting anomalies than IDS systems based on a residual error-based model.

***Pal et al.*** Pal et al. [25] work is somewhat related to Feng et al.'s: this paper describes a data-driven approach to identifying invariants automatically using *association rules mining* [26] with the aim of generate invariants sometimes hidden from the design layout. The study has the same objective of Feng et al.'s and uses too the iTrust SwaT System as testbed.

Currently this technique is limited to only pair wise sensors and actuators: for more accurate invariants generation, the technique adopted must be capable of deriving valid constrains across multiple sensors and actuators.

***Winnicki et al.*** Winnicki et al. [27] instead propose a different approach to process comprehension based on the **attacker's perspective** and not limited to mere *Denial of Service* (DoS): their approach is to discover the dynamic behavior of the system, in a semi-automated and process-aware way, through *probing*, that is, slightly perturbing the cyber physical system and observing how it reacts to changes and how it returns to its original state. The difficulty and challenge for the attacker is to perturb the system in such a way as to achieve an observable change, but at the same time avoid this change being seen as a system anomaly by the IDSs.

***Green et al.*** Green et al. [20] also adopt an approach based on the attacker's perspective: this approach consists of two practical examples in a *Man in the Middle* (MitM) scenario to obtain, correlate, and understand all the types of information an attacker might need to plan an attack to alter the process while avoiding detection.

The paper shows *step-by-step* how to perform a ICS **reconnaissance**, which is fundamental to process comprenension and thus to the execution of MiTM attacks.

***Ceccato et al.*** Ceccato et al. [6] propose a methodology based on a *black box dynamic analysis* of an ICS using a reverse engineering tool to derive from the scans performed on the memory registers of the exposed PLCs and network scans an approximate model of the physical process. This model is obtained by inferring statistical properties, business process and system invariants from data logs.

The proposed methodology was tested on a non-trivial case study, using a testbed inspired by an industrial water treatment plant.

In the next section I will examine this latest work in more detail, which will be the basis for my work and thus the subsequent chapters of this thesis.

## 3.2 Ceccato et al.'s methodology for analyzing water-tank systems

As mentioned earlier, the paper proposes a methodology based on a black box dynamic analysis of an ICS by identifying potential PLCs on the network and scanning the memory registers of the identified controllers to obtain an approximate model of the controlled physical process.

The first objective of this black box analysis is to associate the various memory registers of the target PLCs with a correspondence to the basic concepts of an ICS such as sensors (otherwise known as measurements), actuators, setpoints (range of values of a physical variable), network communications, and so on.
This is performed by analyzing the different types of memory registers associated with the Modbus protocol and trying to figure out what type of data they may contain.
The second objective is to put in relation the runtime evolution of these basic concepts.

To achieve this, Ceccato et al. developed a prototype tool [28] that performs reverse engineering of the physical system through four phases:

1. **scanning of the system and data pre-processing**: data gathering is performed to generate the data logs of PLCs registers
2. **graphs and statistical analysis**: provides information about the memory registers using graphs and statistical data derived from the gathered data
3. **invariants inference and analysis**: generates system invariants and allows user to view invariants related to a given sensor or actuator

4. **business process mining and analysis**: reconstructs, from event logs, the business process that shows how process is carried out

In Figure 3.1 we have a schematic representation of the workflow related to this work. We will cover all these phases in detail in the next sections of this chapter.

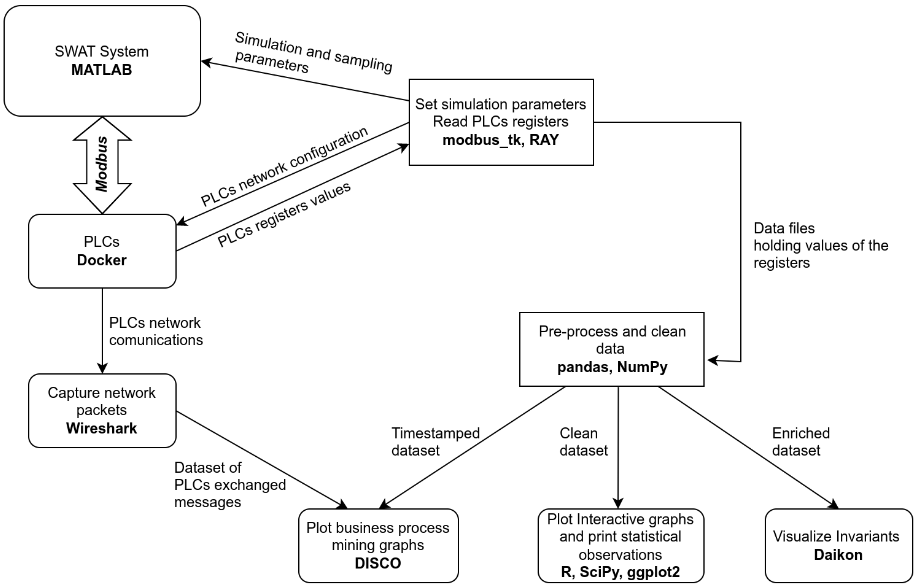

SWAT System
MATLAB
Simulation and sampling parameters
Set simulation parameters
Read PLCs registers
modbus_tk, RAY
Modbus
PLCs network configuration
PLCs registers values
PLCs
Docker
Data files holding values of the registers
PLCs network comunications
Pre-process and clean data
pandas, NumPy
Capture network packets
Wireshark
Timestamped dataset
Clean dataset
Enriched dataset
Dataset of PLCs exchanged messages
Plot business process mining graphs
DISCO
Plot Interactive graphs and print statistical observations
R, SciPy, ggplot2
Visualize Invariants
Daikon

***Figure 3.1:*** *Overview*

## 3.2.1 Testbed

Before describing the various phases of the methodology, let's take a look at the testbed on which this methodology will be tested. The testbed used to test this methodology is a (very) simplified version of the iTrust SWaT system [29] implemented by Lanotte et al. [30]: in Figure 3.2 we can see a graphical representation of the testbed. This simplified version consists of three stages, each controlled by a dedicated PLC:

***Stage 1*** At the first stage, a **tank** with a capacity of 80 gallons (identified by the code T-201) is filled with raw water by the P-101 pump: the

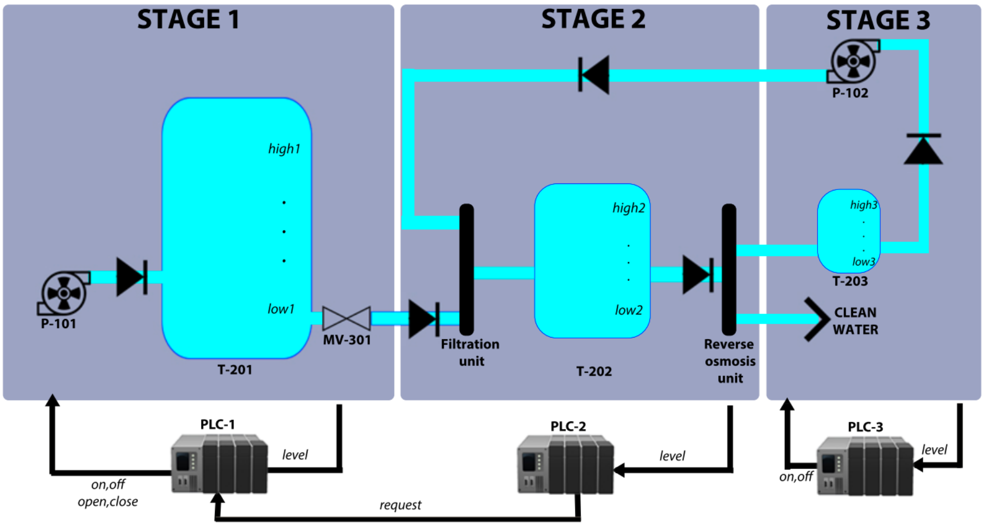

STAGE 1
STAGE 2
STAGE 3
P-102
high1
high2
high3
low3
T-203
P-101
low1
MV-301
Filtration unit
low2
Reverse osmosis unit
CLEAN WATER
T-201
T-202
PLC-1
PLC-2
PLC-3
level
level
level
on,off open,close
request
on,off

***Figure 3.2:*** *The simplified SWaT system used for running Ceccato et al. methodology*

MV-301 valve (where MV stands for *motorized valve*), also connected to the T-201 tank, flushes out the water collected in the tank to send it to the second stage, first to the *filtration unit* (here not identified by any sensor) and from there to a **second tank**, identified by the code T-202 and with a capacity of 20 gallons.

***Stage 2*** At the second stage, water contained in T-202 flows into the *reverse osmosit unit* (RO, which in this case also acts as a valve, extracting water continuously: however, it is not identified as a pump) to reduce organic impurities in the same water. The water then flows from the *RO unit* to the third and last stage.

***Stage 3*** At the third stage, the water from the *RO unit* is divided according to whether standards are met: if the water is clean it will be fed into the distribution system, otherwise it will go to a *backwash tank*, identified by code T-203 and a capacity of one gallon. The water in this tank will then be pumped back to the stage 2 *filtration unit* through pump P-102.

As mentioned, each stage corresponds to a PLC that controls it, PLC1, PLC2 and PLC3, respectively. Let us briefly see the behavior of each of them:

***PLC1*** PLC1 checks the level of tank T-201 distinguishing three cases:

- if T-201 reaches the *low setpoint low1* (hardcoded in memory registers), pump **P-101 is opened** and valve **MV-301 is closed**, so that the tank can be filled
- if T-201 reaches *high setpoint high1* (also hardcoded in the memory registers), pump **P-101 is closed**
- in intermediate cases, **PLC1 waits for request from PLC2** to open/close valve MV-301: if a request to open the valve MV-301 arrives, water will flow from T-201 to T-202, otherwise the valve is closed. In both situations, pump P-101 remains closed

***PLC2*** PLC2 monitors the level of tank T-202, behaving accordingly depending on the level of water in it. Here again there are three cases to consider:

- if the water level reaches the *low setpoint low2* (also hardcoded in the memory registers), PLC2 sends a request to PLC1 via a Modbus channel to **open valve MV-301** in order to flow water from tank T-201 to tank T-202. The transmission channel is implemented by copying a boolean value from a memory register of PLC2 to a corresponding register of PLC1
- if the water level reaches the *high setpoint high2* instead (hardcoded in the memory registers as the previous setpoints), PLC2 sends PLC1 a **close request** for valve MV-301
- In intermediate cases, the valve remains open (closed) while the tank is filling (emptying)

***PLC3*** PLC3 monitors the level of the T-203 backwash tank, behaving accordingly. Here there are only two cases to consider: if the tank reaches the *low setpoint low3*, pump **P103 is set to off**, so that the backwash tank can be filled: otherwise, if the *high setpoint high3* is reached, pump **P103 is opened** and the entire content of the backwash tank pumped back to the filter unit of T-202.

### 3.2.2 Scanning of the System and Data Pre-processing

**Scanning tool** The Ceccato et al. scanning tool is closely derived from a project I did [31] for the "*Network Security*" and "*Cyber Security for IoT*" courses taught by Professors Massimo Merro and Mariano Ceccato, respectively, in the 2020/21 academic year. The original project involved, in its first part, the recognition within a network of potential PLCs listening on the standard Modbus TCP port 502 using the Nmap module for Python, obtaining the corresponding IP addresses: then a (sequential) scan of a given range of the memory registers of the found PLCs was performed to collect the register data. The data thus collected were saved to a file in *JavaScript Object Notation* (JSON) format for later use in the second part of my project.

The scanning tool by Ceccato et. al works in a similar way, but extends what I originally did by trying to discover other ports on which the Modbus protocol might be listening (since in many realities Modbus runs on different ports than the standard one, according to the concept of *security by obscurity*) and, most importantly, by **parallelizing and distributing the scan** of PLC memory registers through the Ray module [32], specifying moreover the desired granularity of the capture. An example of raw data capture can be seen at Listing 3.1:

```
  "127.0.0.1/8502/2022-05-03 12_10_00.591": {
    "DiscreteInputRegisters": {"%IX0.0": "0"},
    "InputRegisters": {"%IW0": "53"},
    "HoldingOutputRegisters": {"%QW0": "0"},
    "MemoryRegisters": {"%MW0": "40","%MW1": "80"},
    "Coils": {"%QX0.0": "0"}}
```

***Listing 3.1:** Example of registers capture*

The captured data includes PLC's IP address, Modbus port and timestamp (first line), type and name of registers with their values read from the scan (subsequent lines).

The tool furthermore offers the possibility, in parallel to the memory registers scan, of **sniffing network traffic** related to the Modbus protocol using the *Man in the Middle* (MitM) technique on the supervisory control network using a Python wrapper for tshark/Wireshark [33] [34]. An example of raw data obtained with this sniffing can be seen in Listing 3.2:

```
  Time,Source,Destination,Protocol,Length,Function
   ↪ Code,Destination Port,Source Port,Data,Frame
   ↪ length on the wire,Bit Value,Request Frame,
   ↪ Reference Number,Info
  2022-05-03 11:43:58.158,IP_PLC1,IP_PLC2,Modbus/TCP
   ↪ ,76,Read Coils,46106,502,,76,TRUE,25,,"Response
   ↪ : Trans: 62; Unit: 1, Func: 1: Read Coils"
```

***Listing 3.2:** Example of raw network capture*

**Data Pre-processing** The data collected by scanning the memory registers of the PLCs are then reprocessed by a Python script and converted in order to create a distinct raw dataset in *Comma Separated Value* format (CSV) for each PLC, containing the memory register values associated with the corresponding controller registers. These datasets are reprocessed again through the Python modules for **pandas** [35] and **NumPy** [36] by another script to first perform a **data cleanup**, removing all those memory registers that do not take values and are therefore useless within the system, **merged** into a single dataset, and finally **enriched** with additional data[1].

---

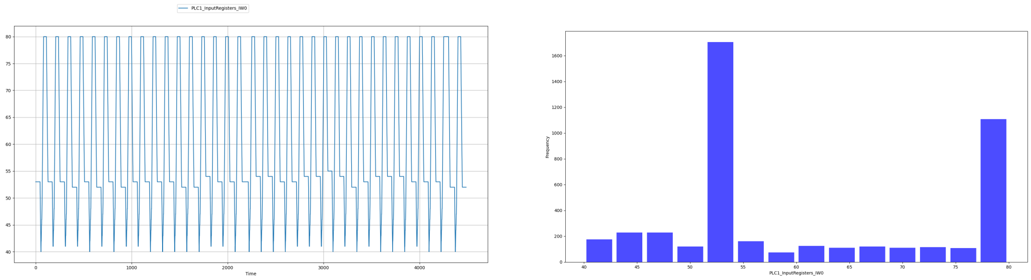
[1] Not all additional data are calculated and entered automatically by the tool: some are manually inserted.

This process leads to the creation of two copies of the full dataset: one enriched with the additional data, but not timestamped, which will be used for the invariant analysis; the other unenriched, but timestamped, which will be used for business process mining.

### 3.2.3 Graphs and Statistical Analysis

The paper mentions the presence of a *mild graph analysis*, performed with **R** [37] at the time of data gathering to find any uncovered patterns, trends and identify measurements and/or actuator commands through the analysis of registers holding mutable values.

There is actually no trace of this within the tool: *graph analysis* and *statistical analysis* of the data contained in the PLC memory registers are instead performed using the **matplotlib libraries** and statistical algorithms made available by the **SciPy libraries** [38], through two separate Python scripts (see Figure 3.3).



***(a)*** *Chart plot for* `PLC1_InputRegisters_IW01` *register*

***(b)*** *Histogram plot for* `PLC1_InputRegisters_IW01` *register*

***Figure 3.3:*** *Output graphs from graph analysis*

The first script plots the charts, one at the time, of certain registers entered by the user from the command line, plots in which one can see the trend of the data and get a first basic idea of what that particular register contains (a measurement, an actuation, a hardcoded setpoint, ...) and possibly the trend; the second script, instead, shows **a histogram and statistical informations** about the register entered as command-line input.

These informations include:

- the mean, median, standard deviation, maximum value and minimum value
- two tests for the statistical distribution: *Chi-squared* test for uniformity and *Shapiro-Wilk* test for normality, as shown in Listing 3.3:

```
Chi-squared test for uniformity
Distance      pvalue    Uniform?
12488.340   0.00000000    NO

Shapiro-Wilk test for normality
Test statistic    pvalue    Normal?
0.844   0.00000000    NO

Stats of PLC1_InputRegisters_IW0
Sample mean = 60.8881; Stddev = 13.0164; max = 80;
 ↪ min = 40 for 4488 values
```

***Listing 3.3:*** *Statistical data for* `PLC1_InputRegisters_IW0` *register*

### 3.2.4 Invariants Inference and Analysis

For invariant analysis Ceccato et al. rely on **Daikon** [39], a framework to **dynamically detect likely invariants** within a program. An *invariant* is a property that holds at one or more points in a program, properties that are not normally made explicit in the code, but within assert statements, documentation and formal specifications: invariants are useful in understanding the behavior of a program (in our case, of the cyber physical system).

Daikon uses *machine learning* techniques applied to arbitrary data with the possibility of setting custom conditions for analysis by using a specific file [40] with a *.spinfo* extension (see Listing 3.4). The framework is designed to find the invariants of a program, with various supported programming languages, starting from the direct execution of the program itself or passing as input the execution run (typically a file in CSV format): the authors of the paper tried to apply it by analogy also to the execution runs of a cyber physical system, to extract the invariants of this system.

```
PPT_NAME aprogram.point:::POINT
VAR1 > VAR2
VAR1 == VAR3 && VAR1 != VAR4
```

***Listing 3.4:*** *Generic example of a .spinfo file for customizing rules in Daikon*

Therefore, Daikon is fed with the no-timestamp enriched dataset obtained in the pre-processing phase (in the paper, the timestamped dataset is erroneously mentioned as input): a simple bash script launches Daikon (optionally specifying the desired condition for analysis in the *.spinfo* file), which output is simply redirected to a text file containing the general invariants of the system (i.e., valid regardless of any custom condition specified), those generated based on the custom condition in the *.spinfo* file, and those generated based on the negation of the condition. When the analysis is finished, the user is asked to enter the name of a registry to view its related invariants.

Some examples of invariants derived from the enriched dataset may be:

- measurements bounded by some setpoint
- Actuators state changes occourred in the proximity of setpoints or, vice versa, proximity of setpoints upon the occurrence of a regular actuator state change
- state invariants of some actuator correspond to a specific trend in the evolution of the measurement (ascending, descending, or stable) or, vice versa, the measurement trend corresponds to a specific state invariant of some actuator

### 3.2.5 Businness Process Mining and Analysis

*Process mining* is the analysis of operational processes based on the event log [41]: the aim of this analysis is to **extract useful informations** from the event data to **reconstruct and understand the behavior** of the business process and how it was actually performed.

Process mining for the system under consideration starts from the event logs obtained from scanning the memory registers of the PLCs and sniffing the network communications related to the Modbus protocol, described in Subsection 3.2.2 and representing the *execution trace* of the system: through a Java program, information is extracted and combined from these event logs, and the result saved in a CSV format file.

This file is fed to **Disco** [42], a commercial process mining tool, which generates an *activity diagram* similar to UML Activity Diagram and whose nodes represent the activities while the edges represent the relations between these activities: in Figure 3.4 we can see an example of this diagram referred to PLC2 of the testbed.

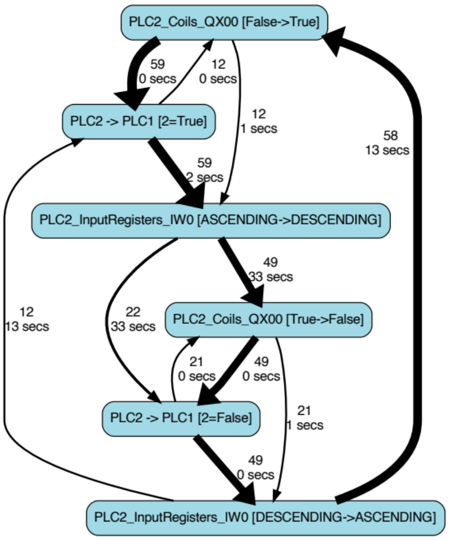

PLC2_Coils_QX00 [False->True]
59
0 secs
12
0 secs
PLC2 -> PLC1 [2=True]
12
1 secs
58
13 secs
59
2 secs
PLC2_InputRegisters_IW0 [ASCENDING->DESCENDING]
49
33 secs
12
13 secs
22
33 secs
PLC2_Coils_QX00 [True->False]
21
0 secs
49
0 secs
PLC2 -> PLC1 [2=False]
21
1 secs
49
0 secs
PLC2_InputRegisters_IW0 [DESCENDING->ASCENDING]

***Figure 3.4:*** *An example of Disco generated activity diagram for PLC2*

The *business process* obtained in this way provides an **overview of the system** and makes it possible to **make conjectures** about its behavior, particularly between changes in actuator state and measurement trends (i.e., a given change in state of some actuators corresponds to a specific measurement trend and vice versa), and with the possibility of **establishing causality** between Modbus communications and state changes within the physical system.

### 3.2.6 Application

In this section we will see how the black box analysis presented above in its various phases is applied in practice, using the testbed described in Subsection 3.2.1. The methodology supports a ***top-down* approach**: that is, we start with an overview of the industrial process and then gradually refine our understanding of the process by descending to a higher and higher level of detail based on the results of the previous analyses and focusing on the most interesting parts of the system for further in-depth analysis.

**Data Collection and Pre-processing** According to what is described in the paper, the data gathering process lasted six hours, with a granularity of one data point per second (a full system cycle takes approximately 30 minutes). Each datapoint consists of 168 attributes (55 registers plus a special register concerning the tank slope of each PLC) after the enrichment. In addition, IP addresses are automatically replaced by an abstract name identified by the prefix PLC followed by a progressive integer (PLC1, PLC2, PLC3), in order to make reading easier.

**Graphs and Statistical Analysis** It is unclear from the paper where exactly the information that follows was derived (graph analysis? Statistical analysis? Human reading of the dataset?), however, three properties about the contents of the registers were discovered:

*Property 1:* `PLC1_MemoryRegisters_MW0`, `PLC1_MemoryRegisters_MW1`,
`PLC2_MemoryRegisters_MW0`, `PLC2_MemoryRegisters_MW1`,
`PLC3_MemoryRegisters_MW0` and `PLC3_MemoryRegisters_MW1`

registers contain constant integer values (40, 80, 10, 20, 0, 10 respectively)[2]. We may speculate that they may be (relative) hardcoded **setpoints**.

*Property 2:* `PLC1_Coils_QX01`, `PLC1_Coils_QX02`, `PLC2_Coils_QX01`, `PLC2_Coils_QX02`, `PLC3_Coils_QX01` and `PLC3_Coils_QX03` contain mutable binary (Boolean) values. We can assume that these registers can be associated with the **actuators** of the system.

*Property 3:* `PLC1_InputRegisters_IW0`, `PLC2_InputRegisters_IW0` and `PLC3_InputRegisters_IW0` registers contain mutable values.

*Property 3* suggests that those registers might contain **values related to measurements**: it is therefore necessary to investigate further to see if the conjecture (referred to as *Conjecture 1* in the paper) is correct.

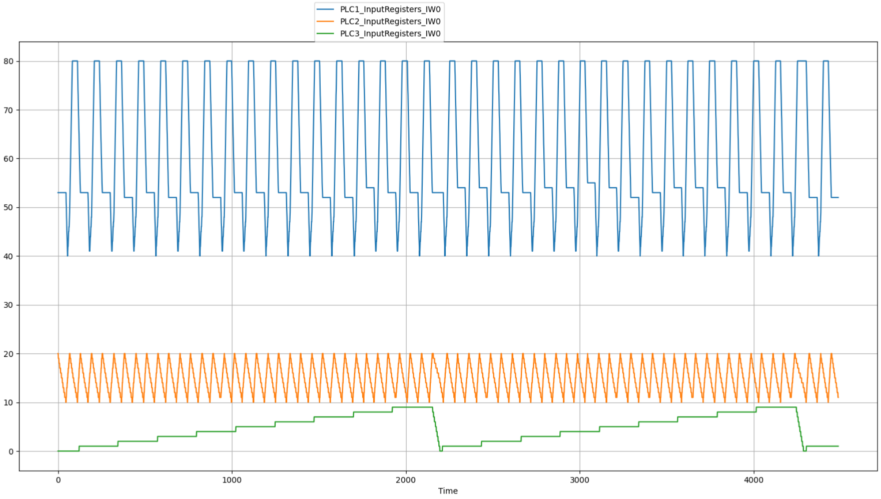


***Figure 3.5:*** *Execution traces of InputRegisters_IW0 on the three PLCs*

The graph analysis of the `InputRegisters_IW0` registers of the three PLCs (summarized in Figure 3.5 with a single plot) not only seems to confirm the conjecture, but also allows the measurements to be correlated with

---

[2] From my tests on the original tool and dataset, the `PLC3_MemoryRegisters_MW0` register is deleted during the *pre-processing* phase, as it is recognized as an unused register because of the constant value "0" it takes on. This leads me to assume that the properties are derived from a human read of the dataset prior to the *pre-processing* phase.

the contents of the `MemoryRegisters_MW0` and `MemoryRegisters_MW1` registers to the measurements, which represent the **relative setpoints of the measurements**.
Hence, we have *Conjecture 2* described in the paper referring to the relative setpoints:

*Conjecture 2:*
- 40 and 80 are the relative setpoints for `PLC1_InputRegisters_IW0`
- 10 and 20 are the relative setpoints for `PLC2_InputRegisters_IW0`
- 0 and 9 are the relative setpoints for `PLC3_InputRegisters_IW0`

Further confirmation of this conjecture may come from statistical analysis. Indeed, in the example in Listing 3.1, some statistical data are given for the register `PLC1_InputRegisters_IW0`, including the maximum value and the minimum value: these values are, in fact, 80 and 40 respectively.

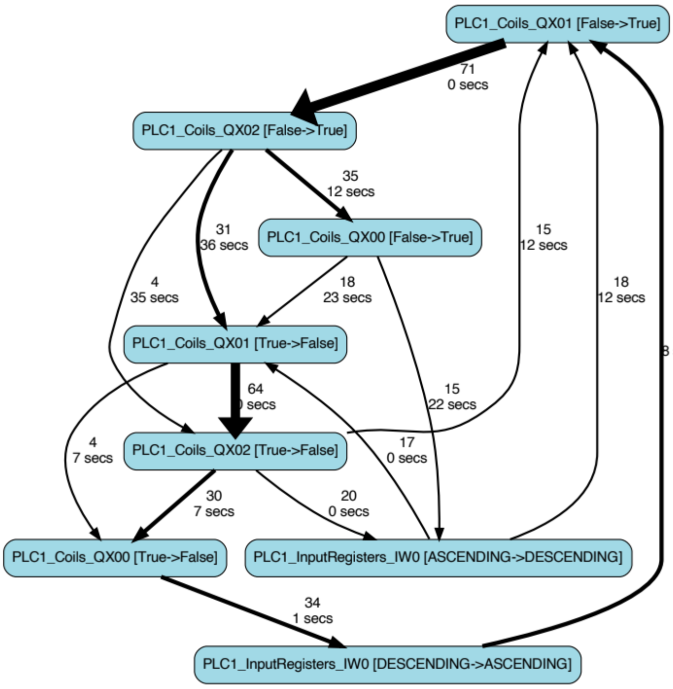
**Business Process Mining and Analysis** With Business Process Mining, the authors aim to **visualize and highlight relevant system behaviors** by relating PLC states and Modbus commands.

Through analysis of the activity diagrams shown in Figure 3.6, drawn through Disco, we derive the following properties and conjectures:

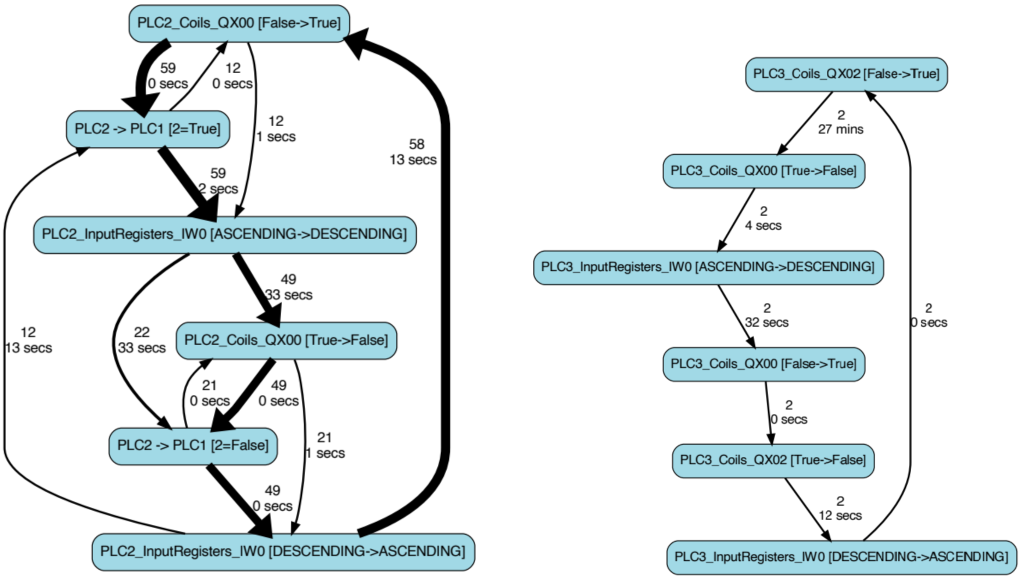
*Property 4:* PLC2 sends messages to PLC1 (see Figure 3.6b) which are recorded to `PLC1_Coils_QX02`.

*Conjecture 3:* `PLC2_Coils_QX00` determines the trend in tank T-202 (Figure 3.6b).
When this register is set to *True*, the input register `PLC2_InputRegisters_IW0` related to the tank controlled by PLC2 starts an **ascending trend**; vice versa, when the coil register is set to *False*, the input register starts a **descending trend**.

*Conjecture 4:* If `PLC1_Coils_QX00` change his value to True, trend in tank T-201, related to `PLC1_InputRegisters_IW0` and controlled by PLC1, become **ascending** (see Figure 3.6a)


PLC1_Coils_QX01 [False->True]
71
0 secs
PLC1_Coils_QX02 [False->True]
35
12 secs
31
36 secs
PLC1_Coils_QX00 [False->True]
15
12 secs
4
35 secs
18
23 secs
18
12 secs
34
8 secs
PLC1_Coils_QX01 [True->False]
64
secs
15
22 secs
4
7 secs
PLC1_Coils_QX02 [True->False]
17
0 secs
30
7 secs
20
0 secs
PLC1_Coils_QX00 [True->False]
PLC1_InputRegisters_IW0 [ASCENDING->DESCENDING]
34
1 secs
PLC1_InputRegisters_IW0 [DESCENDING->ASCENDING]

*(a) States in PLC1*


PLC2_Coils_QX00 [False->True]
59
0 secs
12
0 secs
PLC2 -> PLC1 [2=True]
12
1 secs
58
13 secs
59
2 secs
PLC2_InputRegisters_IW0 [ASCENDING->DESCENDING]
49
33 secs
12
13 secs
22
33 secs
PLC2_Coils_QX00 [True->False]
21
0 secs
49
0 secs
PLC2 -> PLC1 [2=False]
21
1 secs
49
0 secs
PLC2_InputRegisters_IW0 [DESCENDING->ASCENDING]
PLC3_Coils_QX02 [False->True]
2
27 mins
PLC3_Coils_QX00 [True->False]
2
4 secs
PLC3_InputRegisters_IW0 [ASCENDING->DESCENDING]
2
32 secs
2
0 secs
PLC3_Coils_QX00 [False->True]
2
0 secs
PLC3_Coils_QX02 [True->False]
2
12 secs
PLC3_InputRegisters_IW0 [DESCENDING->ASCENDING]

***(b)** States and Modbus command in PLC2*

***(c)** States in PLC3*

***Figure 3.6:** Business process with states and Modbus commands for the three PLCs*

*Conjecture 5:* `PLC3_Coils_QX00` starts a **decreasing trend** in tank T-203, related to `PLC3_InputRegisters_IW0` and controlled by PLC3, whereas `PLC3_Coils_QX02` starts an **increasing trend** on the tank (see Figure 3.6c)

**Invariants Inference and Analysis** The last phase of the analysis of the example industrial system is invariant analysis, performed through Daikon framework. At this stage, an attempt will be made to confirm what has been seen previously and to derive new properties of the system based on the results of the Daikon analysis.

To get gradually more and more accurate results, the authors presumably performed more than one analysis with Daikon, including certain rules within the *splitter information file* (see Section 3.2.4 and Listing 3.4) based on specific conditions placed on the measurements, for example, the level of water contained in a tank. Given moreover the massive amount of invariants generated by Daikon's output, it is not easy to identify and correlate those that are actually useful for analysis: this must be done manually.

However, it was possible to have confirmation of the conjectures made in the previous stages of the analysis: starting with the setpoints, analyzing the output of the invariants returned by Daikon[3] reveals that

```
PLC1_InputRegisters_IW0 >= PLC1_MemoryRegisters_MW0 == 40.0
PLC1_InputRegisters_IW0 <= PLC1_MemoryRegisters_MW1 == 80.0
PLC2_InputRegisters_IW0 >= PLC2_MemoryRegisters_MW0 == 10.0
PLC2_InputRegisters_IW0 <= PLC2_MemoryRegisters_MW1 == 20.0
PLC3_InputRegisters_IW0 >= PLC3_MemoryRegisters_MW0 == 0.0
PLC3_InputRegisters_IW0 <= PLC3_MemoryRegisters_MW1 == 9.0
```

i.e., that the `MemoryRegisters_MW0` and `MemoryRegisters_MW1` registers of each PLC contain the **absolute minimum and maximum setpoints**, respectively (*Property 5*).

---

[3] The invariants shown here are a manual summary and derivation of those actually returned in output by Daikon. I will discuss this more in Section 3.2.7

There is also a confirmation regarding *Property 4*: from the computed invariants it can be seen that

```
PLC1_Coils_QX01 == PLC1_Coils_QX02 == PLC2_Coils_QX00
```

and from this derive that there is a **communication channel between PLC2 and PLC1**, where the value of `PLC2_Coils_QX00` is copied to `PLC1_Coils_QX01` and `PLC1_Coils_QX02` (*Property 6*).

Regarding the **relationships between actuator state changes and measurement trends**, invariant analysis yields the results summarized in the following rules:

*Property 7:* Tank T-202 level *increases* iif `PLC1_Coils_QX01 == True`. Otherwise, if `PLC1_Coils_QX01 == False` will be *non-increasing*.

This is because if the coil is *True* the condition
`PLC2_InputRegisters_IW0 == PLC2_MemoryRegisters_MW0 == 20.0 && PLC2_slope > 0`
is verified. On the opposite hand, if the coil is *False*, the condition
`PLC2_InputRegisters_IW0 == PLC2_MemoryRegisters_MW0 == 20.0 && PLC2_slope <= 0` is verified. The *slope* is an auxiliary attribute indicating the trend of the measurement: increasing if > 0, decreasing if < 0, stable otherwise.

*Property 8:* Tank T-201 level *increases* iif `PLC1_Coils_QX00 == True`. On the other hand, if `PLC1_Coils_QX00 == False` and if `PLC1_Coils_QX01 == True` the level will be *non-decreasing*.

*Property 9:* Tank T-203 level *decreases* iif `PLC3_Coils_QX00 == True`. It will be *non-decreasing* if `PLC1_Coils_QX00 == False`.

The last two properties concern the **relationship between actuator state changes and the setpoints**: it is intended to check what happens to the actuators when the water level reaches one of these setpoints. From the analysis of the relevant invariants, the following properties are derived:

*Property 10:* Tank T-201 reaches the upper absolute setpoint when `PLC1_Coils_QX00` changes its state from *True* to *False*. If the coil changes from *False* to *True*, the tank reaches its absolute lower setpoint.

*Property 11:* Tank T-203 reaches the upper absolute setpoint when `PLC3_Coils_QX00` changes its state from *True* to *False*. If the coil changes from *False* to *True*, the tank reaches its absolute lower setpoint.

### 3.2.7 Limitations

The methodology proposed by Ceccato et al. is certainly valid and offers a good starting point for approaching the reverse engineering of an industrial control system from the attacker's perspective, while also providing a tool to perform this task.

The limitations of this approach, however, all lie in the tool mentioned above and also in the testbed described in Section 3.2.1. In this section I will explain which are the criticisms of each phase, while in Chapter 4 I will formulate proposals to improve and make this methodology more efficient.

**General Criticism** The general critical aspects of the application of this approach are many: the primary one concerns the fact that the proposed tool seems to be built specifically for the testbed used and that it is not applicable to other contexts, even to the same type of industrial control system (water treatment systems, in this case).

What severely limits the analysis performed with the tool implemented by Ceccato et al. is the use of *ad hoc* solutions and *a posteriori* interventions done manually on the datasets after the data gathering process: I will discuss this last aspect in more detail later.
Moreover, there is the presence of many *hardcoded* variables and conditions within the scripts: this makes the system unconfigurable and unable to properly perform the various stages of the analysis as errors can occur due to incorrect data and mismatches with the system under analysis.

Having considered, furthermore, only the Modbus protocol for network communications between the PLCs is another major limiting factor and does not help the methodology to be adaptable to different systems communicating with different protocols (sometimes even multiple ones on the same system).

Let us now look at the limitations and critical aspects of each phase.

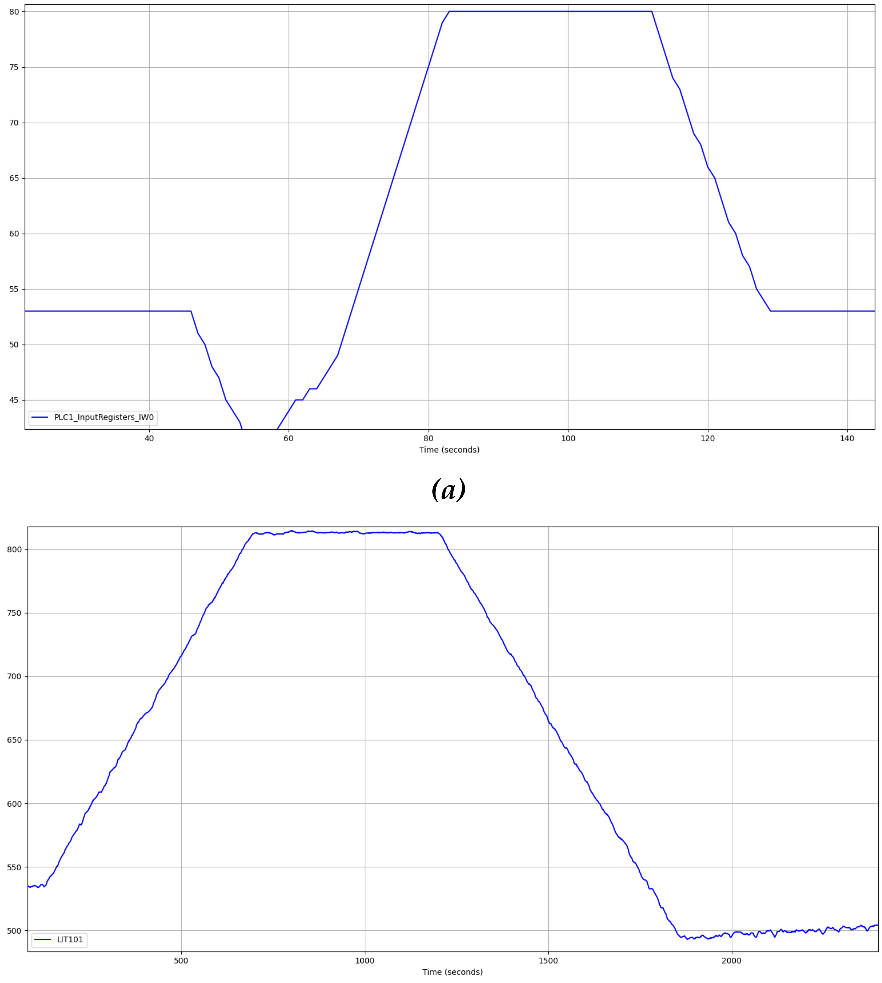
**Testbed** The testbed environment used by Ceccato et. al is entirely simulated, from the physical system to the control system. The PLCs were built with **OpenPLC** [43] in a Docker environment [44], while the physics part was built through **Simulink** [45].

OpenPLC is an open source cross-platform software that simulates the hardware and software functionality of a physical PLC and also offers a complete editor for PLC program development with support for all standard languages: *Ladder Logic* (LD), *Function Block Diagram* (FBD), *Instruction List* (IL), *Structured Text* (ST), and *Sequential Function Chart* (SFC).
It is for sure an excellent choice for creating a zero-cost industrial or home automation and *Internet of Things* (IoT) system that is easy to manage via a dedicated, comprehensive and functional web interface. In spite of these undoubted merits, however, there are (at the moment) **very few supported protocols**: the main one and also referred to in the official documentation is **Modbus**, while the other protocol is DNP3.

The biggest problem with the testbed, however, is not with the controller part, but with the **physical part**: first of all, it must be said that although this is something purely demonstrative even though it is fully functional, the implemented Simulink model is really **oversimplified** compared to the iTrust SWaT system, which itself is a scaled-down version of a real water treatment plant. In fact, in the entire system there are only three actuators, two of which are connected to the same tank and controlled by the same PLC, and sensors related only to the water level in the system's tanks: in a real system there are many more *field devices*, which can mon-

itor and control other aspects of the system beyond the mere contents of the tanks. Consider, for example, measuring and controlling the chemicals in the water, the pressure of the liquid in the filter unit, or more simply the amount of water flow at a given point or time.
All these must be considered and represent a number of additional variables that makes analysis and consequently reverse engineering of the system more difficult.

The second critical aspect concerns the **simulation of the physics of the liquid** inside the tanks: Simulink does not consider the fact that inside a tank that is filling (emptying) the liquid in it undergoes **fluctuations** which cause the level sensor not to see the water level constantly increasing (decreasing) or at most being stable at each point of detection. Figure 3.7 exemplifies more clearly with an example the concept just expressed: these oscillations cause a **perturbation** in the data.
This issue leads to the difficulty, on a real physical system, of **correctly calculating the trend of a measurement** by using the slope attribute: if this was obtained with a too low granularity, the trend will be oscillating between increasing and decreasing even when in reality this would be in general increasing (decreasing) or stable; on the other hand, if the slope was obtained with a too high granularity there is a loss of information and the trend may be "flattened" with respect to reality.
In the present case, the slope in the Simulink model was calculated statically *point-to-point*, thus with a granularity of one second: an averagely careful reader will have already guessed that this granularity is inapplicable to the real system in Figure 3.7b. As we will later see, we need to **operate on the data perturbations** to be able to obtain a suitable granularity and a correct calculation of the slope and consequently of the measurement trend.



*(a)*

*(b)*

***Figure 3.7:*** *Water physics compared: simulated physics in the Simulink model (a) and physics in a real system (iTrust SWaT) (b). Fluctuations in the tank level in (b), almost completely absent in (a), can be appreciated.*

**Pre-processing** In the pre-processing phase, the authors make use of a Python script to merge all the datasets of the individual PLCs into a single dataset, remove the (supposedly) unused registers, and finally enrich the obtained dataset with additional attributes. These attributes are:

- the previous value of all registers

- some additional relative setpoints named `PLC`$x$`_Max_safety` and `PLC`$x$`_Min_safety` (where $x$ is the PLC number), which represent a kind of alert on reaching the maximum and minimum water levels of the tanks

- the measurement slope, that is, the trend of the water level in the system tanks along the system cycles.

Merging the datasets of all individual PLCs into a single dataset to represent the entire system can be a sound practice if the system to be anlized is (very) small as is the testbed analyzed here, consisting of a few PLCs and especially a few registers. If, however, the complexity of the system increases, this type of merging can become counterproductive and make it difficult to analyze and understand the data obtained in subsequent steps. In short, there is no possibility to analyze only one of its subsystems and thus make the analysis faster and more understandable. Moreover, a data gathering can take up to days, and the analyst/attacker may need to make an analysis of the system isolating a precise time range, ignoring everything that happens before and/or after: all of this, with the tool we have seen, cannot be done.

Regarding the additional attributes, looking at the code of the script that performs the enrichment, I observed that **some attributes were manually inserted** after the merging phase: I am referring in particular to the attributes `PLC`$x$`_Max_safety` and `PLC`$x$`_Min_safety`, whose references were moreover hardcoded into the script, and the *slope* whose calculation method I mentioned in the previous paragraph about the testbed limitations.
In the end, only the attribute *prev* related to the value at the previous point of the detection is inserted automatically for all registers, moreover without the possibility to choose whether this attribute should be extended to all registers or only to a part.

**Graphs and Statistical Analysis** Describing the behavior of graphical analysis in Section 3.2.3 I had already mentioned that only one register plot

at a time was shown and not, for example, a single window containing the charts of all registers entered by the user as input from the command line, such as in Figure 3.5.
While displaying charts for individual registers still provides useful information about the system such as the distinction between actuators and measurements and the general trend of the latter, single display does not allow one to catch, or at least makes it difficult, the relationship that exists between actuators and measurements, where it exists, because a view of the system as a whole is missing.
In this way, the risk is to make conjectures about the behavior of the system that may prove to be at least imprecises, if not inaccurates.

On the other hand, regarding the statistical analysis, two observations need to be made: the first is that for the given system, I personally was unable to appreciate the usefulness of the generated histogram, as it does not provide any particular new information that has not already been obtained from the graphical analysis (except maybe something marginal); the second observation is that precisely the plot of the histogram "hides" the statistical informations obtained: these are in fact shown on the terminal from which the script is launched, but to an uncareful eye or one unfamiliar with the script's behavior they can easily be interpreted as simple debugging output, since at the same time the window containing the histogram plot is shown. In general, however, little statistical information is provided.

**Business Process Mining and Analysis** Concerning the data mining, this is a purely ad *hoc solution*, designed to work under special conditions: first, the timestamped dataset of the physical process and the one obtained after the packet sniffing operation of Modbus traffic on the network need to be synchronized and have the same granularity, in this case one event per second.
It is relatively easy, therefore, to find correspondences between Modbus commands sent over the network and events occurring on the physical

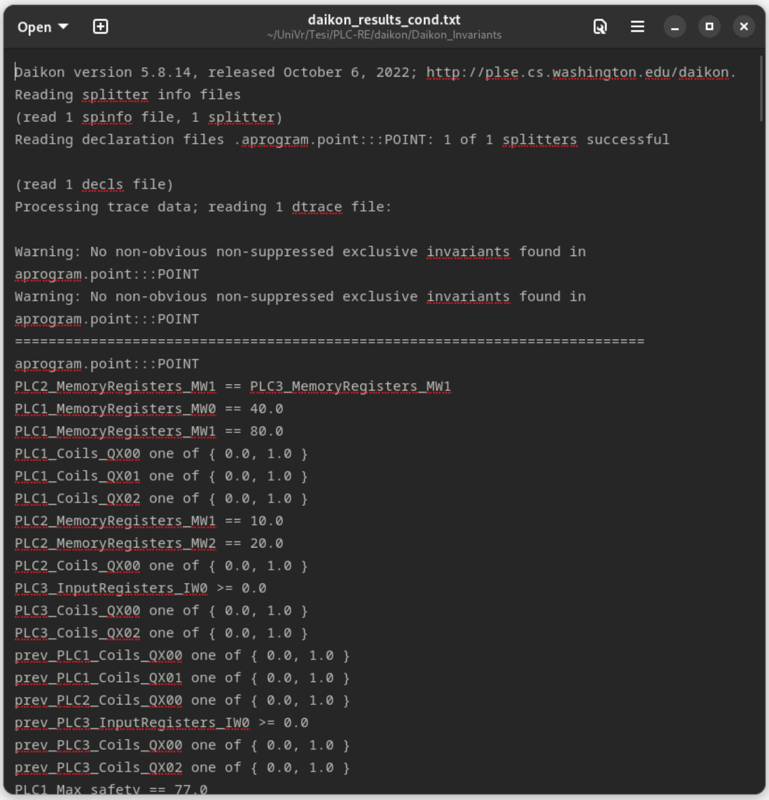
system, such as state changes in actuations, due in part to the fact that the number of communications over the network is really small (see Section 3.2.1). In a real system, network communications are much more numerous and involve many more devices even in the same second: finding the exact correspondence with what is happening in the cyber physical system becomes much more difficult.

Since this is, as mentioned, an *ad hoc* solution, only the Modbus protocol is being considered: as widely used as this industrial protocol is, other protocols that are widely used such as EtherNet/IP (see Section 2.3.1.2) should be considered in order to extend the analysis to other industrial systems that use a different communication network.

The other limiting aspect of the business process mining phase is the **process mining software** used to generate the activity diagram. As mentioned in Section 3.2.5, the process mining software used by Ceccato et al. is **Disco**: this is commercial software, with an academic license lasting only 30 days (although free of charge), released for Windows and MacOS operating systems only, which makes its use under Linux systems impossible except by using emulation environments such as Wine.
For what is my vision and training as a computer scientist, it would have been preferable to use a *cross-platform*, *freely licensed open source* software alternative to Disco: one such software could have been **ProM Tools** [46], a framework for process mining very similar to Disco in functionality, but fitting the criteria just described, or use Python libraries such as **PM4PY** [47], which offer ready-to-use algorithms suitable for various process mining needs.

**Invariants Inference and Analysis** The limitation in this case is principally Daikon: this software is designed to compute the invariants of a software from its live execution or from a file containing its execution flow, not to find the invariants of a cyber physical system. Since there are currently no better consolidated alternatives for inferring invariants, however, an attempt was still made to use Daikon as best as possible.

The biggest problem with Daikon applied to the computation of invariants of an industrial system is the difficult reading of the resulting output: the software in fact returns a very long list of invariants, one invariant per line, many of no use and without correlating invariants that may have common features or deriving additional information from them. The process of screening and recognizing the significant invariants, as well as the correlation between them, must be done by a human: certainly not an easy task given the volume of invariants one could theoretically be faced with (hundreds and hundreds of invariants). An example of Daikon's output can be seen in Figure 3.8.



```
Daikon version 5.8.14, released October 6, 2022; http://plse.cs.washington.edu/daikon.
Reading splitter info files
(read 1 spinfo file, 1 splitter)
Reading declaration files .aprogram.point:::POINT: 1 of 1 splitters successful

(read 1 decls file)
Processing trace data; reading 1 dtrace file:

Warning: No non-obvious non-suppressed exclusive invariants found in
aprogram.point:::POINT
Warning: No non-obvious non-suppressed exclusive invariants found in
aprogram.point:::POINT
===========================================================================
aprogram.point:::POINT
PLC2_MemoryRegisters_MW1 == PLC3_MemoryRegisters_MW1
PLC1_MemoryRegisters_MW0 == 40.0
PLC1_MemoryRegisters_MW1 == 80.0
PLC1_Coils_QX00 one of { 0.0, 1.0 }
PLC1_Coils_QX01 one of { 0.0, 1.0 }
PLC1_Coils_QX02 one of { 0.0, 1.0 }
PLC2_MemoryRegisters_MW1 == 10.0
PLC2_MemoryRegisters_MW2 == 20.0
PLC2_Coils_QX00 one of { 0.0, 1.0 }
PLC3_InputRegisters_IW0 >= 0.0
PLC3_Coils_QX00 one of { 0.0, 1.0 }
PLC3_Coils_QX02 one of { 0.0, 1.0 }
prev_PLC1_Coils_QX00 one of { 0.0, 1.0 }
prev_PLC1_Coils_QX01 one of { 0.0, 1.0 }
prev_PLC2_Coils_QX00 one of { 0.0, 1.0 }
prev_PLC3_InputRegisters_IW0 >= 0.0
prev_PLC3_Coils_QX00 one of { 0.0, 1.0 }
prev_PLC3_Coils_QX02 one of { 0.0, 1.0 }
PLC1_Max_safety == 77.0
```

*Figure 3.8: Example of Daikon's output*

The bash script used in this phase of the analysis does not help at all in deriving significant invariants more easily: it merely launches Daikon and saves its output to a text file by simply redirecting the stdout to file. No data reprocessing is done during this step. In addition, if a condition is to be specified to Daikon before performing the analysis, it is necessary each time to edit the .spinfo file by manually entering the desired rule, an inconvenient operation when multiple analyses are to be performed with different conditions each time.

Chapter 4

# A framework to improve Ceccato et al.'s work.

In Chapter 3, I presented the state of the art of *process comprehension* of an Industrial Control System (ICS) focusing later on the methodology proposed by Ceccato et al. [6][Section 3.2], explaining what it consists of, its practical application on a testbed, and most importantly highlighting its limitations and critical issues (see Section 3.2.7).

In this chapter I will present **my proposals to improve the methodology** presented in the previous chapter, overcoming (or at least trying to do so) the criticalities mentioned above by almost completely rewriting the original framework, enhancing its functionalities and inserting new ones where possible, while keeping its general structure and approach: the system analysis will in fact consist of the same four steps as in the original methodology (Data Pre-processing, Graph and Statistical Analysis, Business Process Mining and Invariants Inference), but each of them will be deeply revised in order to provide a richer, clearer and more complete process comprehension of the industrial system to be analyzed and its behavior.

As it may have already been noted, my proposals do not involve improving the data gathering phase: this is due simply to the fact that the novel framework will not be tested on the same case study used by Ceccato et, al. (Section 3.2.1), but on a different case study, the ITrust SWaT system [29], of which (some) datasets containing the execution trace of the physical system and the network traffic scan are already provided by iTrust itself. For more details about this case study, see Chapter 5.

## 4.1 Phase 0: General Improvements

The implementation of the novel framework for ICSs analysis starts from several assumptions:

1. it must be implemented in a **single programming language**
2. it must be **independent of the system** to be analyzed
3. It must provide greater **flexibility and ease of use** for the user at every stage

In the following subsections, these three points will be discussed in more detail.

### 4.1.1 Single Programming Language

The original tool was implemented using various programming languages in each of the different phases: from Python up to Java, passing through Bash scripting.
In my opinion, this heterogeneity makes it more difficult and less intuitive for the user to operate on the tool: moreover, the use of multiple technologies makes it more difficult to maintain the code and add new features, particularly if only a single person is managing the code (he/she might be proficient in one language, but little of the others).

For these reasons, I decided to use a single programming language, to ensure homogeneity to the framework and ease of use and maintenance of the code for anyone who wants to manage it in the future: I chose to use Python, because of its simplicity and easy readability combined with its versatility and powerfulness: moreover, Python can count on a massive number of available libraries and packages that meet all kinds of needs.

### 4.1.2 System Independence

One of the biggest limitations of Ceccato et al.’s tool that I highlighted in Section 3.2.7 is the fact that it is **highly dependent on the testbed used**: that is, it is *not* possible to configure any of the tool’s parameters to analyze different industrial systems.
To overcome this issue and make my framework independent of the system to be analyzed, also eliminating all references to hardcoded variables and values present in the previous tool, I decided to use a **general configuration file**, named *config.ini*, in which the user can, at will, customize all the parameters necessary to perform the analysis of the targeted system.

**General Configuration File *config.ini*** The configuration file *config.ini* for the framework I am presenting in this thesis is intended, as mentioned, to make it customizable in order to fit a variety of different systems and allow for their analysis. Here the user can configure general parameters and options, such as paths to read from or write files to, or related to individual analysis phases.
The file is divided into sections, each covering a different aspect of the configuration: each section contains user-customizable constants that will then be called within the Python scripts that constitute the framework. An example can be appreciated in Listing 4.1:

```
[PATHS]
root_dir = /home/marcuzzo/UniVr/Tesi
project_dir = %(root_dir)s/PLC-RE
input_dataset_directory = %(root_dir)s/
 ↪ datasets_SWaT/2015
net_csv_path = %(root_dir)s/datasets_SWaT/2015/
 ↪ Network_CSV

[DEFAULTS]
dataset_file = PLC_SWaT_Dataset.csv
granularity = 10
```

```
number_of_rows = 20000
skip_rows = 100000

[DAIKON]
daikon_dir = daikon
daikon_invariants_dir = %(daikon_dir)s/
 ↪ Daikon_Invariants
daikon_results_dir = %(daikon_invariants_dir)s/
 ↪ results
daikon_results_file_original = daikon_results_full.
 ↪ txt
inv_conditions_file = Inv_conditions.spinfo
max_security_pct_margin = 1
min_security_pct_margin = 2

...
```

***Listing 4.1:*** *Example of config.ini file*

### 4.1.3 Flexibility and Ease on Use

## 4.2 Phase 1: Data Pre-processing

## 4.3 Phase 2: Graphs and Statistical Analysis

## 4.4 Phase 3: Invariants Analysis

### 4.4.1 Invariants Generation

## 4.5 Phase 4: Businness Process Analysis

## 4.6 Extra Information on the Physics

## Chapter 5

# Case study: the iTrust SWaT System

Chapter 6

# Our framework at work: reverse engineering of the SWaT system

## 6.1 Pre-processing

## 6.2 Graph Analysis

### 6.2.1 Conjectures About the System

## 6.3 Invariants Analysis

### 6.3.1 Actuators Detection

### 6.3.2 Daikon Analysis and Results Comparing

## 6.4 Extra information on the Physics

## 6.5 Businness Process Analysis

Chapter 7

# Conclusions

## 7.1 Discussions

## 7.2 Guidelines

## 7.3 Future work

# List of Figures

2.1 SCADA architecture schema . . . . . . . . . . . . . . . . . . . . 6
2.2 PLC architecture . . . . . . . . . . . . . . . . . . . . . . . . . 8
2.3 PLC communication schema . . . . . . . . . . . . . . . . . . . . 9
2.4 Comparison between ST language and Ladder Logic . . . . 10
2.5 Modbus Request/Response schema . . . . . . . . . . . . . . . 14
2.6 Modbus RTU frame and Modbus TCP frame . . . . . . . . . 15
2.7 Example of packet sniffing on the Modbus protocol . . . . . 17
2.8 OSI model for EtherNet/IP stack . . . . . . . . . . . . . . . . 18

3.1 Overview . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 25
3.2 The simplified SWaT system used for running Ceccato et al. methodology . . . . . . . . . . . . . . . . . . . . . . . . . . . 26
3.3 Output graphs from graph analysis . . . . . . . . . . . . . . . 30
3.4 An example of Disco generated activity diagram for PLC2 . 33
3.5 Execution traces of InputRegisters_IW0 on the three PLCs . 35
3.6 Business process with states and Modbus commands for the three PLCs . . . . . . . . . . . . . . . . . . . . . . . . . . . . 37
3.7 Water physics compared: simulated physics in the Simulink model (a) and physics in a real system (iTrust SWaT) (b). Fluctuations in the tank level in (b), almost completely absent in (a), can be appreciated. . . . . . . . . . . . . . . . . . . . 43
3.8 Example of Daikon's output . . . . . . . . . . . . . . . . . . . 47

# List of Tables

2.1 Modbus Function Codes list . . . . . . . . . . . . . . . . . . . . . 16

# References

[1] *ICS defintion.* URL: `https://csrc.nist.gov/glossary/term/industrial_control_system` (visited on 2023-04-06).

[2] *What is SCADA?* URL: `https://www.inductiveautomation.com/resources/article/what-is-scada` (visited on 2023-04-05).

[3] G. M. Makrakis, C. Kolias, G. Kambourakis, C. Rieger, and J. Benjamin. "Industrial and Critical Infrastructure Security: Technical Analysis of Real-Life Security Incidents". In: *IEEE Access* (2021-12-06). DOI: `https://dx.doi.org/10.1109/ACCESS.2021.3133348`. URL: `https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=9638617` (visited on 2023-04-20).

[4] *PLC defintion.* URL: `https://csrc.nist.gov/glossary/term/programmable_logic_controller` (visited on 2023-04-06).

[5] W. Bolton. *Programmable Logic Controllers, 6th edition.* Newnes, 2015, pp. 7–9.

[6] M. Ceccato, Y. Driouich, R. Lanotte, M. Lucchese, and M. Merro. "Towards Reverse Engineering of Industrial Physical Processes". In: CPS4CIP@ESORICSAt 2022 (Copenhagen, Denmark, Sept. 30, 2022). 2022.

[7] H. S. G. Pussewalage, P. S. Ranaweera, and V. Oleshchuk. "PLC security and critical infrastructure protection". In: 2013 IEEE 8th International Conference on Industrial and Information Systems (Dec.

2013). 2013. DOI: `https://dx.doi.org/10.1109/ICIInfS.2013.6731959`.

[8] *Remote Terminal Unit.* URL: `https://en.wikipedia.org/wiki/Remote_terminal_unit` (visited on 2023-04-08).

[9] *HMI defintion.* URL: `https://csrc.nist.gov/glossary/term/human_machine_interface` (visited on 2023-04-11).

[10] *What is Modbus and how does it work?* URL: `https://www.se.com/us/en/faqs/FA168406/` (visited on 2023-04-13).

[11] *Hacking: Modbus - Cyber security OT/ICS.* URL: `https://blog.omarmorando.com/hacking/ot-ics-hacking/hacking-modbus` (visited on 2023-04-15).

[12] Modbus.org. "MODBUS/TCP Security". In: pp. 7–8. URL: `https://modbus.org/docs/MB-TCP-Security-v21_2018-07-24.pdf` (visited on 2023-04-16).

[13] ODVA Inc. "EtherNet/IP - CIP on Ethernet Technology". In: URL: `https://www.odva.org/wp-content/uploads/2021/05/PUB00138R7_Tech-Series-EtherNetIP.pdf` (visited on 2023-04-18).

[14] *Odva, Inc.* URL: `https://www.odva.org` (visited on 2023-04-21).

[15] *Introduction to EtherNet/IP Technology.* URL: `https://scadahacker.com/library/Documents/ICS_Protocols/Intro%20to%20EthernetIP%20Technology.pdf` (visited on 2023-04-19).

[16] *Common Industrial Protocol.* URL: `https://www.odva.org/technology-standards/key-technologies/common-industrial-protocol-cip/` (visited on 2022-12-26).

[17] *Common Industrial Protocol.* URL: `https://en.wikipedia.org/wiki/Common_Industrial_Protocol` (visited on 2023-04-21).

[18] *What is the Common Industrial Protocol (CIP)?* URL: `https://www.motioncontroltips.com/what-is-the-common-industrial-protocol-cip/` (visited on 2023-04-21).

[19] *CIP (Common Industrial Protocol): CIP messages, device types, implementation and security in CIP.* URL: `https://resources.infosecinstitute.com/topic/cip/` (visited on 2023-04-21).

[20] B. Green, M. Krotofil, and A. Abbasi. "On the Significance of Process Comprehension for Conducting Targeted ICS Attacks". In: Workshop on Cyber-Physical Systems Security and PrivaCy (Nov. 2017). ACM, 2017, pp. 57–67. DOI: `https://doi.org/10.1145/3140241.3140254`.

[21] A. Keliris and M. Maniatakos. "ICSREF: A Framework for Automated Reverse Engineering of Industrial Control Systems Binaries". In: 26th Annual Network and Distributed System Security Symposium (NDSS) (Feb. 2019). 2019. DOI: `https://doi.org/10.48550/arXiv.1812.03478`.

[22] Y. Yuan, X. Tang, W. Zhou, W. Pan, X. Li, H. T. Zhang, H. Ding, and J. Goncalves. "Data driven discovery of cyber physical systems". In: *Nature Communications 10* (2019-10-25). URL: `https://www.nature.com/articles/s41467-019-12490-1` (visited on 2023-04-20).

[23] C. Feng, V.R. Palleti, A. Mathur, and D. Chana. "A Sysematic Framework to Generate Invariants for Anomaly Detection in Industrial Control Systems". In: NDSS Symposium 2019 (San Diego, CA, USA, Feb. 24–27, 2019). 2019. DOI: `https://dx.doi.org/10.14722/ndss.2019.23265`. URL: `https://www.ndss-symposium.org/wp-content/uploads/2019/02/ndss2019_07A-3_Feng_paper.pdf` (visited on 2023-04-20).

[24] Singapore University of Technology and Design. *iTrust - Center for Research in Cyber Security.* URL: `https://itrust.sutd.edu.sg/` (visited on 2022-12-08).

[25] K. Pal, A. Adepu, and J. Goh. "Effectiveness of Association Rules Mining for Invariants Generation in Cyber-Physical Systems". In: 2017 IEEE 18th International Symposium on High Assurance Systems Engineering (HASE) (Jan. 2017). 2017. DOI: `https://doi.org/`

10.1109/HASE.2017.21. URL: https://ieeexplore.ieee.org/document/7911883 (visited on 2023-04-20).

[26] *Association rules mining.* URL: https://en.wikipedia.org/wiki/Association_rule_learning (visited on 2023-04-21).

[27] K. Pal, A. Adepu, and J. Goh. "Cyber-Physical System Discovery: Reverse Engineering Physical Processes". In: 3rd ACM Workshop on Cyber-Physical System Security (Apr. 2017). 2017, pp. 3–14. DOI: https://doi.org/10.1145/3055186.3055195.

[28] *Ceccato et al. reverse engineering prototype tool.* URL: https://github.com/SeUniVr/PLC-RE (visited on 2023-04-23).

[29] Singapore University of Technology and Design. *Secure Water Treatment.* URL: https://itrust.sutd.edu.sg/testbeds/secure-water-treatment-swat/ (visited on 2023-04-23).

[30] R. Lanotte, M. Merro, and A. Munteanu. "Industrial Control Systems Security via Runtime Enforcement". In: *ACM Transactions on Privacy and Security* 26.4 (2023-02), pp. 1–41. DOI: https://doi.org/10.1145/3546579.

[31] M. Oliani. *AttackPLC - Project for Network Security teaching.* URL: https://github.com/marcooliani/AttackPLC (visited on 2023-04-23).

[32] *Ray - Productionizing and scaling Python ML workloads simply.* URL: https://www.ray.io/ (visited on 2023-04-25).

[33] *Tshark.* URL: https://tshark.dev/ (visited on 2023-04-25).

[34] *Wireshark.* URL: https://www.wireshark.org/ (visited on 2023-04-25).

[35] *pandas - Python Data Analysis library.* URL: https://pandas.pydata.org/ (visited on 2023-04-25).

[36] *NumPy - The fundamental package for scientific computing with Python.* URL: https://numpy.org/ (visited on 2023-04-25).

[37] *R project for statistical computing.* URL: https://www.r-project.org/ (visited on 2023-04-25).

[38] *SciPy - Fundamental alghorithms for scientific computing with Python.* URL: `https://scipy.org/` (visited on 2023-04-25).

[39] *The Daikon dynamic invariant detector*. URL: `http://plse.cs.washington.edu/daikon/` (visited on 2023-04-25).

[40] *Enhancing Daikon output*. URL: `https://plse.cs.washington.edu/daikon/download/doc/daikon/Enhancing-Daikon-output.html#Splitter-info-file-format` (visited on 2023-04-25).

[41] *Process mining*. URL: `https://en.wikipedia.org/wiki/Process_mining` (visited on 2023-04-26).

[42] *Fluxicon Disco*. URL: `https://fluxicon.com/disco/` (visited on 2023-04-26).

[43] *OpenPLC - Open source PLC software*. URL: `https://openplcproject.com/` (visited on 2023-04-23).

[44] *Docker*. URL: `https://www.docker.com/` (visited on 2023-04-26).

[45] MathWorks. *Simulink*. URL: `https://it.mathworks.com/products/simulink.html` (visited on 2023-04-26).

[46] *ProM Tools - The Process Mining Framework*. URL: `https://promtools.org/` (visited on 2023-04-26).

[47] Fraunhofer Institute for Applied Information Technology. *pm4py - Process Mining for Python*. URL: `https://promtools.org/` (visited on 2023-04-26).