

UNIVERSITATEA POLITEHNICA BUCUREȘTI  
FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE  
DEPARTAMENTUL CALCULATOARE



## RAPORT DE CERCETARE

Controller Logic Programabil (PLC) pentru Aplicații Industriale

Nicolae-Andrei Vasile

**Coordonator științific:**  
Conf. Dr. Ing. Dan Stefan Tudose

**BUCUREȘTI**

2022

UNIVERSITY POLITEHNICA OF BUCHAREST  
FACULTY OF AUTOMATIC CONTROL AND COMPUTERS  
COMPUTER SCIENCE DEPARTMENT



## RESEARCH REPORT

Programmable Logic Controller (PLC) for Industrial Applications

Nicolae-Andrei Vasile

**Thesis advisor:**

Conf. Dr. Ing. Dan Stefan Tudose

**BUCHAREST**

2022

## TABLE OF CONTENTS

Abstract.....	2
1 Introduction.....	3
1.1 Industrial Automation .....	3
1.2 History .....	3
1.3 Programmable Logic Controller (PLC) .....	3
1.4 Overview .....	4
2 State of the Art .....	5
2.1 Hardware Architecture.....	5
2.2 Standards and Norms.....	5
2.3 Programming Languages.....	6
2.4 Relay Ladder Logic.....	6
2.5 Petri Nets.....	8
2.6 Software Engineering in Industrial Automation .....	8
3 Related work.....	10
3.1 Petri Nets.....	10
3.2 Model Verification.....	13
3.3 Security.....	15
4 Conclusion .....	17
5 Bibliography.....	18
6 Anexes.....	21

## **ABSTRACT**

This report presents a basic overview of the industrial automation scene, alongside the fundamental component of it, The Programmable Logic Controller (PLC). We are aiming to provide a general idea of the state on which the domain is currently part of, discoveries, present approaches and trends, and future implementations and ideas.

The paper is structured in three main chapters: Introduction, where an initial description of the aspects discussed is given, State of the Art, where we explain more in-depth the means of the industrial automation domain, and Related Work, an overview of the scientific and practical approaches that have been done so far regarding PLCs and related topics as well.

# **1 INTRODUCTION**

## **1.1 Industrial Automation**

Industrial automation represents the use of control systems, such as robots and specialized computers to replace human operation. It handles all the necessary processes and machineries in order to perform the given tasks, following a strict set of rules, usually abstracted by an implemented software program.

Industrial automation has been a focus of research attention in both academia and industry alike for several decades. Due to that, strong expectations are in place in the context of increased productivity, improved part quality, reduced costs, and relaxed part design constraints. The basis for these expectations is two-fold. First, machining process automation, if exercised strategically and advantageously, can perform consistently for large batch production or flexibly for small batch jobs. Secondly, process automation can be set up to autonomously tune the parameters in pursuit of desirable performance (tolerance, finish, cycle time, etc.), thereby bridging the gap between product design and process planning while reaching beyond the human operators' capability.

The success of manufacturing process automation hinges primarily on the effectiveness of process monitoring and control systems.

## **1.2 History**

Before 1970's relays were used to control the automation processes in industrial environment. They are electro-mechanical devices used for switching higher loads. One of the limitations, such as hard wiring, configuration wearing out due to changes in wiring at each design, and so on.

As the automation processes grew more and more in complexity, debugging relay failures and changing functionality by modifying the sequence of operations became extremely time-consuming and costly because of the required rewiring, so the need for a better replacement was imminent at that time.

In 1968, Hydramatic Division of General Motors Corporation (GM) specified the design criteria for the first PLC, as described in Erickson's paper [1]. Since then, programmable logic controllers (PLC) have been the primary workhorse of industrial automation [2].

## **1.3 Programmable Logic Controller (PLC)**

A programmable logic controller (PLC) is an electronic device used in many industries to monitor and control building systems and production processes. Unlike PCs and smartphones,

which are designed to perform any number of roles, a PLC is designed to perform a single set of tasks, except under real-time constraints and with superior reliability and performance.

The architecture of the PLC is like a general-purpose computer, containing a central processing unit (CPU), power supply and input and output (I/O) modules. In fact, some of the early PLCs were computers with special I/O. However, as K. T. Erickson states in his work also [1], some important characteristics distinguish PLCs from general purpose computers. They can be placed in an industrial environment that has extreme temperatures, high humidity, electrical noise, electromagnetic interference, and mechanical vibration, making them suitable for industrial environments.

Further, the software for the PLCs can be developed using a variety of programming languages standardized in IEC 61131-3 (IEC standard for programmable controllers), and contains two textual ones: Structured Text (ST) and Instruction List (IL); and two diagrammatic ones: Function Block Diagrams (FBD) and Ladder Logic Diagrams (LLD), as described in articles [1], [2] and [3]. The ladder logic is by far the most common programming language used for developing industrial applications for PLCs [1].

## **1.4 Overview**

In this paper we are to analyze and document the development, approaches and integration of programmable logic controllers in the industrial scene. Categories of interest are the used standards, such as IEC61131-3 and IEC 61499, the differences between them and what impact do they have in PLC design and development, programming languages (ex.: Ladder Logic, Structured Text) and methods of organizing and creating embedded software for industrial applications, hardware architectures and the general state of the industrial scene.

Further, we aim to design and develop a lightweight programmable logic controller with usages in various industrial applications following the mandatory standards and procedures relying on the existent approaches as well.

In the next chapter, the State-of-the-Art in industrial automation using PLCs is described more in-depth.

## 2 STATE OF THE ART

### 2.1 Hardware Architecture

Although they have roughly the same sort of components found in many other computer systems, PLCs operate quite differently. A PLC operating cycle, consists of the following (please note that the order can differ):

- Reading and storing the current value of each input.
- Sequentially executing the instructions in program memory, while storing any updated variables or outputs to data memory.
- Changing all physical outputs to match the output table values stored in data memory.

The main components of a typical hardware architecture of a PLC include, according to “*Programmable Logic Controllers*” [1]:

- *Central Processing Unit (CPU)*: the CPU is responsible with the implemented logic execution for a given system. The operations are executed sequentially, according to the data provided by external machines, sensors, and so on.
- *Input and Output (I/O) modules*: the I/O connects various external devices to the PLC to read and process data, for the inputs, and to change the outputs according to the logic of the application. They provide as well means to program and debug the device using various tools and protocols.
- *Power supply*: the power supply provides power to all the components of the systems for them to work properly

In a small PLC, all the described parts are enclosed in a single compact unit, in contrast to larger PLCs, that have the possibility of acquiring them separately [1].

### 2.2 Standards and Norms

As Vyatkin describes in his paper [3], standards and norms are very influential in the design and development of industrial automation software. They provide the technical rules that ensure the safety and performance, giving the basis as well for conformance testing. We discuss the relevant ones in this section, the most important being *IEC 61499* and *IEC 61131-3*, as described in „*Formal methods in PLC programming*” [2] as well.

- *IEC 61131-3*

The IEC 61131-3 is the most successful standard for the industrial control software regarding PLC Programming, according to articles [2] and [3]. Various implementations of industrial applications are using it as the main source of rules and norms, such as papers [4], [5] and [6].

- *IEC 61499*

The IEC 61499 reference architecture has been formed to facilitate the use of distributed automation intelligence in systems with decentralized logic, according to references [3] and [7].

According to article [3], other notable mentions are *ISA 88/95*, intended to provide solutions to *application configuration* and *system integration* problems, *IEC 61804*, describes the specification and requirements of distributed process control systems based on function blocks, and *IEC 61850*, which addresses the interfacing issues and standardizes communication to avoid the use of vendor-specific protocols in power systems automation.

„*IEC 61499 as Enabler of Distributed and Intelligent Automation*” [7] discusses in part about the relationship between standards *IEC 61499* and *IEC 61131-3*. For example, ISaGRAF shows that it is possible to develop distributed control applications using both standards for PLC Programming.

## 2.3 Programming Languages

The programming languages that are to be discussed in this section conform to the international standard IEC 61131-3 for PLC Programming. As stated by Erickson [1], there are four defined languages that can be used: ladder logic, sequential function charts, function blocks and a text language (Structured text - C-like).

As many papers [1], [5], [6] and [8] show that the ladder logic is the most prevalent language. The ladder logic symbology was developed from the relay ladder logic wiring diagram. One of the reasons why the ladder logic programming language is mostly used, is because it is easy to learn [1], and the PLC software is commonly monitored and maintained by users, not by developers [3], giving it a huge advantage.

## 2.4 Relay Ladder Logic

Vyatkin states as well in his work [1] that the ladder logic diagram is only a symbolic representation of the computer program. In order to properly explain how the logic ladder diagrams work, we need to introduce basic components to work with. The fundamental symbols count to 3, and are as following [1]:

- Normally open (NO) contacts
- Normally closed (NC) contacts
- Output (relays coils)

For the symbols to compose a ladder diagram, the „rung” concept needs to be defined. A rung represents the horizontal line in a diagram, between the two power rails, that contain characteristic symbols in order to create the program logic [1], [8] and [9].



Normally open (CO) are the type of contacts that do not facilitate a continuous path unless they are energized by an outer source. On the other hand, the normally closed (NC) contacts usually connect the two heads in normal state, the path being disrupted when the contact is energized. As for the output (relay coils), they are usually energized when the contact is energized. As for the output (relay coils), they are usually energized when the rung they are part of have a continuous closed path from one power rails to another [1], [8] and [9].

The logic of a usual ladder diagram flows from the left power rails to the right power rail, and from the top rung to the bottom one.

Further, we will take an example from „*Programmable Logic Controllers*” [1] to describe the workflow of an actual diagram. Let us say that we want to energize an output with two contacts: A and B. The A contact is given as normally open (NO), and the B contact as normally closed (NC) and the output should be ON when A is ON, and B is OFF.

The image below shows the truth table for our example (a), the equivalent relay logic (b) and the equivalent ladder logic (c).

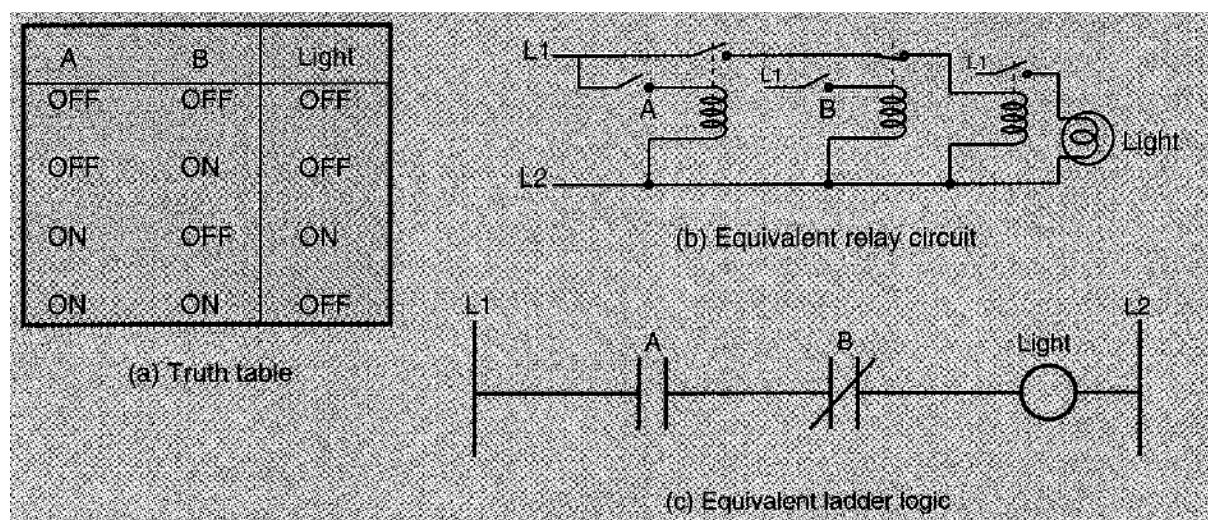


Figure 2-1. Logical NOT in ladder logic [1]

The standard symbols for NO and NC contacts and output (relay coil) are used. The ladder consists of only one rung that implements the described logic. In normal state, A contact does not connect its two pins, being normally open, in contrast to contact B. Therefore, when A is energized and B is left in normal state, a continuous path is created from one power rail to another, turning on the output.

## 2.5 Petri Nets

Traditionally, the RLL was developed to smooth the transition from relay control systems to PLCs. It is used as a way of capturing the sequence of operations executed by the system's control software, specify the I/O procedures of the Programmable Logic Controller (PLC) [8] and [9].

Many papers [2], [8] and [9] drew attention to the fact that, the larger the control system, the more difficult it is to determine the initial design specifications (how the system operates) by examining the control logic. The ladder logic diagrams grew more and more in complexity making the troubleshooting process extremely difficult. In addition, ladder logic diagrams are limited only to control the correspondent system [9].

Petri Nets came in the scene as a solution for the limitations of RLLs described above. Zhou and Twiss describe Petri Nets in their work [8] as being a graph-related model and a visually graphical tool designed for modeling, analysis, performance evaluation, and control of discrete event systems. Further, the capability of modeling sequential, asynchronous, and concurrent events that drive an industrial process is mentioned also in *"Comparing Ladder Logic Diagrams and Petri Nets for Sequence Controller Design Through a Discrete Manufacturing System"* [9] as well.

Petri Nets display great flexibility and understandability of the process, exceeding today's standard RLL programming techniques. Therefore, they hold a promise as a solution to modern industrial control problems, [8] and [9].

Figure 6-1 presents a basic overview of the PN model in contrast with ladder logic diagrams.

## 2.6 Software Engineering in Industrial Automation

The complexity and importance of software has become more and more relevant in the industrial automation domain. It is stated by Vyatkin in his work [3] that the ratio of software has increased from 20% up to 40% in the last decade, predicting a shift of the main activities of automation towards software engineering. The increasing relevance and dependability of software is sustained by *"A Real-Time Service-Oriented Architecture for Industrial Automation"* [10] also.

Software requirements are a very important part of a system design. Papers [3] and [10] describe the following as being quite important in developing industrial applications:

- *Interoperability*: The system characteristic to be able to communicate with others
- *Heterogeneity*: The capability to be diverse in content
- *Scalability*: The capacity to be changed in scale
- *Dependability*: The characteristic of systems to be reliable in any condition
- *Distribution*: The ability to decentralize the tasks over a set of distributed units

All designs take in consideration those characteristics and customize their approach and architecture according to the priority of each requirement. Various designs are proposed in this domain, but we will present three of the most important ones, as [2], [3], [10] describe as well:

- *Formal models*

The purpose of formal models is to create dependable software for industrial systems with properties that can be guaranteed by the design, according to [3].

Based on formal methods themselves, they support the need for high quality solutions and the application of PLC in safety-critical tasks to verify and validate certain characteristics of programs, such response times, and others [2]. In this way, the reliability of the systems in the given conditions is assured.

However, the practical use of formal models in industrial practice is not yet common [3].

- *Multi-agent architectures*

The multi-agent architectures put accent on modularity, scalability and flexible software design. Structure-wise, they offer an alternative solution based on decentralization of functions over a set of distributed entities [3].

However, the existent agent architectures do not always fulfill the practical industrial requirements of a plant. It can easily reach the point where the resource usage and response time requirements become unattainable because of the overhead created by the model.

- *Service-oriented architectures*

Service-oriented architectures came as a proposed solution to the high demand of for efficiency in machine reconfiguration and reduced time-to-market of new products [3] and [10]. These models ensure a high level of interoperability between the system's components, distribution for units of the systems, contributing to scalability and flexibility as well.

According to [10], a bottleneck in today's industrial systems which affects efficiency and flexibility is represented by the network infrastructure between the components. Standardized protocols are making way into the industrial scene, such as Modbus, Ethernet, and so on, providing a simpler workflow for engineers. In today's systems, most of the communication networks are proprietary.

### 3 RELATED WORK

#### 3.1 Petri Nets

After the origination of Petri nets in 1962, significant work has been accomplished on this subject. Due to the increasing complexity of the ladder logic diagrams, researchers from both academic and industrial scenes have started to pursue the possibilities of using Petri nets, in order to eliminate the limitations brought by relay ladder logic [9], [11], [12], discussed in section 2.5. On the theoretical side, the focus was designing a mathematical framework in which different properties of a Petri net can be analyzed, as well as extending the modeling capabilities of PNs. On the practical side, however, PNs have been used for specification, verification and performance evaluation of different systems, such as computers or manufacturing lines [13].

Formally, a PN can be described by a five-tuple [8], [11], [12], as follows:

$PN = (P, T, I, O, M_0)$ ; where:

- 1)  $P = \{p_0, p_1, \dots, p_m\}$  is a finite set of places;
- 2)  $T = \{t_0, t_1, \dots, t_n\}$  is a finite set of transitions, where  $P \cup T \neq \emptyset$  and  $P \cap T = \emptyset$
- 3)  $I: P \times T \rightarrow N$  is an input function that specifies arcs directed from places to transitions, where  $N$  is the set of all natural numbers;
- 4)  $O: P \times T \rightarrow N$  is an output function which defines directed arcs from transitions to places;
- 5)  $M_0: P \rightarrow N$  is the initial marking;

According to references [8], [11] and [12], places are used to define conditions, resource availability, or process status, while transitions represent events of the system. Alongside input and output functions, the mentioned two constitute the structure of a PN. A marking is a vector which holds the number of tokens at each defined place. ( $i^{th}$  element represents the number of tokens for the  $i^{th}$  place). The state is given by the set of markings .

For a marking in a PN to be changed, it must act according to a set of rules that define the dynamic behavior of a system. That is, the underlying flow of the tokens of said marking is determined by what are called *enabling* and *firing* rules [8], [11], which are defined as follows:

- 1) *Enabling* rule:

A transition  $t$  is said to be enabled if each input place  $p$  of  $t$  contains the number of tokens at least equal to the weight of the arc connecting  $p$  to  $t$ .

- 2) *Firing* rule:

- a) an enabled transition  $t$  may or may not fire (depending if the event occurs or not);

- b)  $I(p, t)$  tokens are removed from each input place  $p$  of  $t$  when an enabled transition is fired, and  $O(p, t)$  tokens are added to each output place  $p$  of  $t$ , where  $O(p, t)$  is the weight of the arc connecting  $p$  to  $t$ ;

PNs, as a mathematical tool, possess a number of properties for the model designer to investigate whether the system lacks important specific functionalities that belong to the application domain [11]. The PN's properties, that are often mentioned in literature [8], [11], [12], are as follows:

#### 1) *Reachability*

It is important to know if a system can reach a specific state as a result of a required functional behavior. Therefore, it is necessary such transition firing sequence that would result in transforming the initial marking  $M_0$  into  $M_i$ , where  $M_i$  is the specified state and the firing sequence represents the required functional behavior.

#### 2) *Boundedness and Safeness*

The property which determines whether a modeled system has overflows or not. A PN is said to be  $k$ -bounded if the number of tokens at any place  $p$ , where  $p \in P$ , is smaller or equal to  $k$  ( $k$  is a nonnegative integer). In addition, a PN is safe if it is 1-bounded [8], [11].

#### 3) *Conservativeness*

If a vector  $w$  exists, where  $w = \{w_0, w_1, \dots, w_m\}$ ,  $m$  being the size of  $P$ , such as  $w(p) > 0$ , for each  $p \in P$ , then the PN is conservative. If a transition has the same amount of output arcs as input arcs, then the PN is conservative.

#### 4) *Liveness*

In some cases, deadlocks can appear in the modeled systems. A deadlock-free system is required to be live, meaning that for all markings  $M$ , which are reachable from  $M_0$ , it is possible to fire any transition in the current net following a specified firing sequence. Because this requirement might be too strict for real-life scenarios, five levels of liveness are introduced for a transition  $t$ :

- a)  $L_0$  – *live*: there is no firing sequence in  $L(M_0)$  in which  $t$  can fire;
- b)  $L_1$  – *live*:  $t$  can be fired at least once in some firing sequence in  $L(M_0)$ ;
- c)  $L_2$  – *live*:  $t$  can be fired at least  $k$  times in some firing sequence in  $L(M_0)$  given any positive integer  $k$ ;
- d)  $L_3$  – *live*:  $t$  can be fired infinitely often in some firing sequence in  $L(M_0)$ ;
- e)  $L_4$  – *live*:  $t$  is  $L_1$ -Live in every marking in  $L(M_0)$ ;

A PN is said to be  $L_i$  – *live*, for marking  $M_0$ , if every transition in the net is  $L_i$  – *live* [11].

### 5) Reversibility and Home State

One of the requirements that a system should have is to be able to recover from failure states and return to preceding correct states. A PN is said to be reversible if for each  $M \in R(M_0)$ ,  $M_0$  is reachable from  $M$ . Moreover, a PN state  $M_i$  is said to be home state if for each  $M \in R(M_0)$ ,  $M_i$  is reachable from  $M$  [11]. The home state is less restrictive than the reversibility property of a PN.

Another interesting concept has been introduced in order to achieve real-time control, an augmented version that assigns timing and I/O information to the general PN model, RTPN (Real-Time PN) [8], [9]. RTPNs can be defined by an eight-tuple [8], as follows:

$RTPN = (PN, D, X, Y)$ ; where:

- 1)  $PN = (P, T, I, O, M_0)$  is the general PN model;
- 2)  $D: P \rightarrow \mathbb{R}^+$  is a firing time-delay function, where  $\mathbb{R}^+$  is the set of nonnegative real numbers;
- 3)  $X: P \rightarrow B$  is an input signal function, where  $B$  is the set of Boolean expressions of input addresses;
- 4)  $Y: T \rightarrow \{0,1\}$  is an output signal function;

The timing vector  $D$  is defined to associate time delays to transitions, replicating real-life scenarios. It is to be noted that in reference [9],  $X$  is associated to places, and is intended to read the input signals from the digital input interface, and output signal  $Y$  is associated to transitions, and is used to send output signals through the digital output interface. However, in reference [8],  $X$  is associated with transitions, and  $Y$  with places, suggesting that it is easier to work with.

A few of the experimental approaches that use Petri nets will be discussed next.

Zhou and Twiss describe an example of design of both relay ladder logic and Petri nets using an industrial system in their work [8]. The paper presents for the first time a convincing comparison between Petri nets and relay ladder logic for an industrial automated system. Unfortunately, the real-time implementation of the detailed Petri nets design was not performed, due to the resource constraints.

A comparison of LLDs and PNs for sequence controller design is done in “*Comparing Ladder Logic Diagrams and Petri Nets for Sequence Controller Design Through a Discrete Manufacturing System*” [9] through a discrete manufacturing system. According to this work, design complexity and response time are the most important factors for comparing LLDs and PNs. Moreover, it presents real-time PNs as an emerging technology that can be applied to control any discrete event system that has digital input/output interfaces and a computer.

A one-to-one mapping technique is used to transform Petri nets model into PLC programming languages, in particular LLDs, is described in “*Transformation from Petri Nets Model to Programmable Logic Controller using One-to-One Mapping Technique*” [12]. The

objective of this paper is to use modelling techniques, Petri nets for graphical and mathematical representation, and to generate LLDs based on the model.

Thomas Boucher presents in his work [13] and automated manufacturing cell controlled by Petri nets. He has illustrated the application of a PN design to control a machining cell and has also pointed out some advantages of PNs over more traditional control methods, such as RLL.

### 3.2 Model Verification

Since system failures or undetected errors can have hazardous effects on people and the environment also, it is highly recommended that proper verification and validation to be performed in order to ensure the correctness of system's implementation. Many different techniques are used in industry to check the PLC programs, e.g. manual and automatic testing or simulations, however, they come with various limitations [14] . As described in section 2.6 also, formal verification is a solution that fulfills those requirements, ensuring that the issues difficult to detect are, in fact, observed and corrected, such as safety, liveness and others [14], [15], [16].

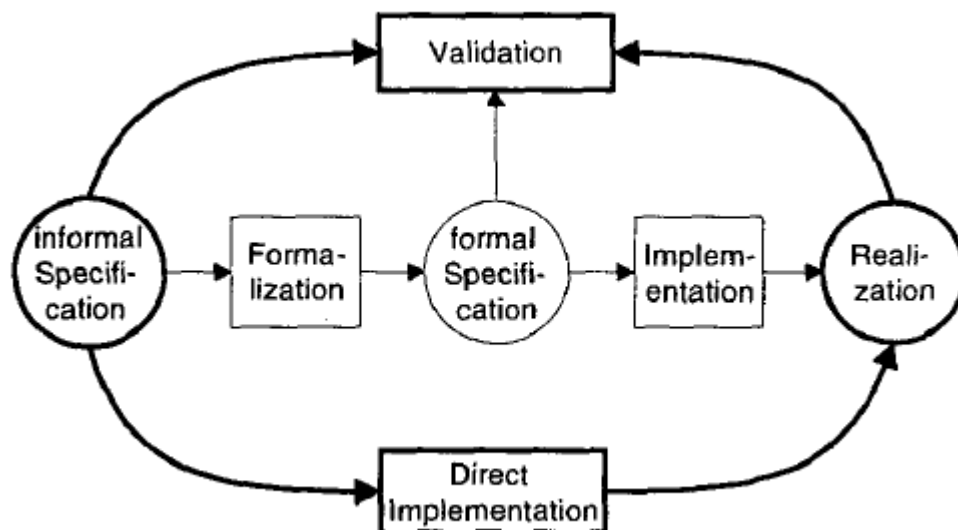


Figure 3-1. Design process for logic control systems [2]

According to Frey and Litz [2], a general design process for a logic control system begins with an *informal specification*, which describes in a relaxed fashion what the general behavior should be, without defining the exact elements of the model precisely. The *direct implementation* and *realization* steps use *informal specification* to develop the control

system. This is the popular approach in the industry, according to references [2], [14], [15] and [16], and the *validation* is done by control designers and users. This requires quite a long time and can be hard to find various mistakes regarding the model. Including formal methods, three more steps are added in the middle part of Figure 3-1. The *formalization* step transforms the *informal specification* into *formal specification*, defining all the required parts accordingly. The *implementation* develops both the hardware and software parts using the mentioned *formal specification*.

A few approaches will be discussed next, as significant work has been done in this field.

“Applying model checking to industrial-sized PLC programs” [14], presents an interesting implementation of automated formal verification of complex properties regarding PLC control systems developed at CERN, using the Siemens manufacturer as the main device provider. The paper provides a straightforward automated methodology for formal verification of PLC programs, making use of an intermediate model (IM) to facilitate the transition between all the PLC programming and formal modeling languages that are used.

Another similar approach of the topic is the work of Biallas, Brauer and Kowalewski [16]. Using an intermediate model as well, TODO

“Efficient representation for formal verification of PLC programs” [15], takes a slightly different approach, proposing various representations of PLC programs that favor the application of model-checking techniques.

All of the mentioned papers take into account the biggest problem model checking is currently facing, the *state space explosion problem*, stating that the number of states in a system can reach enormous values in certain cases, e.g. for  $m$  processes, each with  $n$  states, the asynchronous composition of said processes would be  $m^n$  [17].

Various techniques have been proposed in literature [17], [18], to combat the *state explosion problem*; among the notable advances in this area there are:

#### 1) Symbolic Model Checking with Binary Decision Diagrams

Finite state-transition systems can be represented and manipulated as *ordered binary decision diagrams*, which are a canonical form of Boolean formulas. The reason why the *symbolic model checking* relies on such representation and manipulation is the fact that OBDD is more compact and efficient than by listing each state individually.

#### 2) Partial Order Reduction

Many events are independent of each other and can be executed in arbitrary order without affecting the system’s expected behavior. Partial order reduction, one of the most used techniques in asynchronous systems, avoids exploring certain paths in the state-transition model.



### 3) *Counterexample-Guided Abstraction Refinement*

When the state space of a system is very large, then going through each state is not feasible. The *counterexample-guided abstraction refinement* method uses counterexamples to refine an initial abstraction, taking away the irrelevant information from the system.

### 4) *Bounded Model Checking*

*Bounded model checking* generates a formula that is fed to a *Boolean satisfiability (SAT) solver*, and is satisfiable only if it can be disproved by a counterexample of length  $k$ , where  $k$  is the defined bound.

*Bounded model checking (BMC)* is one of the most used approaches in dealing with the *state explosion problem* in today's industry [17]. However, references [14] and [16] make use of the *counterexample-guided abstraction refinement (CEGAR)* method to resolve the described problem. In addition, article [16] further implements a predicate abstraction based on *ordered binary decision diagrams (OBDD)* to further reduce the state space.

## 3.3 Security

Given the fact that PLCs control the behavior of control systems and are responsible for the safe operation for physical processes, especially in manufacturing environments, it is required to ensure that no other sequence is executed, rather than the one intended for that specific system (e.g. malicious code that turns all the safety checking sequence down before a physical operation, or turns all the sensing unit off for a period of time) [19], [20], [21].

An increasing amount of attention has been paid since the Stuxnet attack [22], [23], regarding the vulnerability and susceptibility of PLCs of such breaches.

According to references [22] and [23], Stuxnet was a 500-kilobyte worm that, unlike most malware, targeted industrial control systems which are used in factories, industry lines and power plants. The attack had three phases: it selected Microsoft Windows machines and networks, replicating on each viable host, searched for industrial control systems software, in particular Siemens Step7 (Siemens system control devices were the main target), to, ultimately, corrupt and compromise the PLCs.

What makes Stuxnet so interesting is the complexity of the implementation, showing detailed knowledge of the target, and the very precise target selection, mainly industrial control systems. It made use of four zero-day Windows exploits, from using USB and network related vulnerabilities, to operating system priority selection ones in order to remain undetected [22], [23].

Various approaches have been pursued since Stuxnet attack, that describe similar behaviors of such malware [20], [21]. We can observe four main steps for the attack to be performed successfully:

*1) Infection*

As with any malicious attack, one or more targeted hosts need to be infected with the malware. Said infection can be done in various ways, such as via Internet, connecting an USB drive to the host, or any other attack vectors.

*2) Process analysis*

In the beginning, the malware does not know how the system is structured and what peripherals are in place. It needs to perform a process representation in order to obtain the execution sequence via decompiling the source code in mnemonics and translating it into higher-level languages, as well as mapping found specifications to a model. Furthermore, the peripheral-related variables found in PLC's memory must be mapped to input/output connected devices as well.

*3) Payload generation*

The information extracted in the previous step may not be sufficient for generating a feasible payload. Therefore, additional error checking and resolving potential conflicts are in place before generating the desired payload.

*4) Payload execution*

The last part of the malware is uploading the payload into the PLCs in order to be executed.

One of the additional features that the presented work has over Stuxnet is the flexibility of the dynamic payload. The designer does not need to know precisely what is the structure of the targeted system, unlike Stuxnet that came with a precompiled malicious code, the malware being autonomous in extracting the necessary information to generate the payload accordingly.

Another notable contribution for the security of industrial control systems is article "*Security for Industrial Communication Systems*" [19]. The paper describes how the connection to enterprise networks or Internet increases the risk of a network-based attack on industrial control systems, and provides several standardizations and best-practices in order to ensure the safety of the system.

## **4 CONCLUSION**

In this paper we have cover a basic overview of the industrial automation domain and programmable logic controllers (PLCs), current problems that we are facing today proposed solutions to them, as well as various approaches both in academic and industry scene that make use of said controllers.

Since 1970's, PLCs have been a fundamental part in automation processes and raised attention in both industrial and academic scene, representing a research topic for decades now. Further, software engineering is expected to grow in importance in automation due to the need of more complex and reliable systems, therefore the specified domain is a large and growing area with a rich body of knowledge.

Future work consists of documenting new software architectures that have emerged recently and, ultimately, building a programmable logic controller on our own, using the techniques that fit our requirements and purposes most.

## 5 BIBLIOGRAPHY

- [1] K. T. Erickson, "Programmable Logic Controllers," *IEEE Potentials*, vol. 15, no. 1, pp. 14-17, 1996.
- [2] G. Frey and L. Litz, "Formal methods in PLC programming," in *IEEE International Conference on Systems, Man and Cybernetics*, Nashville, TN, USA, 2000.
- [3] V. Vyatkin, "Software Engineering in Industrial Automation: State-of-the-Art Review," *IEEE Transactions on Industrial Informatics*, vol. 9, no. 3, pp. 1234-1249, 2013.
- [4] D. Schütz, A. Wannagat, C. Legat and B. Vogel-Heuser, "Development of PLC-Based Software for Increasing the Dependability of Production Automation Systems," *IEEE Transactions on Industrial Informatics*, vol. 9, no. 4, pp. 2397-2404, 2013.
- [5] O. G. Bellmunt, D. Montesinos-Miracle, A. Sumper, S. Galceran-Arellano and A. Sudrià-Andreu, "A Distance PLC Programming Course Employing a Remote Laboratory Based on a Flexible Manufacturing Cell," *IEEE Transactions on Education*, vol. 49, no. 2, pp. 278-284, 2006.
- [6] M. G. Ioannides, "Design and Implementation of PLC-Based Monitoring Control System for Induction Motor," *IEEE Transactions on Energy Conversion*, vol. 19, no. 3, pp. 469-476, 2004.
- [7] V. Vyatkin, "IEC 61499 as Enabler of Distributed and Intelligent Automation: State-of-the-Art Review," *IEEE Transactions on Industrial Informatics*, vol. 7, no. 4, pp. 768-781, 2011.
- [8] M. Zhou and E. Twiss, "Design of Industrial Automated Systems Via Relay Ladder Logic Programming and Petri Nets," *IEEE Transactions on Systems, Man and Cybernetics - part C: Applications and Reviews*, vol. 28, no. 1, pp. 137-149, 1998.
- [9] K. Venkatesh, M. Zhou and R. J. Caudill, "Comparing Ladder Logic Diagrams and Petri Nets for Sequence Controller Design Through a Discrete Manufacturing System," *IEEE Transactions on Industrial Electronics*, vol. 41, no. 6, pp. 611-619, 1995.
- [10] T. Cucinotta, A. Mancina, G. Anastasi, G. Lipari, L. Mangeruca, R. CheccoZZo and F. Rusinà, "A Real-Time Service-Oriented Architecture for Industrial Automation," *IEEE Transactions on Industrial Informatics*, vol. 5, no. 3, pp. 267-276, 2009.
- [11] R. Zurawski and M. Zhou, "Petri nets and industrial applications: A tutorial," *IEEE Transactions on Industrial Electronics*, vol. 41, no. 6, pp. 567 - 583, 1994.

- [12] D. Thapa, S. Dangol and G.-N. Wang, "Transformation from Petri Nets Model to Programmable Logic Controller using One-to-One Mapping Technique," in *International Conference on Computational Intelligence for Modelling, Control and Automation and International Conference on Intelligent Agents, Web Technologies and Internet Commerce*, Vienna, Austria, 2005.
- [13] T. O. Boucher, M. A. Jafari and G. A. Meredith, "Petri net control of an automated manufacturing cell," *Computers & Industrial Engineering*, vol. 17, no. 1-4, pp. Pages 459-463, 1989.
- [14] B. F. Adiego, D. Darvas, E. B. Vinuela, J.-C. Tournier, S. Bliudze, J. O. Blech and V. M. Gonzalez Suarez, "Applying Model Checking to Industrial-Sized PLC Programs," *IEEE Transactions on Industrial Informatics*, vol. 11, no. 6, pp. 1400-1410, 2015.
- [15] V. Gourcuff, O. De Smet and J. M. Faure, "Efficient representation for formal verification of PLC programs," in *International Workshop on Discrete Event Systems*, Ann Arbor, MI, USA, 2006.
- [16] S. Biallas, J. Brauer and S. Kowalewski, "Arcade.PLC: a verification platform for programmable logic controllers," in *IEEE/ACM International Conference on Automated Software Engineering*, Essen, Germany, 2012.
- [17] E. M. Clarke, W. Klieber, M. Novacek and P. Zuliani, "Model Checking and the State Explosion Problem," in *LASER Summer School on Software Engineering: Tools for Practical Software Verification*, Elba Island, Italy, 2012.
- [18] E. M. Clarke, O. Grumberg, S. Jha, Y. Lu and H. Veith, "Progress on the State Explosion Problem in Model Checking," in *Informatics - 10 Years Back. 10 Years Ahead*, 2001.
- [19] D. Dzung, M. Naedele, T. P. Von Hoff and M. Crevatin, "Security for Industrial Communication Systems," *Proceedings of the IEEE*, vol. 93, no. 6, pp. 1152-1177, 2005.
- [20] S. McLaughlin and P. McDaniel, "SABOT: specification-based payload generation for programmable logic controllers," in *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, Raleigh, NC, United States, 2012.
- [21] S. McLaughlin, "On dynamic malware payloads aimed at programmable logic controllers," in *Proceedings of the 6th USENIX conference on Hot topics in security*, San Francisco, CA, 2011.
- [22] T. M. Chen and S. Abu-Nimeh, "Lessons from Stuxnet," *Computer*, vol. 44, no. 4, pp. 91-93, 2011.
- [23] D. Kushner, "The real story of stuxnet," *IEEE Spectrum*, vol. 50, no. 3, pp. 48-53, 2013.



## 6 ANEXES

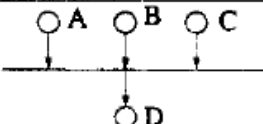
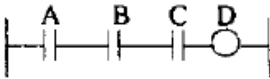
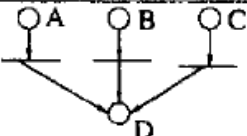
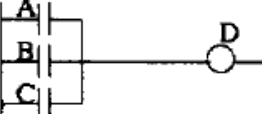
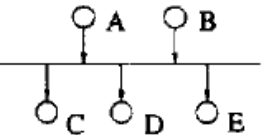
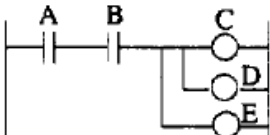
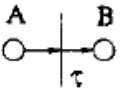
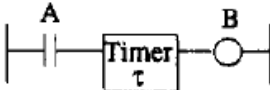
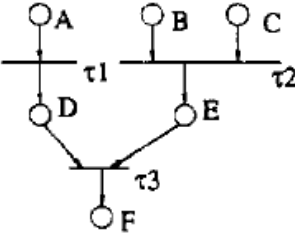
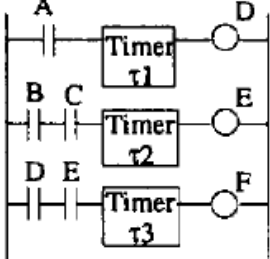
Logic constructs	Petri nets	Ladder logic diagrams
Condition or the status of a system element	○ Place	No explicit representation
An activity	— Transition	No explicit representation
Flow of information or material.	→ Directed arc	No explicit representation
Objects such as machines, robots, pallets, etc.	⊙ Token(s) in place(s)	No explicit representation
<b>Logical AND</b> IF A = 1 and B = 1 and C = 1 THEN D = 1		
<b>Logical OR</b> IF A = 1 or B = 1 or C = 1 THEN D = 1		
<b>Concurrency</b> IF A = 1 and B = 1 THEN C = 1 and D = 1 and E = 1		
<b>Time delay</b> IF A = 1 THEN delay "τ time units" B = 1		
<b>Synchronization</b> IF A = 1 THEN delay "τ1 time units"; D = 1 IF B = 1 and C = 1 THEN delay "τ2 time units"; E = 1 IF D = 1 and E = 1 THEN delay "τ3 time units"; F = 1		

Figure 6-1. Control logic representation by PNs and LLDs