Agile Synthesis and Identification of Industrial Processes

José Meleiro, José Gaspar
Instituto Superior Técnico / UTL, Lisbon, Portugal
francisco.meleiro@gmail.com, jag@isr.tecnico.ulisboa.pt

I. Introduction

Every Industrial Process, is a Discrete Event System (DES) and this idea is easily proved with the definition of DES. A Discrete Event System is a dynamic system in which its state evolution depends on the inputs of the system. The element that allows the implementation of DES in the physical world is the Programmable Logic Controller (PLC), that is the most common used controller in industry. A PLC is a programmable device and its working flow has three phases that are: (i) input signals reading; (ii) program execution; (iii) output signals writing. Revisiting the previous definition of DES, it is clear that PLCs implement DESs, where a state evolution (output writing) depends on the inputs (input signals reading).

Sometimes it is hard to develop programs that run on PLC, due to the complexity of the considered system, or due to the large number of signals on the system. A DES, can be represented in multiple forms, but the most appropriate representation is on Petri Nets, that besides its simple notation, also has properties that allow the study of system dynamics. In this work, is studied and implemented a technique that synthesizes a program that runs on PLCs, from a Petri Net representation of a Discrete Event System.

On this work it is also studied an algorithm of black-box identification of Discrete Event Systems. The identification on DESs, addresses the problem of construct a model of an unknown system, from its input and output signals.

A. Related Work

In 1962, in Dr. Petri's Ph.D. Dissertation [20], it was introduced the concept of Petri Nets, that was invented to describe chemical processes. Nevertheless, it was only in 1981 that [19] presented a complete work about Petri Nets, on his book.

Cutts and Rattigan [2] proposed in 1992, the use of Petri Nets to modelling systems before its implementation on PLCs as the solution to the problem of being hard to predict the results of illegal inputs, for systems with a large number of inputs. Later, in 2002 Minas and Frey [18], presented and developed graphical programming software to convert Petri Nets diagrams into PLC programming language Instruction List (IL). In 2015, on his MSc Dissertation, Gonçalves [3] proposed/used MATLAB software to convert a Petri Net into PLC programming language Structured Text (ST).

The concept of Identification of Discrete Events Systems arises, in the 60 years of the last century, to solve problems of language identification. On Gold's work in 1967[12], from positive samples of accepted words, a finite automaton was built. Later other works were developed for the identification of finite-state machines [15], [21].

In the last decades, several identification methods had been presented, where input and output signals of unknown DESs, are processed in order to obtain a representation of such systems. In 1998 [1], Meda-Campanã introduced a methodology based on the least square estimator. In 2005, S. Klein, L. Litz and J. J. Lesage, [17][16], proposed a method, where from the I/O signals measured from the DES initial condition, builds a non deterministic finite automaton. It was also in 2005, that A. Guia and C. Seatzu. [11], introduced an identification method for DESs, solving an (ILP) Integer Linear Programing Problem. M. Dotoli introduced in 2008 [4] a method where, from the system signals, is built an Interpreted Petri Net that represents the studied Discrete Event System.

More recently Estrada-Vargas, López-Mellado and J.J. Lesage, introduced in 2012 [6], a methodology of identification based on statistical tools, to build Interpreted Petri Nets with the observable behavior of the analyzed system, from its input and output signals. One year later, the same authors proposed on [7] a method of identification of partially observable DES. All the previous works, are also introduced in 2013, in Estrada-Vargas Ph.D Dissertation [5].

B. Objectives and Challenges

The goal of this work, is to study the algorithm introduced on [6] to solve the problem of identification on Discrete Event Systems. More in detail, the algorithm that is going to be studied, builds Petri Net models from the observable behavior of systems. One of the challenges of this study is the absence of an available program that implements the algorithm. In this work, it is proposed an implementation of the referred algorithm in a JAVA program.

On a black-box model, there is no prior knowledge of the system to be studied. The only available information of such a systems is its input/output signals. There are several options to monitoring the signals of a system. In [6], the input and output sequences are acquired using embedded software. This solution increases the complexity of PLC program and increases the execution time of each cycle,and the consumption of memory resources. On this thesis, a different option to monitoring the system signals is suggested. Our proposal, involves the use of hardware to acquire signals from system, outside the controller (PLC), with low cost hardware.

II. Theoretical Introdution

A discrete event system (DES) is a dynamic event-driven system, in which the state evolutions happen at discrete events over time. An event is the trigger for the state transition an can be of various types like an action taken, the fulfillment of a certain condition or the occurrence of a spontaneous event. DESs are presents in multiple systems of ours day-to-day, and can be implemented on controllers like Programmable Logic Controllers (PLC), because usually a controller change its state when an input change occurs, which translates into a change of outputs. A Petri Net(PN) is the most common used representation of Discrete Events Systems, because of its notation simplicity and graphical representation that provide a lot of structural information about the system.

A PN graph has five components that are: Places, represented by circles and represent system states; Transitions, represented by bars and represent events; Marking, represented by black dots inside the circle of a place and represent the state; Oriented Arcs, represented by arrows that link Places to Arcs and Arcs to Places.

A. Petri Net Definitions

Definition 1.1: A Petri net is a quintuplet, that can be represented by (P, T, A, w, x), where: P is a finite set of places. T is a finite set of transitions, A is a set of arcs that connect places to transitions and transitions to places, w is a weight function, that stores the weight of each arc and x are the markings of places.

Definition 1.2: A Petri net is a quintuplet, that can be represented by (P, T, I, O, x), where: P is a finite set of places. T is a finite set of transitions, I is a transition input function O is a transition output function and x are the markings of places.

B. Incidence Matrix

The state evolution of a Petri net, can be expressed in function of x and B, the last one also known as Incidence Matrix. The formal notation of a Petri Net evolution is presented bellow.

$$x(k+1) = x(k) + B \times q(k)$$

where: $\mu(k)$ is the present marking; $\mu(k+1)$ is the new marking; $q(k)$ is the triggering vector; $B = w(t_j, p_i) - w(p_i, t_j)$, is the incidence matrix.

An Incidence Matrix, is a representation of the places (rows of B) and the transitions (columns of B), of a PN. If a transition t_j, is reached from place p_i, the value of entry (i, j) on B is -1. In the same sense, if a place p_i, is reached from transition t_i, the value of entry (i, j) on B is 1.

C. Petri Net Properties

Petri Net properties provide us a way to characterize systems. In this sense, in the following are introduced the referred properties.

Property 1. Reachability: The group of all possible markings that can be obtained, given a Petri Net $N = (P, T, I, O, x_0)$ with initial marking x_0, is the Reachable Set, $R(N)$.

The reachability problem, responds to the problem of determine if the marking x' belongs to the Reachable Set $x' \in R(N)$, given a Petri Net $N = (P, T, I, O, x_0)$ with initial marking x_0.

Property 2. Coverability: given a Petri Net $N = (P, T, I, O, x_0)$ with initial marking x_0, and $x', x'' \in R(N)$, if

$$x'(i) \leq x''(i),$$

then x' is covered by x''.

Property 4. Boundedness: A Petri net place $p_i \in P$ is $k - bounded$ if, for all possible markings $x' = (x'_1, ..., x'_i, ..., x'_N) \in R(N)$, it is verified that

$$x'_i \leq k$$

If all the places of a Petri Net are $k-bounded$, the considered Petri Net is $k - bounded$.

Property 3. Safeness: A Petri Net place $p_i \in P$ is safe if, for all possible markings $x' \in R(N)$, it is verified that

$$x'_i \leq 1$$

In other words, a Petri Net place is safe if is $1 - bounded$. If all the places of a Petri Net are $1 - bounded$, the considered Petri Net is safe.

Property 5. Conservation: If for every marking, the number of marks remains the same, a Petri Net is strictly conservative. In a formal representation, a PN $N = (P, T, I, O, x)$ is strictly conservative if for every marking $x' \in R(N)$,

$$\sum x'(p_i) = \sum x(p_i)$$

Property 6. Liveness: If a Petri Net never reaches a deadlock, the PN is alive.

D. Alternative Representations of Petri Nets

Definition 1.3: An Interpreted Petri Net (IPN) is a triplet, that is represented by N=(Q, M_0), where, Q is a net structure and M_0, is the initial marking. On the sextuplet Q, of the net structure, G is the Petri Net Structure ($G = (P, T, A, w, x)$), Σ the input signals, Φ the output signals, λ is the output labeling functions and φ is an output function.

III. Discrete Event System Synthesis

In this Section, the Discrete Event System used on this thesis and the ST compiler are presented.

A. Case Study

The Discrete Event System used on this thesis is an alarm like the one used on the course of Industrial Processes Automation [9], where a PLC controls an interface that simulates the inputs and outputs of the alarm. The proposed alarm works like a store alarm and has two modes of operation that are the Presence Detector and the Active mode. The first mode, can be seen as an acoustic alert that is triggered anytime that a customer enters the store. When a client opens the store

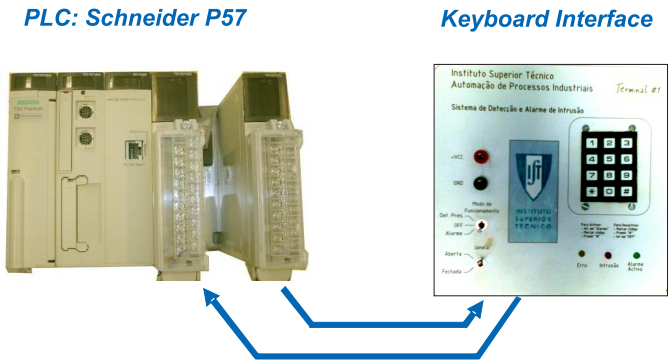


Fig. 1. Alarm system, based in one PLC and one terminal - - *extracted from [3]*

door, an alarm light is turned on for 5 seconds and after that period, a buzzer sound is emitted for 1 second. During the previous introduced periods if the door closes, all the alarms are deactivated and the alarm returns to its presence detector mode.

The second mode works like a typical alarm that is programmed at the end of the day to detect the presence of thieves during the night. If a robber gets into the store, an alarm light is turned on for 5 seconds and after that period, a buzzer sound is emitted intermittent with 1 second ON and 2 seconds OFF. All the warnings are deactivated, and the alarm returns to the Active mode if the keyboard key # is pressed or if the door is closed. A 30 seconds interval of inactivity starts when the alarm is armed, in order to allow that the person who turns on the Active mode leaves the store. After that period, the Active mode is fully functional. The first iteration of the DES synthesis process was performed with the help of the software [13], where it is possible to easily design Petri Nets. Furthermore, the referred software also provides a simulator to test system behavior.

B. Proposed Petri Net Model

Figure 2 introduces the Petri Net that describes the previous alarm. In terms of incidence matrix B, the proposed Petri Net can be described on a 11×29 matrix, due to the fact that the alarm system is represented on a Petri Net with 11 places and 29 transitions. The initial marking of the system is an array with length 11 filled with zeros, except on the first position (that represents the deactivated state) that is equal to one.

On the proposed Petri Net, each place may have one or more outputs associated. For instance, when the working mode of the alarm is set to detect the presence of somebody, and a detection occurs ($P4$), the outputs of the system are {yellow led on; red led on}. On the same logic, when the working mode of the alarm is set to alarm mode, and a detection occurs ($P7$), the outputs of the system are {green led on; red led on}. On different places, the need of turning on the same output occurs. The sharable nature of lights implies there is no need of multiple exclusion on the outputs.

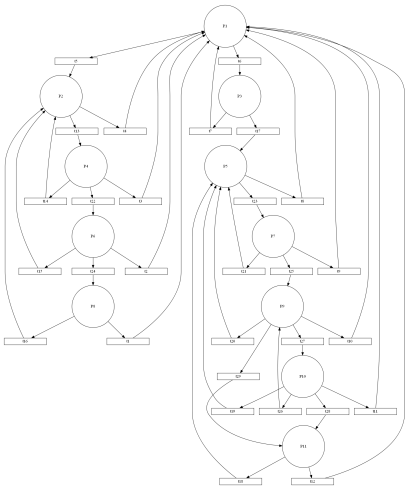


Fig. 2. Proposed Petri Net Model

C. Petri Net Properties Study

The toolbox [22] has been used to obtain the reachability tree and the reachable set of the Petri net (PN) obtained for the alarm system. The toolbox returned a **finite reachable set**. The analysis of the reachable set allows to conclude that the PN **is safe (1-bounded)** and **is conservative**, as no places have more than one mark and all states contain a summation of marks equal to one. **No state covers** another state.

From temporal invariance analysis, **one finds cycles of operation** involving all states. From each state one can reach any other state of the reachable set. By inspection, it is possible to state that the transitions are **all live (level L4)**, since one finds always a sequence of transitions which allows firing a specific transition under liveness study.

D. Conversion to PLC Programming Format

To generate the Structured Text program that later is used on a PLC controller, the same approach as the one in [3] was used meaning that a ST compiler based on [8] was developed in MATLAB. This compiler can be seen in four distinct parts that are the definition of the petri net, the definition of the input mapping, the definition of the output mapping and the ST construction that wraps the results of the previous functions and build a file with the output program.

Very briefly, first this function creates n variables on PLC addresses $\%MW2xx$, where n is the number of places of the specified Petri Net and xx the number associated to each place. All those variables are of type integer, and their initial value is 0 or 1, according with the specified initial marking of the Petri Net. Like is done with all the places, all the transitions timed or untimed, are also declared as variables on PLC addresses $\%MW1xx$ where xx is the number associated to each transition. The value of each one of the "transitions variables", is the combination of the input signals that trigger each transition converted on a integer (`BOOL_TO_INT`), or the timers output flags. Then the Petri Net is encoded in Structured Text, resulting on a block of code with $m\ if$ conditions where each if represents a set $transition - place$. Those conditions

are verified in order to check in which state (place) is the system, on each PLC cycle. Finally, the function creates the block of code that writes the outputs on the output addresses of PLC, according with its actual state.

IV. Discrete Event System Identification

In this Chapter, some definitions and assumptions the algorithm proposed on [6] and the methodology used on this thesis to identify a DES from its Input and Output Signals, are presented

A. Identification Methodology

The identification methodology detailed in this section is proposed in [6], [7], [5] by Paula Estrada-Vargas et al. In the next section is detailed our implementation of the identification methodology.

1) Input and Output Vector Sequence: An **Input Signal** is a sequence of binary vectors

$$\left\{I(i) \in \{0,1\}^{N_I} : i = 1, \cdots, N_S\right\}$$

where $I(i)$ denotes a vector of inputs acquired at scan cycle i, N_I is the number of simultaneous binary inputs and N_S is the number of scan cycles.

An **Output Signal** is a sequence of binary vectors

$$\left\{O(i) \in \{0,1\}^{N_O} : i = 1, \cdots, N_S\right\}$$

where $O(i)$ denotes a vector of inputs acquired at scan cycle i.

An **IO Vector Sequence** is built at the end of each PLC cycle, with the following structure:

$$\left\{w(i) = \begin{bmatrix} I(i) \\ O(i) \end{bmatrix} \in \{0,1\}^{N_I+N_O} : i = 1, \cdots, N_S\right\}$$

where $w(i)$ denotes a vector of inputs and outputs acquired at scan cycle i.

An **Events Vector Sequence** results from the differencing IO vector sequences

$$E(j) = w(i+1) - w(i) = \begin{bmatrix} I(i+1) - I(i) \\ O(i+1) - O(i) \end{bmatrix} =$$
$$= \begin{bmatrix} IE(j) \\ OE(j) \end{bmatrix} \in \{-1, 0, +1\}^{N_I+N_O} : j = 1, \cdots, N_E$$

where j is a selector of scan cycles numbered by i. In particular one may write $i(j)$ to make explicit the selection. In another words, j indicates only the scan cycles where there was a change in the inputs or outputs. Therefore, the number of events N_E is lesser or equal than the number of scan cycles N_S.

Each entry of an input or output events vector can have three possible values that are -1, 0 and 1. When an entry of an input (or output) events vector is equal to 1 $IE_i(j) = 1$ ($OE_i(j) = 1$), a rising input (output) event of the input I_i (output O_i) occurred and is represented by I_i_1 (O_i_1). In a similar way, when an entry of an input (or output) events vector is equal to -1 $IE_i(j) = -1$ ($OE_i(j) = -1$), a falling input (output) event of the input I_i (output O_i) occurred and is represented by I_i_0 (O_i_0).

a) Type of Events: Now that the concept of Events Vector is presented, it is possible to describe the four types of events, presented bellow, that can occur when it is analyzed the subtraction between two consecutive IO vectors.

1) $IE(j) \neq 0$ and $OE(j) \neq 0$
2) $IE(j) = 0$ and $OE(j) \neq 0$
3) $IE(j) \neq 0$ and $OE(j) = 0$
4) $IE(j) = 0$ and $OE(j) = 0$

On the previous conditions, 0 denotes a vector of zeros.

2) Observable and Unobservable Events:

a) An observable event: , in a DES is an event that can be identified by an *outside observer* of the system. More in detail, *outside observer* denotes an acquisition system device which acquires signal samples asking no extra signaling or protocol from the main system. In formal terms, an Observable Event can be represented by:

$$E(j) = \begin{bmatrix} IE(j) \\ OE(j) \end{bmatrix} \text{ s.t. } \begin{array}{l} IE(j) \neq 0 \wedge OE(j) \neq 0, \\ IE(j) = 0 \wedge OE(j) \neq 0, \\ IE(j) \neq 0 \wedge OE(j) = 0. \end{array}$$

The third condition $IE(j) \neq 0 \wedge OE(j) = 0$, represents cases in which an input event does not represent an output change, but may imply output changes in the future.

b) An unobservable event: , in a DES is a event that occur in the system, but that can not be observed by an outside observer of the system behavior. This kind of events are not considered on the identification algorithm that will be introduced later.

In formal terms, an Unobservable Event can be represented by:

$$E(k) = \begin{bmatrix} IE(j) \\ OE(j) \end{bmatrix} \text{ s.t. } \begin{array}{l} IE(j) \neq 0 \wedge OE(j) = 0, \\ IE(j) = 0 \wedge OE(j) = 0. \end{array}$$

The first condition $IE(j) \neq 0 \wedge OE(j) = 0$ represents cases in which an input change provoked a state evolution that is not observed.

3) Untimed DES: An Untimed DES, is a Discrete Event System, in which no event that triggers a state transition depends on time.

4) Assumptions: To solve any engineering problem, assumptions must be declared in order to provide the restrictions of the proposed solution. In that sense, the considered assumptions, to solve the Identification of Discrete Event Systems problem are: (i) There is no counters in this work, so counters will not be considered; (ii) The system never gets blocked; (iii) All IO signals are binary signals; (iv) All the operations that the PLC can perform in the plant are performed during the acquisition; (v) There is no time dependency in this work, so time behavior will not be computed; (vi) Input changes are lost because they occur between consecutive IO vectors; (vii) From the whole system, the only available information is the IO sequence; (viii) The plant is controlled by a single PLC and does not exist multiple scan-cycle periods nor phase shifts;

5) Identification Algorithm: The studied identification algorithm, using conditional probability, creates relationships between the input and output signals of the system analyzing, as stated on [5], the "relative frequency of the output changes with respect to input changes". The output of the algorithm is a Petri Net or fragments of a Petri Net that describes a Discrete Event System, in terms of its input and output signals, where each place (P_i) represents an output signal of the system an the transitions (T_j) are labeled with expressions of system input signals. The general steps of the algorithm are introduced on Table I.

Identification Algorithm - General Steps
Input: IO vectors sequence $w(j)$
Output: Incidence Matrix B and labeling transition function λ
1st Step. IO vectors sequence $w(i)$ pre-process
1.1 Computation of events vectors sequence E and elementary Input and Output events IE and OE
1.2 Computation of Causality Matrices, DM and IM
1.3 Construction of Output Event Firing Functions from previous computed matrices, $\chi(OE_k)$, $F(OE_k)$ and $G(OE_k)$
1.4 Identification of Input events with differed influence, D
2nd Step. From the previous results, generation of outputs B and λ
2.1 For each output event $OE(j) \in E(j)$, computation of its Firing Functions
2.2 If a new transition is detected add it to λ and updates B

TABLE I
ALGORITHM GENERAL STEPS

a) Events Vector Sequence: As introduced previously on this document an Events Vector $E(j)$ is the result of the difference between two consecutive IO vectors, $E(j) = w(i) - w(i-1)$, and can be describe as $E(j) = \begin{bmatrix} IE(j) \\ OE(j) \end{bmatrix}$, where $IE(j)$ and $OE(j)$ are the Input Events vector and the Output Events vector, respectively. Figure 3, shows an example of produced Events Vectors from an IO vector sequence, where $\{W, X, Y, Z\}$ are the output signals and $\{a, b, c, d, e\}$ are the input signals.

$$\begin{bmatrix} W \\ X \\ Y \\ Z \\ a \\ b \\ c \\ d \\ e \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \begin{bmatrix} 1 \\ -1 \\ 0 \\ 0 \\ 1 \\ 0 \\ -1 \\ 0 \\ 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \begin{bmatrix} -1 \\ 1 \\ -1 \\ 1 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 1 \\ 1 \\ 0 \\ 1 \\ 1 \end{bmatrix} \begin{bmatrix} 0 \\ -1 \\ 1 \\ -1 \\ 0 \\ -1 \\ 0 \\ -1 \\ -1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

E(1) E(2) E(3)

Fig. 3. Computed Events Vector - Example

It is also on step 1.1 that the Elementary Input events and Elementary Output events are computed. The set of Elementary Input (Output) Events, is the representation of the input (output) signals that changed between two consecutive I/O vectors, and that correspond to non-zero entries on the Input (Output) Events vector. For each non-zero entry on the Input (Output) Events vector, an entry equals to 1 is a rising input (output) event of the input I_i (output O_i) and is denoted as I_1 (O_0) on the set of Elementary Input (Output) Events. In the same sense, an entry equals to -1 is a falling input (output) event of the input I_i (output O_i) and is denoted as I_0 (O_0) on the set of Elementary Input (Output) Events. On Table II, is shown the computed Elementary Input Events and Elementary Output Events for the Example given on Figure 3.

E(1)	$IE(1) = a\_1, c\_0, e\_1$	$OE(1) = W\_1, X\_0$
E(2)	$IE(2) = b\_1, d\_1$	$OE(2) = W\_0, X\_1, Y\_0, Z\_1$
E(3)	$IE(3) = b\_0, d\_0, e\_0$	$OE(3) = X\_0, Y\_1, Z\_0$

TABLE II
COMPUTED ELEMENTARY INPUT EVENTS AND ELEMENTARY OUTPUT EVENTS

b) Causality Matrices: The occurrence of an input change and an output change in the same PLC cycle, does not mean that those input and output changes are related. In order to distinguish both situations that are event Types 1 and 2, two causality matrices are computed and they are the Direct Causality Matrix, used later to identify events of Type 1 and the Indirect Causality Matrix, used later to identify events of Type 2.

The direct causality matrix DM, relates the occurrence of an output event OE_k and the occurrence of an input event IE_i at the same PLC cycle and in formal terms is expressed as the conditional probability:

$$DM = P(OE_k|IE_i) = \frac{Observ(OE_k, IE_i)}{Observ(OE_k)}$$

where $Observ(OE_k, IE_i)$ is the number of times that Input Event IE_i and the Output Event OE_k are observed on the same Events Vector E. In other words, is the number of times that both Input and Output events occur on the same PLC cycle. The $Observ(OE_k)$, is the number of times that the Output Event OE_k occurs on the Events Vector sequence $E(j)$.

The indirect causality matrix IM, relates the occurrence of an output event OE_k when an input level $I_i = x$ is observed and in formal terms is expressed as the conditional probability:

$$IM = P(OE_k|I_i = x) = \frac{Observ(OE_k, I_i = x)}{Observ(OE_k)}, x \in \{1, 0\}$$

where $Observ(OE_k, I_i = x)$ is the number of times that the output event OE_k is observed when the input signal was $I_i = x$ with $x \in \{1, 0\}$.

c) Output Event Firing Functions: An Output Event Firing Function $\chi(OE_k)$ defines the sufficient conditions for the occurrence of the output event OE_k, and can be mathematically represented by:

$$\chi(OE_k) = G(OE_k) \bullet F(OE_k)$$

, where $G(OE_k)$ is a function of input events that defines the sufficient input events combination $G(OE_k) = \prod DisjE_j = \prod(IE_x \oplus ... \oplus IE_z)$ to produce OE_k and $F(OE_k) \prod = DisjL_j = \prod(IL_x \oplus ... \oplus IL_z)$ is a function of input levels that defines the sufficient input levels ($I_i = x$ with $x \in \{1, 0\}$) to produce OE_k.

d) The function of input events $G(OE_k)$: is computed analyzing the Direct Causality Matrix DM and represents the Events of Type 1, where an input change produces an output change on the same PLC cycle. As stated previously, this function is a conjunction of disjunctions mathematically represented by $G(OE_k) = \prod DisjE_j$ where each $DisjE_j = (IE_x \oplus ... \oplus IE_z)$ merge, for each column of DM, those input events that are different from zero and that all summed together add up to 1. The previous conditions are mathematically expressed as:

1) $DM_{\text{xj}} \neq 0, DM_{\text{yj}} \neq 0, ..., DM_{\text{zj}} \neq 0$
2) $DM_{\text{xj}} + DM_{\text{yj}} + ... + DM_{\text{zj}} = 1$

If no disjunction is detected, the resultant function of input events for the output event OE_k is $G(OE_k) = \epsilon$.

e) The function of input levels $F(OE_k)$: is computed analyzing the Indirect Causality Matrix IM and represents the Events of Type 2, where an output change is not provoked by an input change in the same PLC cycle. Following the same logic as the one followed to compute the function of input events $G(OE_k)$, the same procedure is done on the Indirect Causality Matrix IM in order to determine the function of input levels $F(OE_k)$, but a new condition is added as shown bellow. Unlike the previous function, the only considered columns of IM, are those positions in which the result of $G(OE_k)$ as ϵ.

1) $IM_{\text{xj}} \neq 0, IM_{\text{yj}} \neq 0, ..., IM_{\text{zj}} \neq 0$
2) $IM_{\text{xj}} + IM_{\text{yj}} + ... + IM_{\text{zj}} = 1$
3) $DM_{\text{xj}} \neq 0, DM_{\text{yj}} \neq 0, ..., DM_{\text{zj}} \neq 0$

This last condition is added because some input levels are redundant which means that they do not have influence on the occurrence of an event, so this condition is added to prevent the representation of such a events during the identification process. If no disjunction is detected, the resulting function of input levels for the output event OE_k is $F(OE_k) = (= 1)$.

Figure 6 shows the computed Firing Functions, for the Causality Matrices shown on Figures fig:sec4fig0.2 and fig:sec4fig0.3.

	X_1	X_0	Y_1	Y_0	Z_1	Z_0
a_1	0.1	0.5	0	0	0.7	0
a_0	0	0	0	0.4	0.1	0
b_1	0	0	1	0	0	0
b_0	0.5	0	0	0	0	1
c_1	0	0.5	0	0	1	0
c_0	0	0	0.3	0.6	0	0

Fig. 4. Direct Causality Matrix Example

	X_1	X_0	Y_1	Y_0	Z_1	Z_0
a_1	1	0.1	0.1	0	0.2	0.1
a_0	0	0.3	0.8	0.4	0.1	0
b_1	0.8	0	1	0.5	1	0.4
b_0	0.2	0.6	0.7	0.9	1	1
c_1	0.2	0.7	0.4	0	1	0.8
c_0	0.8	0.9	0.6	0.6	0	0.6

Fig. 5. Indirect Causality Matrix Example

OE_k	**G(OE_k)**	**F(OE_k)**	$\chi(OE_k)$
X_1	(ϵ)	$(a \wedge (b \otimes c))$	$(\epsilon) \cdot (a \wedge (b \otimes c))$
X_0	$(a\_1 \otimes c\_1)$	$(= 1)$	$(a\_1 \otimes c\_1) \cdot (= 1)$
Y_1	(b_1)	$(= 1)$	$(b\_1) \cdot (= 1)$
Y_0	$(a\_0 \otimes c\_0)$	$(= 1)$	$(a\_0 \otimes c\_0) \cdot (= 1)$
Z_1	(c_1)	$(= 1)$	$(c\_1) \cdot (= 1)$
Z_0	(b_0)	$(= 1)$	$(b\_0) \cdot (= 1)$

Fig. 6. Constructed Firing Functions from Examples on Figures 4 and 5

f) Input events with differed influence: During the process of determine the firing functions, some input events are ignored because they never provoke an output change. Those events are the ones in which both lines of the direct causality matrix DM (I_i_1 and I_i_0) are zeros lines. In order to do not ignore this kind of input events, they are stored on the Input events with differed influence D.

g) Output Generation: The second step of the algorithm, can be described as an algorithm as well, as presented bellow on Table III. On the same Events Vector $E(j)$, more than one

Identification Algorithm - Generation of outputs B and λ

Input: IO vectors sequence $w(i)$, elementary Input events IE and elementary Output events OE, Differed Inputs D and Firing Functions $\chi(OE_k)$

Output: Incidence Matrix B and labeling transition function λ

1. Create a matrix with n rows, where n is the number of outputs of the system.
2. For all Events vector of the Events vector sequence $E(j)$, check on the IO vector sequence $w(j)$ and on the Input ($IE(j)$) and Output ($OE(j)$) Events Vector
- If $OE(j) = 0$ and D has elements of $IE(j)$
  - If so, create a new zero transition with a column of zeros on B, and a transition function λ containing all the $IE(j)$ elements that belong to D.
- If $OE(j) \neq 0$
  1. For each output event of $OE(j) = OE_{\text{jx}} \bullet OE_{\text{jy}} \bullet ... \bullet OE_{\text{jz}}$
  2. Compute its respective firing function $\chi(OE_{\text{jx}})$,$\chi(OE_{\text{jy}})$,...,$\chi(OE_{\text{jz}})$
  3. For each computed firing function $\chi(OE_{\text{jk}}) = G(OE_{\text{jk}}) \bullet F(OE_{\text{jk}})$
  3.1. For each $DisjE_i$ check which input event $IE_{\text{ik}} \in IE(j)$ was made true and add it to the auxiliary disjunction variable $DisjE'_i$
  3.2. The auxiliary function of input events $G(OE_k)'$ is the conjuntion off all auxiliary disjunctions $G(OE_k)' = \prod DisjE'_i$
  3.3. The resulting function of input events $G(OE_k)$ is the conjunction of all previous discovered $G(OE_k)'$. $G(OE_k) = G(OE_{\text{jx}}) \bullet ... \bullet G(OE_{\text{jz}})$
  3.4. For each $DisjL_i$ check which input level $IL_{\text{ik}} \in w(j)$ was made true and add it to the auxiliary disjunction variable $DisjL'_i$
  3.5. The auxiliary function of input levels $F(OE_k)'$ is the conjuntion off all auxiliary disjunctions $F(OE_k)' = \prod DisjL'_i$
  3.6. The resulting function of input levels $F(OE_k)$ is the conjunction of all previous discovered $F(OE_k)'$. $F(OE_k) = F(OE_{\text{jx}}) \bullet ... \bullet F(OE_{\text{jz}})$
  4. If the computed firing function $\chi(OE(j)) = F(OE(j)) \bullet G(OE(j))$ does not belong to the stored firing functions λ, create a new transition t_i where $\lambda(t_i) = \chi(OE(j))$, and relate it with the respective outputs
  4.1. For each output event, put 1 in the respective place if the new transition is a rising event, -1 if it is a falling event, and 0 for the output events that are not related with the new transition.

TABLE III
ALGORITHM TO GENERATE THE OUTPUTS

input event $IE_i(j)$ and output event $OE_i(j)$ may occur. On the other hand, an output event can be triggered by several

input events or input levels. Briefly, what the algorithm on Table III does, is check for each Events vector, which input event and which input level triggered each one of the output events presented on $E(j)$. For each computed firing function of each Event Vector$E(j)$, the algorithm checks if the computed transition already was detected. If not, the labeling transition function λ is updated as well as the incidence matrix B.

B. Identification Methodology Implemented

1) Data Acquisition without Embedding Code: In engineering, a black box is a system in which is unknown its internal working. To study this kind of systems some methodologies can be used, but the most commonly used technique is the input-output technique. On this approach, the input and output signals of the system are acquired and studied in order to produce a model that represents the systems in terms of its inputs and outputs. The introduced algorithm does precisely that, using an I/O vector sequence of a system, produces a representation of a Discrete Event System as a Petri Net with its observable behavior.

In the case of DESs, which usualy are implemented on PLCs, there are multiple ways to obtain the input and output signals of the system. One possible approach is using embedded software on PLCs that store all the I/O signals after each cycle but this is a tricky solution, because the acquired values must be stored in memory, which have some disadvantages namely the available storage capacity.

An alternative solution, was addressed on this work. On our approach, the I/O signals are read from the flat cable that connects the controller (PLC) to plant. Like this, there is no need to put heavy code on PLC. Figure 7 shows the connections between the multiple parts. The PLC is connected to the plant, and the signals that are changed between both are read by the signals acquirer. At the end of an acquisition the results are sent to PC, to be processed later.

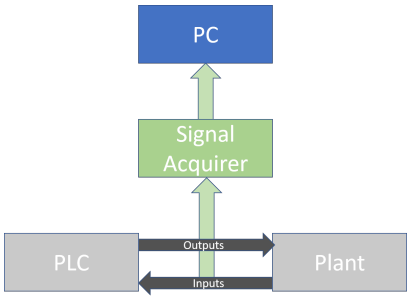


Fig. 7. Data Acquisition without Embedding Code scheme

Nevertheless, this solution also have some difficulties. The scan cycle/scan period of PLC are unknown. To overtake this issue, it is used an high sampling frequency in order to ensure that the acquired input and output signals belong to the same sampling period, that is I(j) and O(j).

2) Signal Acquisition: An acquisition system based on an Arduino and a IO board was developed to acquire I/O signals from a system (controller+plant). In the developed acquisition system, an Arduino Uno was used with an IO board developed by the company InoCAM. The InoCAM IO board, shown in Figure 8, is a board that expands Arduino Uno reading and writing capabilities thanks to its 16 input and 16 output pins and that communicates via SPI (Serial Peripheral Interface) which ensures the highest communication speed between both and a small number of Arduino pins used. Another useful feature of this board, is that its I/O pins can deal with voltage levels of $24V$, which is the input and output voltage of the used PLCs.

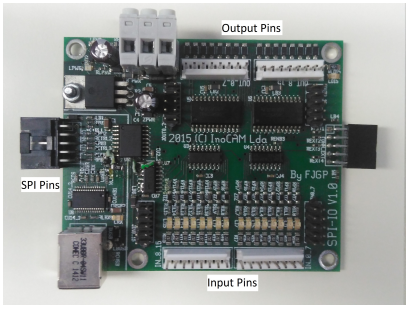
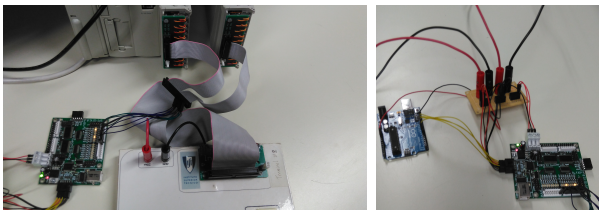


Fig. 8. InoCAM IO Board

To acquire signals from the PLC, each pin of the IO board is connected to one of the conductors of the flat cable that wires the PLC and the Terminal. This technique is as mentioned earlier the authors proposed alternative to the technique used in [6] but requires some cautions while wiring the IO board with the flat cable, and a small adaptation of the program that runs on PLC. Each PLC cycle can be described in three phases that are: (i)input reading; (ii)program execution; (iii)output writing. The signals to be acquired, must belong to the same PLC cycle. More in detail, on each IO vector sequence acquired the output signals result from the input signals.



(a) InoCAM IO Board sniffing PLC IO signals.

(b) InoCAM IO Board wired with Arduino Uno.

Fig. 9. InoCAM IO Board wired with Arduino Uno and PLC

For the signals acquirer, it was also developed an Arduino program, that can be described in two steps that are: (i) Input signals reading; (ii) Storage of read signals. During the acquisition process, in every cycle the data that is received from the IO board is compared with the data that was received in the previous cycle. If different, the ”new” entry is stored, otherwise nothing happens. Figure 10 shows the previous idea, namely the input signals that are stored on Arduino memory, that are marked with red rectangles.

3) Signals Processing: Although in [6] is presented an algorithm that can be used to do the identification of the observable part of the system, no original code is available. In order to study more in depth Estrada-Vargas solution, it was developed a program in Java, that implements the algorithm proposed. The developed program has three classes

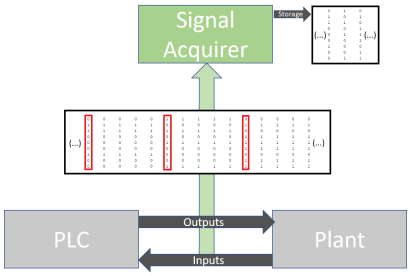


Fig. 10. Strategy to store Signals on Signals Acquirer

named `petrinet.java`, `petrinetgenerator.java` and `CombinationGenerator.java`, where the last one was extracted from [10].

The first class was developed to solve the first part of the algorithm, which means that is in `petrinet.java` class that the events vector sequence E, the elementary events IE and OE, the Direct and Indirect Causality Matrices DM and IM, the Output Event Firing Functions $\chi(OE_k)$, $G(OE_k)$ and $F(OE_k)$ and the Input events with differed influence D are computed/constructed. It is also in this class that a new object of type petrinetgenerator is created and launched in order to compute the observable behavior of the system.

The `petrinetgenerator.java` class implements the second part of the previous algorithm which means that is the one that produces the results of the identification, that are the observable incidence matrix B and the labeling transition function $q(k)$. Two methods, `petrinetgenerator( [args] )` and `generate()`, were developed on this class. The first one, is the class constructor, and simply initializes the class with the provided arguments. The second function, implements the algorithm shown on Table III.

V. Experiments

In order to study the identification algorithm, the alarm presented and described in III, was implemented on a PLC (Controller). The PLC used is composed by four blocks that are: the main module Modicon TSX-57-2634 and is where the program is stored an runs, the power supply module Modicon TSX-PSY-2600, the input analog connection module TSX-DEY-16D2 which provides 16 input pins and output analog connection module TSX-DSY-16T2 which provide 16 output pins. Both input and output modules are connected to a Terminal(Plant) like the one presented on Figure 11 that is used to simulate the inputs and outputs of the alarm.

Previously was introduced that, the acquisition system used to do this experiment, is based on an Arduino Uno and an IO InoCAM board, where the communication between both is established thanks to an SPI interface. This type of connection allows a high speed communication between an Arduino and its peripherals with a small cost on the number of pins used, and it is shown on Table IV where are presented all the pins used on to connect the IO InoCAM board to Arduino Uno.

Beside the connections that are established between the micro-controller and the IO board, the inputs pins of the last one are also connected to the connectors of flat cable that

Fig. 11. System (Controller + Plant) setup

IO InoCAM Board	Arduino Uno
1,3	$+5V$
2	8
4	13
5,8,9,10	GND
6	11
7	12

TABLE IV
IO InoCAM Board - Arduino Uno SPI connections

connects the PLC and the Terminal in order to acquire the IO vector sequences that later will be used to perform the identification of the observable behavior of the system. On Table VI, is presented the mapping between the input pins of the IO InoCAM board, and the connectors of the flat cable as well as the respective PLC address, signal notation and type of signal on PLC perspective. On Table VII, there is a legend for each signal notation introduced on Table VI.

IO Board Pins	Connector	PLC addresses	Signal	Type
0	32	$\%q0.4.12$	WI	Input
1	30	$\%q0.4.10$	DP	Input
2	31	$\%q0.4.11$	AM	Input
3	33	$\%q0.4.14$	$L4$	Input
4	23	$\%q0.4.3$	GR	Output
5	22	$\%q0.4.2$	YE	Output
6	21	$\%q0.4.1$	RE	Output
7	20	$\%q0.4.0$	BU	Output
8	26	$\%q0.4.6$	$C3$	Output

TABLE V
IO InoCAM Board - Arduino Uno SPI connections

Signal Notation	Event	Type
WI	Window Switch	Input
DP	Presence Detector Switch	Input
AM	Active Mode Switch	Input
$L4$	Keyboard Row 4	Input
GR	Green Led	Output
YE	Yellow Led	Output
RE	Red Led	Output
BU	Buzzer	Output
$C3$	Keyboard Column 3	Output

TABLE VI
Signals Notation and respective Input/Output Event

Now that all the hardware setup is presented it is time to present the experience results. As stated on the assumptions

considered to solve the identification problem, all the operation that the controller can perform on the plant should be performed at least once during the acquisition. In other words, this means that all the possible input should be provided to system at least once during the acquisition in order to acquire the "consequences" of each input signal on the system.

After the acquisition, the experimental results were used on the developed identification program. The main outputs of the first step of the used algorithm, are the Direct an Indirect Causality Matrix, the Output Event Firing Functions and the Input events with differed influence, that are shown on Tables VII, VIII, IX and Figure 12 respectively.

	C3_1	C3_0	BU_1	BU_0	RE_1	RE_0	YE_1	YE_0	GR_1	GR_0
L4_1	0.0	0.32	0.0	0.05	0.0	0.04	0.0	0.0	0.0	0.0
L4_0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
AM_1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.33	1.0	0.0
AM_0	0.0	0.51	0.0	0.07	0.0	0.11	0.22	0.0	0.0	1.0
DP_1	0.0	0.08	0.0	0.0	0.06	0.0	1.0	0.0	0.0	0.15
DP_0	0.0	0.0	0.0	0.04	0.0	0.04	0.0	1.0	0.22	0.0
WI_1	0.0	0.0	0.0	0.0	0.37	0.0	0.0	0.0	0.0	0.0
WI_0	0.16	0.0	0.0	0.07	0.0	0.11	0.39	0.0	0.19	0.0

TABLE VII
OBTAINED DIRECT CAUSALITY MATRIX

About the obtained Direct Causality Matrix, an interesting phenomena must be highlighted. The column of the rising edge of the Buzzer, BU_1 is a zeros column. Its meaning, is that the output event BU_1 never happens for none of the input events. This result was expected because the transition that triggers the HIGH state of the Buzzer is a timed transition, and demonstrates that the studied algorithm can not detect timed transitions, as stated on the assumptions.

	C3_1	C3_0	BU_1	BU_0	RE_1	RE_0	YE_1	YE_0	GR_1	GR_0
L4=1	0.0	0.39	0.0	0.07	0.0	0.06	0.0	0.0	0.0	0.0
L4=0	1.0	0.62	1.0	0.93	1.0	0.94	1.0	1.0	1.0	1.0
AM=1	1.0	0.49	0.83	0.76	0.79	0.68	0.0	0.33	1.0	0.0
AM=0	0.0	0.51	0.17	0.24	0.21	0.32	1.0	0.67	0.0	1.0
DP=1	0.0	0.08	0.17	0.13	0.21	0.18	1.0	0.0	0.0	0.15
DP=0	1.0	0.92	0.83	0.87	0.79	0.82	0.0	1.0	1.0	0.85
WI=1	0.53	0.81	1.0	0.93	1.0	0.89	0.39	0.67	0.59	0.85
WI=0	0.47	0.19	0.0	0.07	0.0	0.11	0.61	0.33	0.41	0.15

TABLE VIII
OBTAINED INDIRECT CAUSALITY MATRIX

OE_k	**G**(OE_k)	**F**(OE_k)	$\chi(OE_k)$
C3_1	(ϵ)	$(=1)$	$(\epsilon)\cdot(=1)$
C3_0	(ϵ)	$(=1)$	$(\epsilon)\cdot(=1)$
BU_1	(ϵ)	$(=1)$	$(\epsilon)\cdot(=1)$
BU_0	(ϵ)	$(=1)$	$(\epsilon)\cdot(=1)$
RE_1	(ϵ)	(WI)	$(\epsilon)\cdot(WI)$
RE_0	(ϵ)	$(=1)$	$(\epsilon)\cdot(=1)$
YE_1	(DP_1)	$(=1)$	$(DP\_1)\cdot(=1)$
YE_0	(DP_0)	$(=1)$	$(DP\_0)\cdot(=1)$
GR_1	(AM_1)	$(=1)$	$(AM\_1)\cdot(=1)$
GR_0	(AM_0)	$(=1)$	$(AM\_0)\cdot(=1)$

TABLE IX
OBTAINED FIRING FUNCTIONS

Multiple IO vectors sequence were acquired during this work, in order to reach the optimal IO vectors sequence. More in detail, an IO vector sequence that would generate a Petri Net where all the input signals were labeled to at least one transition. Unfortunately, the optimal sequence was never reached and as such, from all the acquired samples was chosen the one closest to the optimal solution. On the chosen samples, $L4$ is not detected.

```
D = []
```

Fig. 12. Obtained Input Events with differed influence

$$B = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 & -1 & 0 \\ 1 & 0 & -1 & 1 & -1 & 0 & 0 \\ -1 & 0 & 0 & 0 & 1 & -1 & 1 \end{bmatrix}$$

Fig. 13. Incidence Matrix

On Figures 13,14, are presented the obtained Incidence Matrix and the Labeling Transition Function. As expected, any transition is triggered by $L4$ as it can be seen on the Labeling Transition Function, by the reason presented on the previous paragraph. The previous representation of a Petri Net can be converted into a graphical representation of PN, thanks to the software shared on [14], like it was done on the previous section, and that in shown on Figure 15. With this representation it is easier to compare the obtained Petri Net, with the proposed Petri Net of the alarm presented on Chapter III.

It is obvious that both Petri Nets are very different from each other because of the following reasons: (i) On the constructed Petri Net, each place represents an output of the system and each output is represented by one and only one place. On the proposed Petri Net presented on Chapter III, one output could be associated to multiple places and the same place could have associated more than one output; (ii) As a consequence of the previous point, places where no output is associated, are not considered on this solution, that is they does not exist; (iii) The studied system, has timed transitions which violates one of the assumptions on which this identification method is based on. Nevertheless, only studying a system with those transitions would allow drawing conclusions about the behavior of the identification algorithm under such a conditions.

VI. CONCLUSION AND FUTURE WORK

The work described in this thesis is focused in the study of an algorithm of identification of Discrete Event Systems (DESs), from its input and output signals.

First, the concept of Petri Net (PN), which is a form of representation of DESs, was presented. Besides its simple notation and representation, Petri Net properties also provides us a way to study and characterize DESs.

An experimental DES of an alarm system was developed, and designed on a Petri Net. A fast study of the properties of the developed Petri Net was performed. To implement a DES in a PLC, it was used a compiler that converts a Petri Net into Structured Text, that is a PLC programming language. This approach allows the development of PLC programs in less time, and also helps on the debugging process.

A Java program was developed to implement the studied identification algorithm. The identification algorithm, uses statistical methods to establish relations between output changes with input changes. The output of the developed program is a Petri Net, described by its Incidence Matrix and a labeling transition function.

$$\lambda(t_i) = \begin{bmatrix} ((=1)) \cdot (DP\_1 \cdot AM\_0) \\ (WI) \cdot (\epsilon) \\ ((=1)) \cdot (DP\_0) \\ ((=1)) \cdot (DP\_1) \\ ((=1)) \cdot (DP\_0 \cdot AM\_1) \\ ((=1)) \cdot (AM\_0) \\ ((=1)) \cdot (AM\_1) \end{bmatrix}$$

Fig. 14. Labeling Transition Function

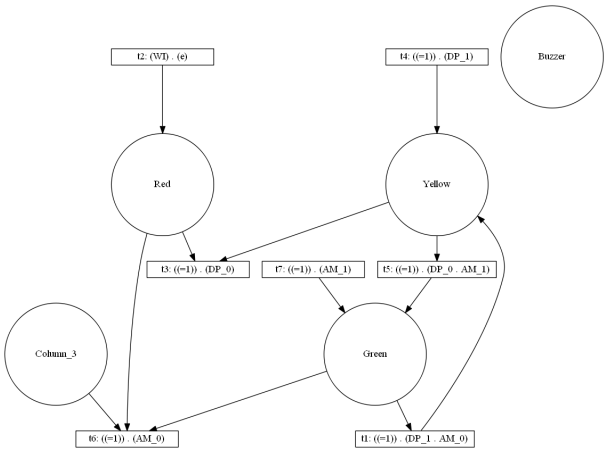


Fig. 15. Petri Net Graphical Representation

In order to study a system developed by us, a signals acquirer based on low cost hardware was built. The hardware monitor, reads PLC signals from outside the PLC, and stores the acquired samples on its memory, to later be used on the developed Java program. In order to acquire consistent samples, the acquisition system works with a high sampling frequency. With this acquisition approach, no embedded software is added on PLC program, allowing small cycle times on the controller and small consumption of controller memory resources.

To test the identification algorithm, an experience was conducted where it was used data, acquired with the built hardware monitor, from the alarm system implemented on a PLC. On this second analysis, it was found some frailties of the proposed algorithm, because the obtained Petri Net is very different from the expected.

Concluding, the studied identification algorithm works well for a small set of DESs with the following characteristics: (i) systems in which each state, is translated into a single output; (ii) systems without timed or conditional events. The synthesized DESs of the alarm system, does not meet the previous requirements, because have timed transitions, and because may exist more that one output associated with a single state. Only few Industrial Processes are included on the set of DESs with the previous characteristics. This means that the studied method only works for a small range of DESs and as such, it is not yet of commercial interest.

To overtake the limitations pointed out to the studied algorithm, future work should focused on two distinct developments/studies. The first work, should focused on the implementation of the algorithm proposed in [7] for the identification of non-observable events on DESs in order to construct better representations of DESs. Future work should also focused on the development of identification algorithms that consider timed and conditional transitions on Discrete Event Systems. The previous algorithms would raise the commercial interest on identification of systems, because most Industrial Processes state evolutions are precisely based on timed and conditional (counters) events.

REFERENCES

[1] M. E. Meda-Campan a. Des identification using interpreted petri nets. In *Proceedings of the International Symposium on Robotics and Automation*, pages 353–357, 1998.

[2] Geoff Cutts and Shaun Rattigan. Using Petri nets to develop programs for PLC systems. In *Application and Theory of Petri Nets 1992*, pages 368–372. Springer, 1992.

[3] Henrique de Almeida-Gonçalves. Monitoring programmable logic controllers. Master's thesis, Departamento de Engenharia Electrotécnica e de Computadores, Instituto Superior Técnico, 2015.

[4] M. Dotoli and A. M.-Mangini M. P.-Fanti. Real time identification of discrete event systems using petri nets. In *Automatica*, pages 1209–1219, 2008.

[5] Ana-Paula Estrada-Vargas. *Black-Box identification of automated discrete event systems*. PhD thesis, Ecole normale supérieure de Cachan, 2013.

[6] Ana-Paula Estrada-Vargas, Jean-Jacques Lesage, and Ernesto López-Mellado. Identification of industrial automation systems: Building compact and expressive petri net models from observable behavior. In *2012 American Control Conference (ACC'12)*, pages 6095–6101, 2012.

[7] Ana Paula Estrada-Vargas, Ernesto López-Mellado, and Jean-Jacques Lesage. Identification of partially observable discrete event manufacturing systems. In *Emerging Technologies & Factory Automation (ETFA), 2013 IEEE 18th Conference on*, pages 1–7. IEEE, 2013.

[8] José Gaspar. Petri net to plc converting. http://users.isr.ist.utl.pt/ jag/course_utils/pn_to_plc/pn_to_plc.html.

[9] José Gaspar. Industrial processes automation (course IST/MEEC). http://users.isr.ist.utl.pt/ jag/courses/api17/api1718.html, 2017.

[10] Michael Gilleland. Combinationgenerator.java. http://www.ccs.neu.edu/home/lieber/courses/csu670/f08/sdg-players/winning-player/src/logic/CombinationGenerator.java.

[11] A. Giua and C. Seatzu. Identification of free-labeled petri nets via integer programming. In *Proceedings of the 44th IEEE Conference on Decision and Control, and the European Control Conference (CDC-ECC '05)*, 2005.

[12] E.M. Gold. Language identification in the limit. In *Information and Control*, pages 447–474, 1967.

[13] GRES Research Group. Iopt tools. http://gres.uninova.pt/IOPT-Tools/login.php.

[14] Gary Wilson Jr. Drawing petri nets. http://thegarywilson.com/blog/2011/drawing-petri-nets/.

[15] J. Kella. Sequential machine identification. In *IEEE Trans. on Computers*, pages 332–338, 1971.

[16] S. Klein. *Identification of discrete event systems for fault detection purposes*. PhD thesis, Ecole Normale Superieure de Cachan, 2005.

[17] S. Klein, L. Litz, and J.J.-Lesage. Fault detection of discrete event systems using an identification approach. In *Proceedings of the 16th IFAC World Congress*, 2005.

[18] Mark Minas and Georg Frey. Visual PLC-Programming using Signal Interpreted Petri Nets. In *American Control Conference, 2003*, 2002.

[19] James L. Peterson. *Petri Net Theory and the Modeling of Systems*. Pretice-Hall, Austin, Texas, 1981.

[20] Carl Petri. *Communication With Automata*. PhD thesis, Darmstadt University of Technology, 1962.

[21] L.P.J Veelenturf. Inference of sequential machines from sample computations. In *IEEE Trans. on Computers*, pages 167–170, 1978.

[22] Martina Svdov Zdenk, Martina Svádová, and Zdeněk Hanzálek. Matlab toolbox for petri nets. In *22nd International Conference ICATPN 2001*, pages 32–36, 2001.