ICARSC 2021, IEEE Int. Conf. on Autonomous Robot Systems and Competitions, pp143-148
doi: 10.1109/ICARSC52212.2021.9429807
https://ieeexplore.ieee.org/document/9429807

Validation of Discrete Event Processes implemented on PLCs based on Petri Nets

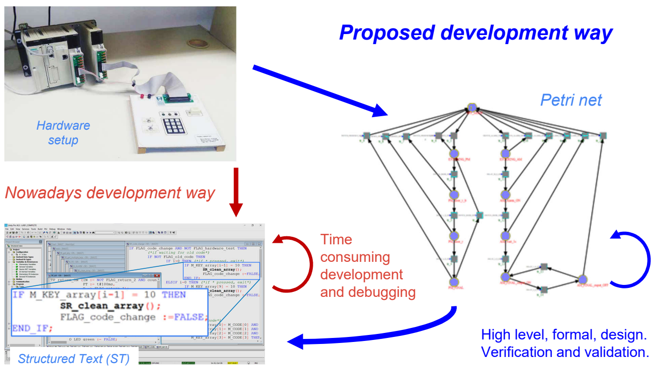
Hugo Conde Barroso
Institute for Systems and Robotics, LARSyS
Instituto Superior Técnico, University of Lisbon
Email: hugo.barroso@tecnico.ulisboa.pt

José António Gaspar
Institute for Systems and Robotics, LARSyS
Instituto Superior Técnico, University of Lisbon
Email: jag@isr.tecnico.ulisboa.pt

***Abstract*—Discrete Event Systems (DES) are commonly found as supervisors in industrial applications, implemented by Programmable Logic Controllers (PLC). Modeling DESs with IOPT Petri Nets (IOPT PNs), a class of PNs extended with input-output, allows translating the models directly to PLC programs when the PNs are bounded.**

Validation is required to assess the PN-based PLC programming tools are error-free. Coverability trees based on the node dominance concept are proposed to consider just DESs with finite reachable sets. Operation cycles, and the associated sequences of transitions, are used to validate the produced code by assessing whether the PLC code reaches all possible states.

Promising results were obtained in practical validation cases of the PLC code production, by the exhaustive test of reachable states determined from the PN representing the industrial process (DES). Moreover, the proposed methodology showed that it is possible to anticipate the detection of design problems and study the effects of restrictions imposed by hardware.


Proposed development way
Petri net
Hardware setup
Nowadays development way
Time consuming development and debugging
High level, formal, design. Verification and validation.
Structured Text (ST)

```
IF M_KEY_array[i-1] = 10 THEN
    SR_clean_array();
    FLAG_code_change :=FALSE;
END_IF;
```

Fig. 1. Automated conversion from high level PN to ST as a faster alternative to direct PLC programming. Validation is important to assess the conversion as well as the PN design.

I. Introduction

Programmable Logic Controllers (PLCs) are the most common devices for integrating and controlling industrial processes. However, despite the widespread usage based on standard programming languages, it is still time consuming their direct programming. We focus on a more convenient way to create PLC programs: high level design with Petri nets (PNs) followed by an automatic conversion to a PLC language, namely Structured Text (ST), as shown in Figure 1.

Unbounded PNs imply infinite markings and therefore cannot be implemented in PLCs. We propose a methodology based on the reachability and coverability concepts [1] to determine a PN can be implemented in a PLC.

In addition, we propose methodologies to validate the conversion as well as to validate the PN design, making use of the PN properties to automatically generate testing sequences.

More in detail, we propose as a first step determining a PN has a finite number of possible states. Starting from an initial state, one may take all possible state transitions, therefore creating a tree, whose leaves indicate no more possible state changes (deadlocks) or duplicate states already explored. This tree can still have an infinite number of nodes. However, one may use a special symbol indicating infinite countings to still obtain a finite tree, which can be built in finite time. One obtains the so-called coverability tree, which immediately indicates bounded PNs and suggests sequences of state transitions to visit all possible states. Cassandras and Lafortune's comprehensive introduction to discrete event systems [1] (2^{nd} edition, 2008, following the seminal book [2]) contains a detailed description of the coverability tree algorithm that is the base of our DES/PLC validation proposal.

In the vein of industrial-systems development-tools, Pais, Barros and Gomes [3] proposed a new PN class based on Place/Transition nets and well-known concepts from Interpreted PN: the IOPT PN class [4]. This PN class provides support for the specification of input and output events. In [5], is developed software to convert PNs to a hardware description language (VHDL) which inspired the creation of a toolchain converting PNs to PLC programs [6]–[8]. In this work the toolchain is complemented with testing (validation) tools.

II. Background

A. Petri Nets

A Marked PN, C is formally defined as a five-tuple

$$C = (P, T, A, w, \mu_0) \tag{1}$$

where P is the finite set of places defined as $P = (p_1, p_2, ..., p_n), n \in \mathbb{N}$, T is the finite set of transitions defined as $T = (t_1, t_2, ..., t_m), m \in \mathbb{N}$, (A, w) represent the arcs and μ_0 is the initial state of the PN given by the markings of all the places. A PN state is defined as $\mu = [\mu(p_1), \mu(p_2), ..., \mu(p_n)] \in \mathbb{N}^n$. About the arcs, (A, w), A denote the finite set of arcs from places to transitions and from transitions to places in the graph defined as $A \subseteq (P\text{x}T) \cup (T\text{x}P)$, and w is the weight function on the arcs defined as $A \rightarrow 1, 2, 3, ...$; one assigned weight for each arc in A. The matrix comprised of all $w(t_j, p_i)$ values is called the postconditions matrix D^+ and the matrix comprised of all $w(p_i, t_j)$ is defined as preconditions matrix D^-, with $D^+,\ D^- \in \mathbb{R}^{nxm}$. For simplicity, a marked PN shall henceforth be referred to as just a PN.

The PN algebraic representation is done in matrix form by the *Incidence Matrix* $D \in \mathbb{R}^{nxm}$, which in turn is obtained from the weight function w as $D_{ij} = w(t_j, p_i) - w(p_i, t_j)$, where $i = 1, ..., n$ and $j = 1, ..., m$. The dynamic of PNs is based on two concepts, *Enabled Transition* and *Firing Rule*.

a) Enabled Transition: For a given marking, a transition t_j is said to be *enabled* if its preconditions are verified, i.e., $\mu(p_i) \geq w(p_i, t_j), \forall (p_i, t_j) \in w$.

b) Firing Rule: Only enabled transitions may fire. Firing is described by the PN dynamics, also known as state transition mechanism, and is based on the equation $\mu' = \mu + D \cdot q(j)$ where q is the firing vector and $q(j)$ a vector representing the firing of transition t_j.

B. IOPT Petri Nets

IOPT PNs [3] are based on Place/Transition nets and concepts from Interpreted PNs. They allow the specification of models with input and output signals and events, namely the association of input events to transitions and output events to places, simulating the readings of sensors and manipulation of actuators. The input and output signals guide the controller through each execution step by defining the system current state while the input or output events are associated to changes in input or output signals.

The IOPT PN dynamics are similar to the PN dynamics. A transition has to be both *enabled* and *ready* to be firable. A transition is ready when the associated input events happen. After a transition fires, the marking changes according to the PN dynamics. The marked places will trigger associated output events.

C. Reachability

The set of all states reachable by a PN from the initial state μ is called reachable set $\mathscr{R}$, which can be graphically represented by a reachability tree. Depending on whether the PN is bounded or unbounded, the reachability tree can be finite or infinite, correspondingly. When the reachable set is finite it may be represented by the finite reachability tree. When the reachable set is infinite the reachability tree becomes infinite. There exists a way of representing an infinite reachability tree in a finite form by introducing the infinity symbol ω and the notion of *node dominance* (section II-E).

The *reachability problem* for a PN with initial marking μ_0 consists of deciding that a marking μ can be reached from μ_0. This problem was first proposed by Karp and Miller [9] within the scope of Vector Addiction Systems, but left unsolved. Esparza's research on decidability issues for PNs [10] states that, after an incomplete proof by Sacerdote and Tenney [11], decidability of the problem was established by Mayr in his seminal STOC 1981 work [12]. The algorithm uses a structure called *regular constraint graphs* and is based on conditions given by Presburger's Arithmetic. The proof was then simplified by Kosaraju [13] by disposing of the complicated tree constructions used by Sacerdote and Tenney, and Mayr, introducing a "more general model o VASS's" known as *Generalized Vector Addiction Systems with States* (GVASS). Further refinements were made by Lambert [14], where he completely suppressed the use of Presburger's Arithmetic.

D. Coverability Tree

In 1969, Karp and Miller [9] introduced the *rooted tree* $\mathscr{T}(\mathscr{W})$, for any vector addition system $\mathscr{W}$, and the *infinity symbol* ω terminology to help represent an infinite reachability set $\mathscr{R}(\mathscr{W})$ in a finite form.

Later in 1981, Peterson [15] used ω to represent "a number of tokens which can be made arbitrarily large", that can be thought of as *infinity*, and described an algorithm to reduce the infinite reachability tree to a finite representation. Peterson chose to name both the finite and the infinite reachability tree as reachability tree.

Note that seeking the finite version is always possible, meaning the algorithm always terminates, as proven in [15] who based the proof on [16] and [9].

Finally, 1993, Cassandras and Lafortune [1] introduced the notation of *node dominance*, which the previous authors also used but did not label, to present the technique of the *coverability tree*. The authors named *coverability tree* as the finite representation of the infinite reachability tree, which contains the infinity symbol ω.

Often overlooked, node dominance makes possible the decision of whether a reachable set is finite or infinite.

E. Node Dominance

Let any state $\mu = [\mu(p_1), \mu(p_2), ..., \mu(p_n)]$. Consider states μ and μ' belonging to the coverability tree and n the total number of places in the PN. If there is a node μ' on the path from the root node to μ such that

(i) $\mu(p_i) \geq \mu'(p_i), \forall i = 1, ..., n$ (state μ covers state μ')

(ii) $\exists i : \mu(p_i) > \mu'(p_i)$, i=1,...,n (there exists at least one place of μ that has more tokens than the corresponding place of μ')

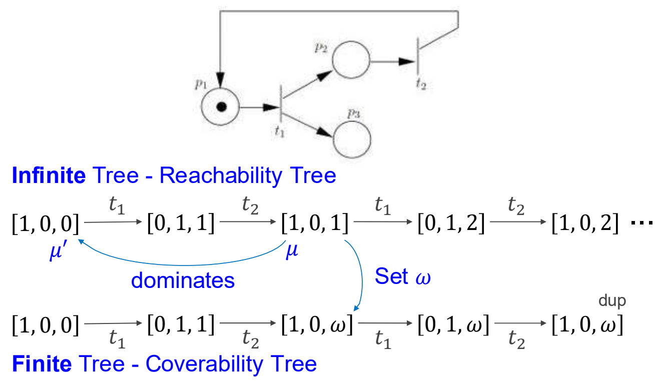
then $\mu >_d \mu'$, i.e., "μ dominates μ'" and set $\mu(p_i) = \omega\ \forall i$ that verify (ii). Example in Figure 2.

III. Implementation and Validation of Discrete-Event Processes

Consider a PLC program represented by a PN with I/O interacting with (supervising) the world. A PN typically has inputs at the transitions and outputs at the places.

A. DES to PLC Conversion Toolchain

The DES to PLC conversion toolchain chosen as the object of analysis started to be developed in [6] and improved in [8]. We assume that the modelling of the DES is made by a system



p_1
p_2
p_3
t_1
t_2
Infinite Tree - Reachability Tree
$[1,0,0] \xrightarrow{t_1} [0,1,1] \xrightarrow{t_2} [1,0,1] \xrightarrow{t_1} [0,1,2] \xrightarrow{t_2} [1,0,2] \cdots$
μ'
μ
dominates
Set ω
dup
$[1,0,0] \xrightarrow{t_1} [0,1,1] \xrightarrow{t_2} [1,0,\omega] \xrightarrow{t_1} [0,1,\omega] \xrightarrow{t_2} [1,0,\omega]$
Finite Tree - Coverability Tree

Fig. 2. Node dominance example.

designer. More in detail, we assume it is used an IOPT PN [1] which properly models the system and that the PN meets the format required by the toolchain. Currently, the DES to PLC conversion toolchain [2] provides, additionally for this work, a controlled simulation environment and includes the automatic generation of testing sequences of events.

B. Finite Reachable Set

Unbounded PNs imply infinite reachable sets, therefore do not allow exhaustive testing of the reachability set. We start by adding a decision method to reject unbounded PNs to the PLC-code production toolchain (Figure 3). Given a bounded PN, we propose a method to automatically generate testing sequences.

As referred, we want to consider just bounded PNs since in those cases one finds finite reachable sets and can automatically generate sequence of events for testing the system. More in detail, we want to start by finding an algorithm to decide whether a PN is bounded. We start by the reachability problem introduced in section II-C.

Given a PN C with initial marking μ_0, the reachability problem consists of finding if a state μ is in the reachable set, i.e. $\mu \in \mathscr{R}(C, \mu_0)$. This problem was shown to be decidable [10], [12], [14]. While approaching the reachability problem in [9], Karp and Miller introduced the *node dominance* concept to build a coverability tree. The coverability tree has a finite number of nodes, but may contain the symbol ω indicating unbounded places on the PN. A coverability tree not containing ω symbols indicates a bounded PN (see section II-D).

Our proposal, complementing the DES to PLC conversion toolchain , involves two additional steps. The first step is to include a coverability tree construction algorithm in the toolchain to reject unbounded PNs, which are associated to coverability trees containing ω. This is introduced in section II-D.

The second step is to extract the finite sequence of transitions for testing, given a finite reachability tree, i.e. a

[1]IOPT tools [5] http://gres.uninova.pt/IOPT-Tools, accessed 2021-02-22.
[2]http://www.isr.tecnico.ulisboa.pt/%7Ejag/tools/pn2plc/

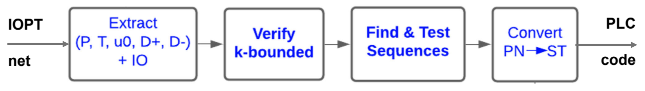


Fig. 3. IOPT PN to ST Converter. A decision method to reject unbounded PN is added as well as to automatically generate testing sequences.

coverability tree which has no ω symbols as the PN did not require its use. We find operation cycles, avoiding therefore deadlock cases, and use sequences of transitions to assess whether the process code effectively reaches all possible states. This methodology is developed in section III-C.

C. Generation of Event Sequences

In a reachability tree are shown the transitions next to each branch. To determine the sequence of transitions to be triggered in order to evolve the PN from one state to another, just follow the path from the first to the second.

Doing this process of extracting a sequence of transitions from the initial state to all its duplicates in the tree, it is possible to obtain a sequence of transitions describing an exhaustive test. Then, using the input and output components of IOPT PN, we replace each transition in the sequence with the associated events by building a table of sequential events. For such a table we use the notation table U_t (see table example in (2)).

The sequentiality is given by the first column of the U_t table, which represents the moments in time where events will occur to make transitions ready, guiding the controller through its reachable states, thus allowing validation of the industrial process implementation by the PN.

D. Extension of Petri net models to include Time

To better model the interaction of the controller with a real system, it is necessary to introduce the concept of time in the toolchain internal PN model. Time allows recreating real hardware operations. Depending on the time scales used, we can find out about the existence of critical races when there is a rapid evolution of states, as well as an unexpected evolution of the controller when the input signals vary rapidly, due to input bouncing, for example. Timed PNs [17], well known extensions of PN, do this using timers in the form of controller inputs, associating them with each transition. The timeouts of the timers dictate times of events enabling the firing of transitions.

IV. Experiments

In this section are combined the tools for creating DES supervisors running on PLCs with the tools for validating the created DES supervisors. The main objective of the experiments is the assessment of the proposed validation tools applied to testing the created DES supervisors and identifying possible design issues.

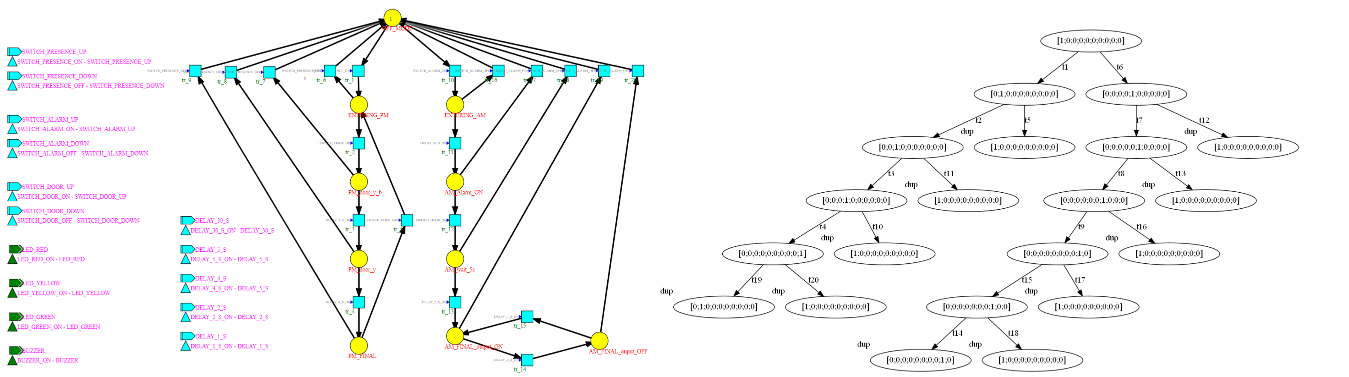


Fig. 4. PN edited with IOPT Tools to control the alarm (left). Coverability tree for the proposed alarm PN (right). Duplicate states are marked ”dup” on the top left corner.

A. Setup, Alarm System

The experimental setup is a PLC based mockup of an alarm system, composed by three inputs - a presence switch, an alarm switch and a door switch - and by four output components - a red LED, a yellow LED, a green LED and a buzzer (see Figure 7(a)).

The IOPT PN in Figure 4 implements three alarm operation modes: (i) OFF, (ii) Presence Detection, where the alarm informs a door was opened by a person entering the room, (iii) Alarm mode (Intrusion Detection), where whenever a door is opened, the alarm must be sound. The PN is 1-bounded and has just one token in all states, i.e. whichever the state, only one place has one mark and all other places have zero marks.

The coverability tree in Figure 4 is obtained by the coverability tree algorithm implemented. Note there are no ω in the tree and thus it is a coverability tree representing a finite reachability set, with just ten possible states.

B. Reach All Possible States

In this use case we are assessing if the toolchain performs an error-free conversion. We feed generated sequences of events to the IOPT PN and evaluate the resulting state. We follow the methodology detailed in section III-B: (i) computing the coverability tree (ii), verifying the alarm controller IOPT PN is bounded, (iii) using the event sequence automatic generation tool to generate the exhaustive test that makes the alarm controller reach all of its states (see section III-C) and finally (iv) simulating the controller with the test sequence.

We validate the conversion, i.e. consider it is done correctly, by comparing the results of the test with those expected from the knowledge PN based controller responses to events. If a difference is found, it is due to the conversion tool. Otherwise, the conversion is considered validated.

The PN contains only 1 token, so the current state corresponds to the marked place. Confirmation that all states are reached is achieved if all places are visited. Figure 5 shows that effectively all states are reached given the generated sequence of events.

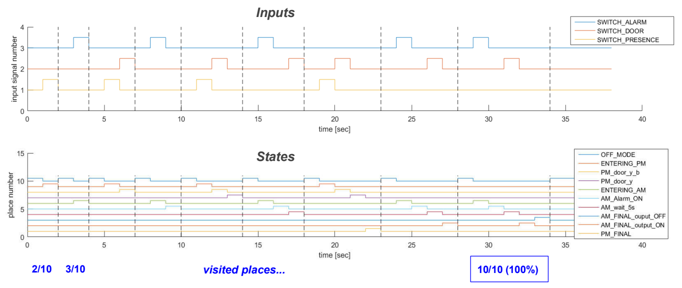


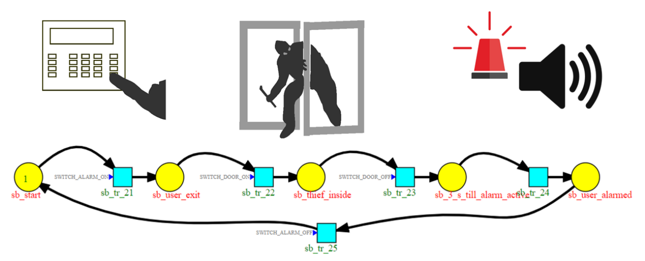
Fig. 5. Response of PN (bottom) to exhaustive test sequence of inputs (top) extracted from its coverability tree.

C. PLC-DES interaction with World-DES

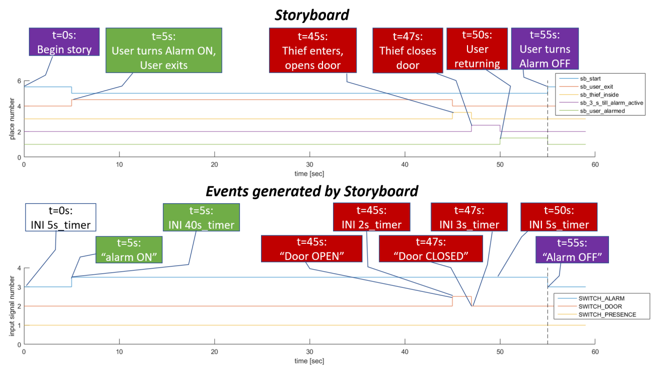
Let PLC-DES denote the alarm controller and World-DES denote the alarm hardware components connected to the PLC. Validating the PLC-DES code while including the to/from world interaction usually requires too much (exhaustive) human intervention. In addition, the World-DES may introduce functional deadlocks or even induce unexpected behaviour in the PLC-DES. Alternatively, we focus on specific tests based on storyboards, representing the World-DES, covering expected usages.

1) Closed Loop Simulation, PLC and World Interaction: The storyboard is created as an IOPT PN that tells a story from which the sequence of events is extracted (Figure 6). For example, in our setup, the storyboard describes a typical correct functioning of the alarm in Alarm mode. The extraction of the test sequence from the storyboard follows the general steps of finding the associated coverability tree and converting the sequence of transitions into a table U_t:

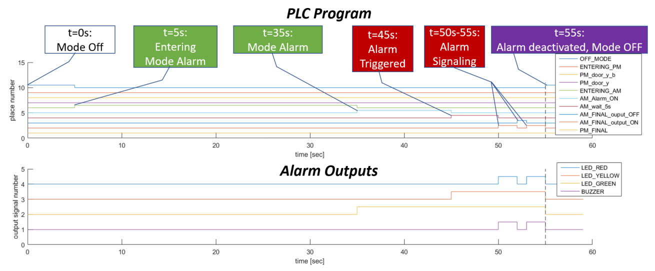
$$U_t = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 5.0010 & 0 & 1 & 1 & 0 & 1 & 0 \\ 5.0030 & 0 & 1 & 1 & 0 & 1 & 0 \\ 45.003 & 0 & 1 & 1 & 0 & 0 & 1 \\ 45.005 & 0 & 1 & 1 & 0 & 0 & 1 \\ 47.005 & 0 & 1 & 1 & 0 & 1 & 0 \\ 47.007 & 0 & 1 & 1 & 0 & 1 & 0 \\ 50.007 & 0 & 1 & 1 & 0 & 1 & 0 \\ 50.009 & 0 & 1 & 1 & 0 & 1 & 0 \\ 55.009 & 1 & 0 & 1 & 0 & 1 & 0 \end{bmatrix} \quad (2)$$



(a) Storyboard written as an IOPT PN.



(b) Storyboard as a sequence of events and expected controller response.



(c) Controller state and output evolution given the input events.

Fig. 6. Storyboard showing a typical correct functioning of the alarm in Alarm mode. Alarm PLC program (Figure 4) driven by a storyboard showing the typical correct functioning of the alarm in Alarm mode.

where the first column indicates time of events and the next columns show input signals (event sequences) that are used to fire PN transitions.

Figure 6 shows the results of simulating the PLC-DES with the World-DES storyboard. The state evolution corresponds to the expected correct functioning of the alarm in Alarm mode. Therefore, the implementation of the controller from the PN is validated in the sense the controller correctly performed the alarm functioning in Alarm mode.

2) Validation using the PLC Network Interface: The PLC programming software, Unity Pro, by Schneider Electric, includes a standard network interface (MODBUS), which allows injecting input signals (events) to the PLC and receiving outputs of the PLC. Thus, the terminal shown in Figure 7(a), connected to the PLC input/output modules by a flat cable, can be replaced by a software interface.

More in detail, the MODBUS (software) terminal allows injecting inputs to the PLC given a table U_t, as the one in (III-C), but now with a number of digital inputs required by the terminal. As referred, the terminal also receives the outputs of the PLC.

We run the DES to PLC conversion toolchain ST compiler to convert the PLC-DES to ST code, which is placed in a Unity Pro section. Then connect to the PLC, transferring the project, and run the project. Finally, in the terminal, we load table U_t (2) and start the injection of inputs protocol.

Figure 6 shows the results of the experiment. We observe a correct injection of inputs and coil writing, following the desired storyboard. In addition, we observe a matching behaviour of the PLC-DES to the expected one. Again, the implementation of the controller from the PN is validated in the sense the controller correctly performed the alarm functioning in Alarm mode.

D. Effects of Hardware Constraints on PN Designs

In case a DES controller developer does not account for PLC and/or real system hardware constraints the implementations can fail validation tests. PLC programs, which are based on scan cycles, may fail to recognize fast input changes (short pulses) depending on their duration and the state of the PLC scan cycle. On an opposite case, if long enough, pulses associated to input bouncing may be accepted incorrectly by the PLCs.

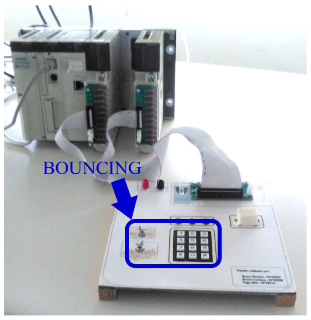
The proposed methodology allows searching for bouncing issues, by creating event sequences to test the PN against bouncing on all inputs (Figure 7). To assess whether the methodology correctly detects the existence of the problem, we looked for a situation in the alarm that was destabilized by the bouncing effect. Such is a situation in presence detection mode, when finalising the detection of an individual. Bouncing occurs on the door switch and a new detection is initiated incorrectly, causing a double detection. We apply a debouncing solution directly to the PN to also check if the methodology effectively detects when the problem is overcome.

Figure 7(b) shows the response of the original PN to the effect of bouncing on the door switch. The first bounce of the signal is incorrectly accepted as an input, reflecting in the detection of a second individual incorrectly. We can validate that the methodology effectively detects problems caused by the bouncing effect.

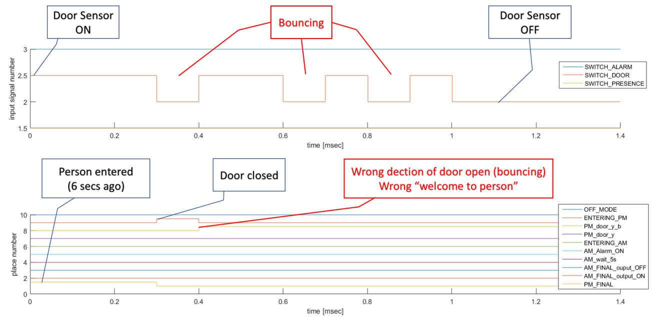
Observing Figure 7(c), showing the response of the debounced PN to the effect of bouncing on the door switch, we see the debouncing mechanism successfully handles the bouncing effect. Only after the door switch input signal stabilizes at the OFF position for 2ms does the PN evolve to the state of waiting to detect a new individual. We validate that the methodology effectively detects when the situation is solved.

V. Conclusion and Future Work

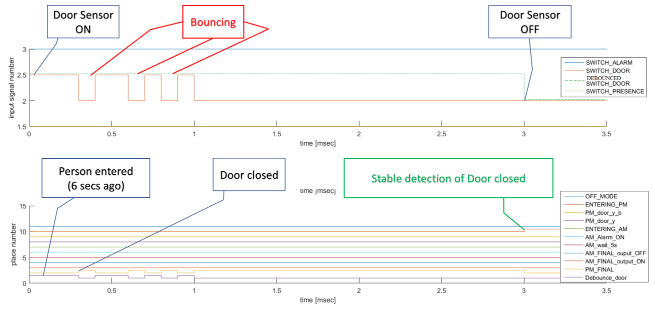
This paper is focused on the production of discrete-event process supervisors implemented in PLCs. In particular is considered the assessment of error-free PN to PLC-code



(a) PLC based setup.



(b) DES/PLC-code from PN without debouncing.



(c) DES/PLC-code from PN with debouncing.

Fig. 7. (a) Alarm setup. The IO terminal allows human interaction. Events can also be injected with the network interface. Bouncing happens on the switches, but exists mostly in push-buttons. (b) Response of PN to bouncing on door switch input. (c) Response of debounced PN to bouncing on door switch input.

conversion. A PLC code production toolchain is used and further developed, and are proposed tools that check whether or not IOPT PNs are correctly translated to PLC ST programs.

One challenge of doing validation by reachability analysis is the so called state space explosion. We consider (accept) only bounded PNs based on the construction and analysis of the coverability tree. An automatic generation of event sequences, based on the coverability tree, is proposed for driving the PN through all reachable states. Time is introduced in the toolchain to allow testing the PLC/DES supervisor code on expected PLC/DES uses.

Experiments with a PLC based alarm system allowed the validation testing of both the DES to PLC conversion toolchain simulation environment, the IOPT PN to PLC code conversion and the PN design. Furthermore, it was addressed the problem of bouncing that arises from the connection of digital systems to noisy, transient-prone, *bouncing* inputs. A method to test the toolchain robustness to bouncing was proposed, as well as a debouncing solution to be implemented directly on the IOPT PN.

Future work, studying state explosion by using symbolic model checking may open more ways for the verification and validation. More in detail, the *Cone of Influence* [18] may identify which part of a DES model is relevant for the evaluation of the given requirement and in that way augment to a larger class the PNs considered in our work that can be implemented in PLCs. Safety studies on robot collision-avoidance systems are a promising application topic.

Acknowledgment

This work was supported by the Portuguese Foundation for Science and Technology (FCT) with the LARSyS - FCT Project UIDB/50009/2020.

References

[1] C. G. Cassandras and S. Lafortune, *Introduction to discrete event systems*. Springer Science & Business Media, 2008.
[2] C. G. Cassandras, *Discrete event systems: modeling and performance analysis*. CRC Press, 1993.
[3] R. Pais, S. P. Barros, and L. Gomes, "A tool for tailored code generation from petri net models," in *2005 IEEE Conference on Emerging Technologies and Factory Automation*, vol. 1, 2005, pp. 8 pp.–864.
[4] L. Gomes, J. P. Barros, A. Costa, and R. Nunes, "The input-output place-transition petri net class and associated tools," in *2007 5th IEEE International Conference on Industrial Informatics*, vol. 1. IEEE, 2007, pp. 509–514.
[5] F. Pereira and L. Gomes, "Automatic synthesis of VHDL hardware components from IOPT Petri net models," in *Annual Conf. of the IEEE Industrial Electronics Society (IECON)*. IEEE, 2013, pp. 2214–2219.
[6] H. Gonçalves, "Monitoring programmable logic controllers," Master's thesis, Instituto Superior Técnico, 2015.
[7] R. Feio, J. Rosas, and L. Gomes, "Translating iopt petri net models into plc ladder diagrams," in *2017 IEEE Int. Conf. on Industrial Technology (ICIT)*. IEEE, 2017, pp. 1211–1216.
[8] R. Reis, "PLC programming based on applied games," Master's thesis, Instituto Superior Técnico, 2019.
[9] R. M. Karp and R. E. Miller, "Parallel program schemata," *Journal of Computer and system Sciences*, vol. 3, no. 2, pp. 147–195, 1969.
[10] J. Esparza and M. Nielsen, "Decidability issues for petri nets," *Petri nets newsletter*, vol. 94, pp. 5–23, 1994.
[11] G. S. Sacerdote and R. L. Tenney, "The decidability of the reachability problem for vector addition systems (preliminary version)," in *Proc. of the 9th annual ACM Symp. on Theory of Computing*, 1977, pp. 61–76.
[12] E. W. Mayr, "An algorithm for the general petri net reachability problem," in *Proceedings of the thirteenth annual ACM symposium on Theory of computing*. ACM, 1981, pp. 238–246.
[13] S. R. Kosaraju, "Decidability of reachability in vector addition systems (preliminary version)," in *Proceedings of the fourteenth annual ACM symposium on Theory of computing*, 1982, pp. 267–281.
[14] J.-L. Lambert, "A structure to decide reachability in petri nets," *Theoretical Computer Science*, vol. 99, no. 1, pp. 79–104, 1992.
[15] J. L. Peterson, *Petri net theory and the modeling of systems*. Prentice Hall PTR, 1981.
[16] M. H. T. Hack, "Decision problems for petri nets and vector addition systems," MIT Project MAC, MAC-TM 59, Cambridge, MA, 1975.
[17] J. Wang, *Timed Petri nets: Theory and application*. Springer Science & Business Media, 2012, vol. 9.
[18] D. Darvas, B. F. Adiego, A. Vörös, T. Bartha, E. B. Vinuela, and V. M. G. Suárez, "Formal verification of complex properties on plc programs," in *Int. Conf. on Formal Techniques for Distributed Objects, Components, and Systems*. Springer, 2014, pp. 284–299.