applied sciences

MDPI

Article

Petri Net-Based Semi-Compiled Code Generation for Programmable Logic Controllers

Igor Azkarate [1,*], Mikel Ayani [2], Juan Carlos Mugarza [1] and Luka Eciolaza [1]

[1] Faculty of Engineering, Mondragon Unibertsitatea, Goiru 2, 20500 Arrasate-Mondragón, Spain; jcmugarza@mondragon.edu (J.C.M.); leciolaza@mondragon.edu (L.E.)
[2] Research and Development, Simumatik AB, Box 133, 54123 Skövde, Sweden; mikel.ayani@simumatik.com
* Correspondence: iazkarate@mondragon.edu

Abstract: Industrial discrete event dynamic systems (DEDSs) are commonly modeled by means of Petri nets (PNs). PNs have the capability to model behaviors such as concurrency, synchronization, and resource sharing, compared to a step transition function chart or GRAphe Fonctionnel de Commande Etape Transition (GRAFCET) which is a particular case of a PN. However, there is not an effective systematic way to implement a PN in a programmable logic controller (PLC), and so the implementation of such a controller outside a PLC in some external software that will communicate with the PLC is very common. There have been some attempts to implement PNs within a PLC, but they are dependent on how the logic of places and transitions is programmed for each application. This work proposes a novel application-independent and platform-independent PN implementation methodology. This methodology is a systematic way to implement a PN controller within industrial PLCs. A great portion of the code will be validated automatically prior to PLC implementation. Net structure and marking evolution will be checked on the basis of PN model structural analysis, and only net interpretation will be manually coded and error-prone. Thus, this methodology represents a systematic and semi-compiled PN implementation method. A use case supported by a digital twin (DT) is shown where the automated solution required by a manufacturing system is carried out and executed in two different devices for portability testing, and the scan cycle periods are compared for both approaches.

Keywords: Petri nets; programmable logic controllers; process modeling; digital twin

Citation: Azkarate, I.; Ayani, M.; Mugarza, J.C.; Eciolaza, L. Petri Net-Based Semi-Compiled Code Generation for Programmable Logic Controllers. *Appl. Sci.* **2021**, *11*, 7161. https://doi.org/10.3390/app11157161

Academic Editors: Luis Gomes and João Paulo Barros

Received: 30 June 2021
Accepted: 30 July 2021
Published: 3 August 2021

1. Introduction

Discrete event dynamic systems (DEDSs) are widely present in industrial manufacturing processes. A DEDS is a dynamic, asynchronous system, where state transitions are initiated by events that occur at discrete instants of time [1]. They are usually modeled by finite state automata with partially observable events together with a mechanism for enabling and disabling a subset of state transitions [2]. The following are some examples:

- Clients arriving or leaving a waiting queue to be attended at a desk.
- A waiter serving two customers, noting requests and serving them in any order.
- Synchronization of traffic lights in road intersections, after expiration of each state temporization (Figure 1).
- Flexible and/or discrete part manufacturing systems, those concerning with robots and programmable machines behaving with concurrent evolution, synchronization and/or shared resources features. In the frame of this set a use case is analyzed in Section 3.

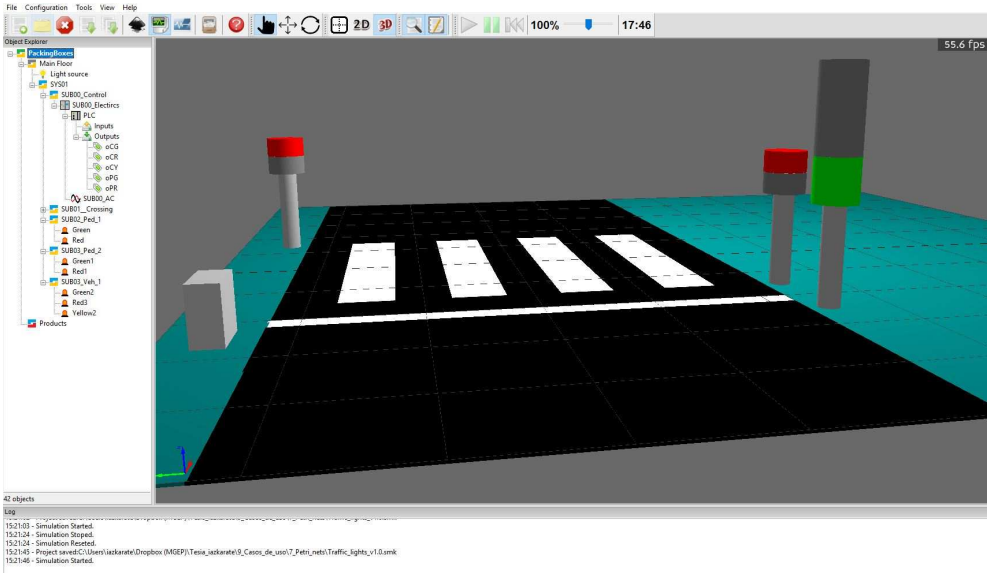

Figure 1. Emulated traffic lights synchronization.

1.1. PN-Based DEDSs Modeling

Petri nets (PNs) are a powerful method to model and control such DEDSs as they have the capability to represent systems characterized as being concurrent, asynchronous, distributed, parallel, nondeterministic, and/or stochastic [3]. A PN is a graphical tool because it combines visual communication aid (block diagram) to which tokens are added to simulate the dynamics and concurrent activities of systems. It is also a mathematical tool because of algebraic models related to its behavior. Ref. [4] considers PNs as suitable formalism to model and visualize the behavior of DEDSs, noting that they have been applied successfully to several problems in different sectors, such as manufacturing. Thus, a PN represents an effective method [5] for both the design and implementation of this type of systems [6]. However, there is not a widely adopted and effective systematic way to implement a PN in a programmable logic controller (PLC), and so it is very common the implementation of such a controller outside a PLC in some external software that will communicate with the PLC.

A PN is composed by its structure and interpretation [7], and its dynamics. In terms of net structure [4] describes a PN graph as a bipartite graph that contains two types of nodes: places (identified by $p \in P$ and represented by circles) and transitions ($t \in T$, bars). These two types of nodes are connected through directed and weighted arcs (Figure 2). Ref. [7] denotes a Place/Transition (P/T) net as $N = <P,T,\text{Pre},\text{Post}>$, where P and T are the sets of places and transitions, and Pre and Post are the $|P| \times |T|$ sized, natural valued incidence matrices. For instance, $\text{Post}[p,t] = w$ means that there is an arc from t to p with weight (or multiplicity) w. When all weights are one the net is *ordinary*. A marking is a $|P|$ sized and natural valued vector. A marked net, P/T system or PN is a pair $<N,m_0>$, where m_0 is the initial marking. A transition t is enabled at marking m if $m \geq \text{Pre}[P,t]$; its firing yields a new marking $m' = m + \text{C}[P,t]$, where $\text{C} = \text{Post} - \text{Pre}$ is the token − flow matrix of the net. The set of all reachable markings, or reachability graph (RG), from m, is denoted by $\text{RG}(N,m)$. C and m_0 corresponding to the example are as follows:

$$C = \begin{array}{c} \begin{array}{ccccc} t_0 & t_1 & t_2 & t_3 & t_4 \end{array} \\ \begin{pmatrix} 1 & -1 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 \\ -1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -1 & 1 \\ 0 & 0 & 0 & 1 & -1 \\ 1 & 0 & 0 & -1 & 0 \\ 0 & -1 & 0 & 0 & 1 \end{pmatrix} \end{array} \begin{array}{c} p_0 \\ p_1 \\ p_2 \\ p_3 \\ p_4 \\ p_5 \\ p_6 \end{array}$$

$$m_0^t = \begin{array}{c} \begin{array}{ccccccc} p_0 & p_1 & p_2 & p_3 & p_4 & p_5 & p_6 \end{array} \\ \begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 1 \end{pmatrix} \end{array}$$

PN dynamics is characterized by the marking, i.e., tokens (dots) distribution among places, and its evolution. Marking progress depends on the controlled firing of transitions i.e., their enabling and the compliance of their associated firing conditions.

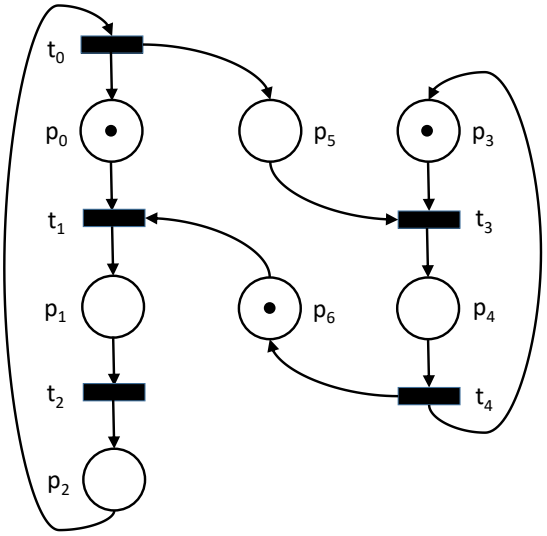

Figure 2. PN structure for traffic lights synchronization.

Table 1. Interpretation to be added to the PN structure shown in Figure 2.

Place	**Action**	**Description**
p_0	*oCR*	Red light for vehicles.
p_1	*oCG*	Green light for vehicles (and timer activation).
p_2	*oCY*	Yellow light for vehicles (and timer activation).
p_3	*oPR*	Red light for pedestrians.
p_4	*oPG*	Green light for pedestrians (and timer activation).
p_5	-	Pedestrian green light activation token.
p_6	-	Vehicle green light activation token.
Transition	**Condition**	**Description**
t_0	*tVY*	Vehicle yellow light timer expiration.
t_1	-	Direct.
t_2	*tVG*	Vehicle green light timer expiration.
t_3	-	Direct.
t_4	*tPG*	Pedestrian green light timer expiration.

In terms of structural validation, [3] defines basic properties to be fulfilled by a PN, such as its liveness, cyclicity, and boundedness. The existence of the aforementioned formal

mathematical definition (C, m_0), as well as formal validation methods [3,8,9], make it possible to detect, in the modeling phase, (i) blocking states (markings), (ii) bounded marking, and (iii) initial marking reachability. Software tools are available on the market with capabilities such as graphical edition of the net, marking evolution simulation, property analysis, and PN formal definition generation through matrix representation.

Finally, PN interpretation is defined by events modeled in transitions, which leads to marking changes, and actions associated to places. It can consist of: (i) time (constant, random) for performance evaluation; (ii) condition action for control. Table 1 shows the interpretation corresponding to the example.

1.2. PN Implementation in PLC

The PLC is the most widely used controller in industrial environment. It is a control device for DEDSs based on information from process and operator [10]. Its use is due to several characteristics that differentiate it from a conventional computer [11]: (i) cyclic execution of the control program, (ii) reliability, (iii) adaptation to an industrial environment with electrical noise, vibrations, extreme temperatures and humidity, and (iv) easy maintenance.

PN implementations in PLCs depend on how the logic of places and transitions is programmed. There is not any broadly used systematic method of implementation. Therefore, there may be as many PN implementation methods as programmers, and any improvement in the PN involves code regeneration. There have been some attempts towards a systematic PN implementation method; however, although these attempts propose an ordered method to program the logic of places and transitions, they have not been widely adopted.

1.3. GRAFCET: A Particular Case of PN

A step transition function chart or GRAphe Fonctionnel de Commande Etape Transition (GRAFCET) diagram is a widely used graphical tool for PLC programming [10]. GRAFCET is a safe PN [12], i.e., the limitation property is guaranteed being the possible marking of any place in the net 0 or 1. That is, $\forall p \in P, m(p) \leq 1$. Its implementation in PLC has traditionally been done by means of a boolean variable for each step [13,14], with its status evaluated at every scan cycle of the device. Currently, many development environments include sequential function chart (SFC), a graphic modeling and description method for sequential automation systems, suitable for GRAFCET and standardized in IEC 61131-3 (Figure 3).

1.4. PN vs. GRAFCET Comparison

It is worth noting the suitability of using a PN based approach to control a DEDS compared to a GRAFCET. Thus, PNs offer capacity to model behaviors such as concurrency, synchronization, and resource sharing, among other advantages [4]. There are certain functionalities that, being implicit in a PN structure, should be coded for a GRAFCET. First, for managing non boolean marking of a PN, counters must be incorporated into GRAFCET, since it is intended for single marking of steps, just binary places in PN (marking vector includes only 0 or 1 values). Next, while features such as shared resources or synchronization are integrated into a PN structure and can be validated directly, their GRAFCET modeling requires a development based on internal variables. Finally, the existence of a formal mathematical definition (C, m_0) and subsequent formal validation methods for a PN structure, enable the detection of possible net deadlocks. Model validation abilities are much more powerful than those proposed for GRAFCET, mainly related with divergence-convergence managing [15].

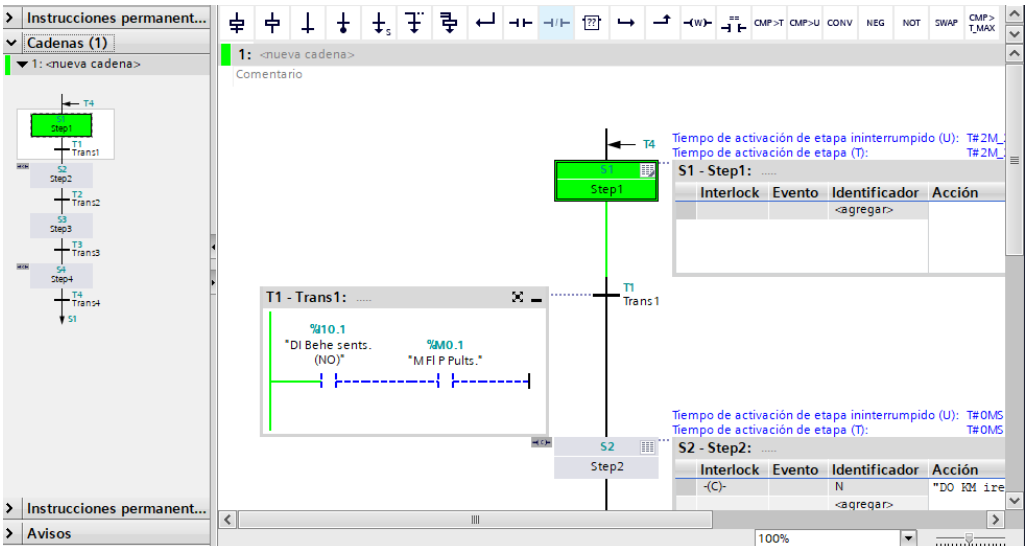

Figure 3. SFC-based programming in PLC development environment.

1.5. Related Works and Proposal

Several works introduce PN implementation methodologies into standard PLC languages and in an interpreted basis, i.e., with structure and marking management integrated into the code itself. Ref. [16] presents a method for generating ladder diagrams (LDs), which are widely used for discrete event control, that enables faster development, debugging, and re-engineering. It is based on PN models that are valid for formal system analysis, as well as their ability to verify liveness and cyclicity. It also addresses the need for building PNs in a simple way, as well as to be able to evaluate their properties and simulate token evolution. Ref. [17] points out the limitation of LD-based implementations in a context in which systems are of increasing complexity, and the need for tools for control, analysis, evaluation, and simulation of systems, presenting PN as an integrated solution and making a survey about approaches for its conversion to LD. Ref. [4] defines a set of rules for implementing LDs and overcomes merely intuitive ad hoc developments. The review by [18] references these PN to LD conversions, among other implementations of logic controllers. In addition, literature shows the use of other PLC programming languages as well. References evolve from the mentioned LDs [19–22] and/or instruction lists [23,24] to more recent proposals in standard programming languages [25] or structured text (ST) [26], for portability to different PLCs. In a recent paper, Ref. [27] proposes editing both net structure and its interpretation in a proprietary graphical environment and automatically converting them into PLC code.

Existing PN implementations in PLCs present some weaknesses. Thus, they are fully interpreted developments, totally dependent on the application and the programmer. The coding workload is significant and error-prone. Moreover, any modification in net structure or interpretation involves code rewriting/regenerating. In terms of PN validation, it is conventionally performed at the end of development, on the equipment already assembled, on a digital twin (DT) that emulates it, or by forcing input values and observing output behaviors in the PLC development environment itself.

This paper proposes a methodology to implement a PN within a PLC in a simple way, making use of the mentioned formal mathematical definition of a PN, that makes possible to automatically validate part of the PLC program, minimizing coding workload and, consequently, error-prone points. Thus, a previously validated PN matrix representation will be directly copied into a PLC device data memory. A fixed program blocks will manage the marking of the PN and the programming workload will be limited to PN interpretation, i.e.: assigning conditions (inputs queries) to transitions and actions (outputs writing) to places. In this way, projects can be carried out and commissioned in a very

efficient way. This represents a semi-compiled PN implementation instead of an exclusively interpreted one.

The proposed PN implementation approach can contribute to the state-of-the-art in several aspects (Figure 4):

- The PN model is constructed, evaluated and simulated in a simple way, which is demanded by authors such as [16]. A visual and intuitive software tool is used to code, generating structural data that are managed by fixed program blocks, and validate part of the implementation. While the tool presented in [27] is exclusively for PN structure and interpretation editing, in this work it is proposed the use of a commercial software that provides structure editing, simulation, and validation.
- Coding workload is reduced to net interpretation: assignment of actions to places and conditions to transitions of the net, thus to process outputs and inputs, respectively. Exclusively the mentioned program section remains to be manually coded and validated.
- Any change in net structure or initial marking is direct and impacts only the controller's data area, never the program, which remains unmodified. In contrast, literature shows code generation methodologies, transforming complete PNs (structure and interpretation) to PLC standard programming languages. Ref. [28], for instance, automates and accelerates PN to LD conversion supported by characteristic matrices, but without automatically generating them or using validation rules, and any change involves all code to be generated and manually validated.

In the context of this work, simple PNs are used, with event/action interpretation. We limit ourselves to basic structural analysis without any previous performance evaluation by means of other temporal and/or stochastic interpretation.

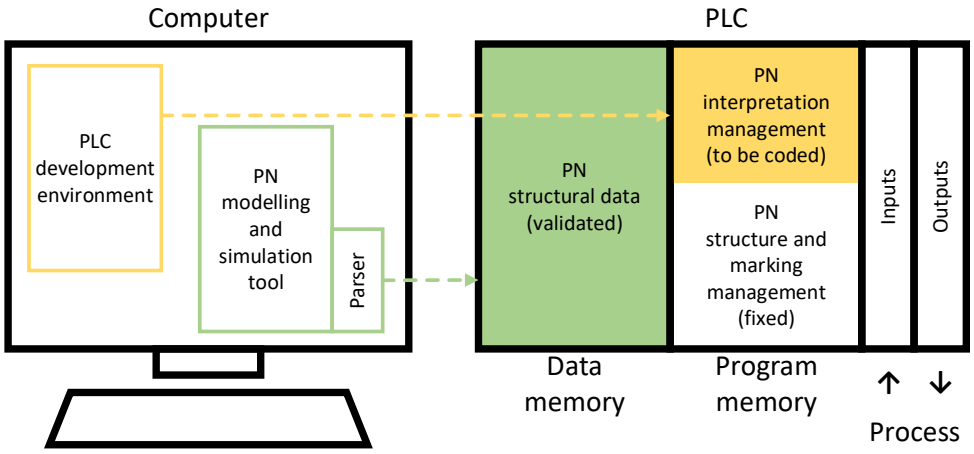

Figure 4. Proposed PN implementation in PLC.

2. Methodology

This section shows the proposed algorithm and its implementation methodology.

2.1. Algorithm for PN Marking Management

The basis of the proposal is a PN-based algorithm with compliance with the following specifications:

- An appropriate program block structure, with a fixed part (related to PN structure and marking evolution) and a customizable or to-be-programmed part (PN interpretation), properly differentiated.
- Apply PN marking evolution rules.
- Coding in ST, due to (i) simple portability regardless of target device vendor, (ii) operating matricially element by element.

- Variables required for algorithm proper operation already pre-declared, and PN structure matrix representation not declared, for its direct transfer by a parser application.

Note that the only task related to PN structure is transferring its matrix representation (C, m_0) to PLC data memory. PN interpretation has to be coded, differing upon particular application.

In compliance with the specifications described above, the algorithm kernel implements the transition firing and marking updating rules for a general PN, as detailed in Algorithm 1. Provided with any particular PN pre-incidence and post-incidence matrices and initial marking vector contained in a PLC data block, this algorithm allows token movements among places managing, thus allowing transition enabling, firing and resulting marking computation. It is coded in ST, in order to operate more simply with matrices and for an easier portability to other manufacturers' devices. The program block execution flow is described below.

Algorithm 1: Transition firing and marking updating for a general PN.

$i = 1, ..., m$; where m is the cardinality of P; $\wedge$ $j = 1, ..., n$; where n is that of T.
marking initialization: $x(p_i) := x_0(p_i)$;
incidence matrix initialization: $Inc(p_i, t_j) := Post(p_i, t_j) - Pre(p_i, t_j)$;
while *TRUE* **do**
  create list of enabled transitions $E(t_j)$:
    a transition is enabled $\Longleftrightarrow x(p_i) \geq Pre(p_i, t_j), \forall p_i \in I(t_j)$;
    $I(t_j)$ is the set of input places of a transition $t_j \in T$.
  compute conditions list: $Cond(t_j), \forall t_j \in E(t_j)$;
  create list of firable transitions $F(t_j)$:
    a transition is firable $\Longleftrightarrow Cond(t_j) = TRUE, \forall t_j \in E(t_j)$;
  if $\exists t_j / F(t_j) \neq 0$ **then**
    decide on which transition $t_j \in F(t_j)$, if any, should be fired;
    if *it is decided to fire* $t_j \in F(t_j)$ **then**
      compute new marking reached: $x'(p_i) := x(p_i) + Inc(p_i, t_j)$;
      update marking: $x(p_i) := x'(p_i)$;
      update actions: $\forall p_i / x(p_i) > 0 \Longrightarrow Acc(p_i) := TRUE$;
    end
  end
end

Startup

Only once, and before the continuous scan cycle execution, initial marking value is given to the current one, and incidence matrix is calculated from the pre-incidence and post-incidence ones.

Enabling

This program block determines transitions enabled by current marking. Marking vector is compared with each of the columns of the pre-incidence matrix to determine whether the marking enables the corresponding transition. If so, it is annotated in an array of enabled transitions. This function will not be executed again until some change has occurred in current marking, that is, until some transition has already been fired.

Firable

It is checked whether currently enabled transitions are likely to be fired. According to net interpretation, it evaluates fulfillment of transitions corresponding to firing conditions, listing them in an array. Subsequently, it is evaluated whether they are liable to be fired, reflecting it in another array.

Make a Decision

A rule is implemented in this block. Note that a firable transition does not have to be fired at that instant. Firing could be postponed. It may not even occur, if the firing of another transition causes it to become no longer enabled and firable. This property of PNs is particularly important and interesting in the modeling of sequencing problems: order control in the sequence of transitions firing. A function is introduced for the decision of the transition to be fired among the liable to be fired ones, maybe being in conflict. In this implementation a simple decision rule is applied: the last of those present in the list of firable is fired. Once the selection has been made, it is possible to update the net marking after the transition is fired.

Marking Update

This is a calculation operation. As a consequence of the firing of previously selected transitions among firable ones, new net marking is computed. It implies updating it by adding to marking vector the incidence matrix column corresponding to the fired transition.

Actions

This block updates actions related with places, as net interpretation determines, according to achieved new marking.

2.2. Implementation Procedure

The proposed procedure for implementing a PN from an algorithm that meets the specifications described, suitable for any process and PLC, consists of the following phases, as shown in Figure 5. It contains the phases of the implementation, only one of them being a source of errors in the code: that of net interpretation programming. The code reaches this phase mostly fixed and validated, pending manual addition/validation of the mentioned code lines.

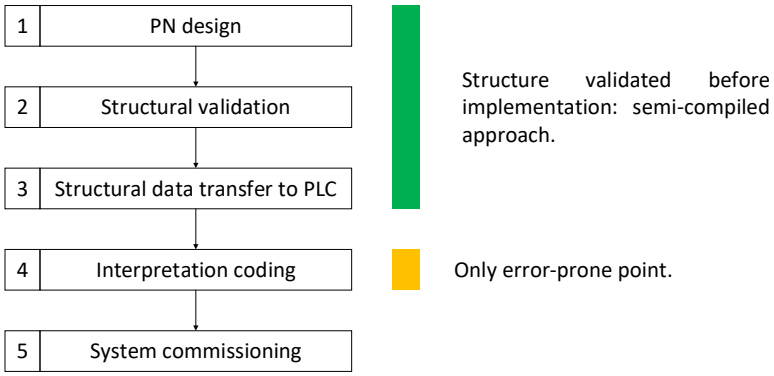

Figure 5. PN implementation methodology phases.

Note that specifications definition, characterizing the desired operation and identifying key signals, is a previous work to be done, as in any automation project, and independent of the proposed methodology. It involves project creation in the development environment, hardware definition, program block addition and transfer of those that are fixed and application independent, and declaration and initialization of variables (internal and I/O). Furthermore, if virtual commissioning (VC) of the automated solution is planned, a process DT must be designed and developed.

2.2.1. PN Design

Once specifications are defined, the PN structural design is carried out for its subsequent analysis. Interpretation is added, associating conditions and actions with PN structure transitions and places, respectively.

2.2.2. Structural Validation

There are software tools available on the market which provide PN editing capabilities and token moving among places simulation [29]. Platform Independent Petri net Editor 2 (PIPE2) [30,31], for example, also owns validation and performance analysis features. Net invariants computing allows for testing structural properties such as boundness, liveness, and deadlock freeness. Finally, a file in HTML format with PN structure's matrix representation can be generated, as well as validation process results.

Note that 100% automatic validation is not possible. For example, a necessary condition is tested for liveness, but it does not fully guarantee this PN property. PIPE2 includes the capability to generate a RG (see Figure 6 as an example). Liveness can be fully verified by detecting a unique strongly connected component in the RG which includes the whole set of transitions T.

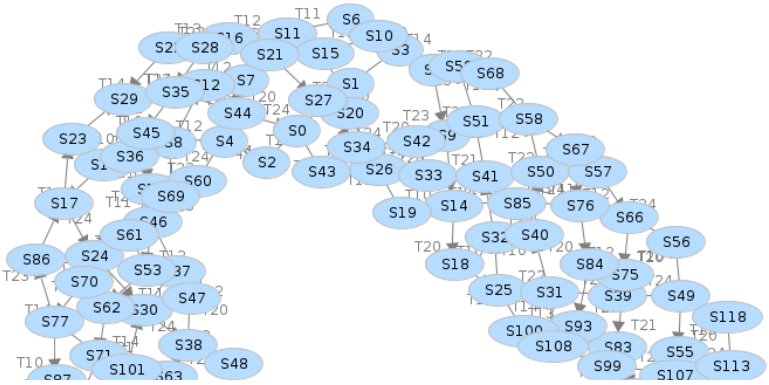

Figure 6. Detail of a RG generated by PIPE2.

2.2.3. Structural Data Transfer to PLC

From the file created in the previous step, the matrix representation is transferred to a data block in PLC memory. A parser can be used to adequate the information provided by the PN editing, analysis and simulation tool to the PLC's syntax.

2.2.4. Interpretation Coding

The corresponding code for each case has to be developed. Conditions and actions are edited in program memory, according to particular net interpretation.

2.2.5. System Commissioning

This step implies verifying the correct behavior of the automated process, according to specifications.

3. Use Case

This section describes the use case in which the experimentation was carried out. The required resources are listed, the process to be automated is described, and the implementations performed are explained.

3.1. Manufacturing Process under Study

3.1.1. Description

The system under study is the one shown in Figure 7. It has two machines (M_1 and M_2), in which parts have to be worked consecutively. A temporary buffer with limited (7) capacity and a robot (R) complete the manufacturing cell. M_1 and M_2 can be in one of the following five states: waiting to be loaded with a part, being loaded, operating, waiting to be unloaded, being unloaded. The purpose of R is to load/unload parts to/from the stations before/after the operations they perform. Being single and shared, it can be in one of these states: waiting for servicing, or executing one of four movements: M_1 station

loading with a part from input; M_1 unloading, a part is driven to intermediate buffer; M_2 station loading with a part from the buffer; M_2 unloading, a part is driven to system output.

It was considered to be an interesting case for a PN-based implementation, due to the use of a limited-capacity intermediate buffer, and the existence of a shared resource, R, that executes one movement each time.

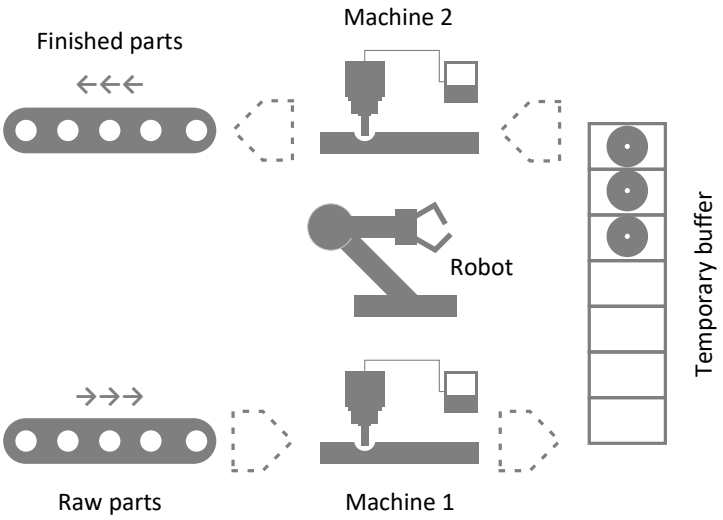

Figure 7. Schema of a manufacturing cell.

3.1.2. Emulation through a DT

As the process under analysis was not available, two possibilities for software validation were considered: the conventional one in the development environment itself, and the VC. In the first case, since all the signals coming from the process have to be reproduced, a good knowledge of the system to be automated is required. Some scenarios may remain unreproduced, leaving the control software not fully validated. Faced with this situation, it was decided to use emulation for VC. Consequently, a DT of the process to be automated was designed, as shown in Figure 8, in which only the part controlled by the PLC is emulated, which is the one to be analyzed in this work. It was connected in a hardware-in-the-loop (HIL) setup and by means of Object Linking and Embedding (OLE) for Process Control Unified Architecture (OPC UA) [32] to the physical PLC, configured DT as a client and controller as a server. Note that the devices used include, as most of the latest generation ones, OPC UA server capability.

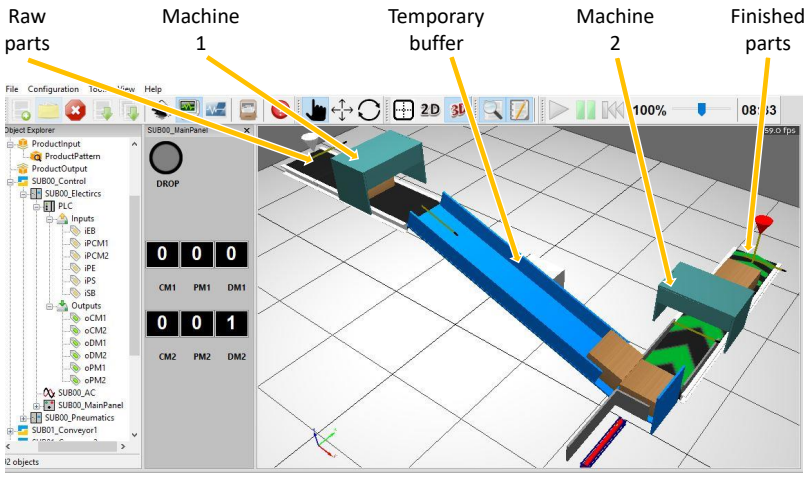

Figure 8. Emulation of the PLC-controlled part of the process.

3.2. Resources

For each of the necessary resources, it is specified what was employed in this particular use case.

- An industrial system and process emulation tool or DT. It can support the whole process of development, simulation and validation, connected to the emulated PLC in a software-in-the-loop (SIL) configuration or to the physical controller in HIL.
  For this study Simumatik3D V1.0.3 (S3D) [33] was used for VC, connected to a physical PLC.
- A PLC programming environment, as well as the device itself and its emulation tool. Two control devices from different vendors were required, with their respective development platforms:
  - SIMATIC S7-1500 CPU 1512C-1 PN of Siemens, with TIA Portal V15.1 for coding and PLCSIM Advanced V2.0 SP1 for emulation. The algorithm was available for these tools.
  - CPU NX102-1020 of Omron, with Sysmac Studio 1.25 for coding and emulation.
- A PN modeling and analysis tool.
  The above-mentioned PIPE2 tool was used for (i) PN structure edition, (ii) PN liveness and/or boundedness properties necessary conditions fullfilment verification, and (iii) PN representative matrices generation.
- In addition, an application was developed for converting the data generated by PIPE2 (HTML format) into a text format suitable for insertion in the variable declaration modules of various development environments, including TIA Portal and Sysmac Studio.

3.3. Manufacturing System Automation

The automated solution for the robotic cell under study was implemented by means of PN-based approach for two PLCs from different vendors. In addition, each of the controllers was programmed following other similar, fully interpreted solutions, in order to compare the scan cycle time required: two PN-based, using LDs and ST, and a GRAFCET-based one, exclusively for Siemens, by means of SFC. This subsection describes the work performed.

3.3.1. PN Implementation by Means of the Proposed Approach

For both implementations carried out on PLCs from different vendors, the procedure described in Section 2.2 was applied and the following starting points were assumed:

- Project created in the development environment, and the PLC hardware used properly configured.
- In device program memory:
  - Created program block structure.
  - Coded the content of fixed blocks, corresponding to PN marking evolution management. In Siemens environment, as a result of the first algorithm development and subsequent testing. For Omron, as a result of code transfer. It should be noted that, for a first implementation in a new development environment, compilation errors may occur as a result of the platform-specific editing procedure. Often, PLC program editing task needs variable name recognition, thus it should be written literally at its very first appearance in a programming module. In that case, simple copy-paste does not work properly.
  - Blocks related to PN interpretation, empty.
- In device data memory:
  - Variables corresponding to inputs and outputs, and specific to algorithm, declared.
  - Variables associated with PN structure were not declared, as they were transferred from the design and simulation package through proprietary software.

Defined by the specifications already outlined in the process description, the PN model development (Section 2.2.1) resulted in the one shown in Figure 9. It consists of 13 places and 10 transitions. It has two branches, left and right, corresponding to M_1

and M_2 machines, respectively, with places representing their loading, processing, and unloading. The model is completed with M place, the shared-use robot, and two places, H and O, representing free and occupied positions, respectively, of the intermediate buffer, conveniently initially marked. It was decided initially being three occupied positions (number or tokens at O place) and four free positions (initial marking of place H).

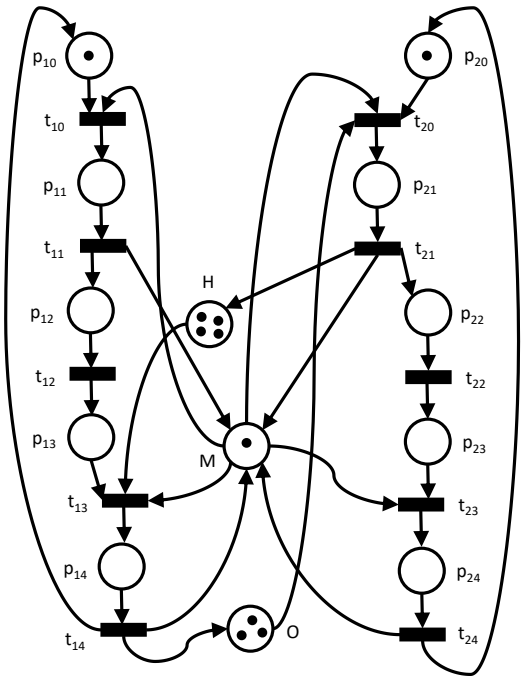

Figure 9. PN structure modeling manufacturing cell behavior.

Table 2. Interpretation to be added to the PN structure shown in Figure 9.

Place	Action	Description
H	-	Buffer free positions.
M	-	Robot, shared use.
O	-	Buffer occupied positions.
p_{10}	-	Wait until station 1 (M_1) can be loaded.
p_{11}	$CM1$	M_1 loading signal.
p_{12}	$PM1$	M_1 processing signal.
p_{13}	-	Wait until M_1 can be unloaded.
p_{14}	$DM1$	M_1 unloading signal.
p_{20}	-	Wait until station 2 (M_2) can be loaded.
p_{21}	$CM2$	M_2 loading signal.
p_{22}	$PM2$	M_2 processing signal.
p_{23}	-	Wait until M_2 can be unloaded.
p_{24}	$DM2$	M_2 unloading signal.
Transition	**Condition**	**Description**
t_{10}	PE	Part at input point.
t_{11}	$PCM1$	Part loaded in M_1.
t_{12}	-	Direct.
t_{13}	-	There are resources for unloading M_1.
t_{14}	EB	Part in buffer input.
t_{20}	SB	Part in buffer output.
t_{21}	$PCM2$	Part loaded in M_2.
t_{22}	-	Direct.
t_{23}	-	There are resources for unloading M_2.
t_{24}	PS	Part at output point.

PN interpretation to be added to the net structure is contained in Table 2. Actions (command signals through PLC digital outputs) were associated to places and conditions (sensor status queries) to transitions. Note that processing tasks were simulated by means of timers, so conditions $FPM1$ and $FPM2$ correspond to their endings. Places p_{13} and p_{23} do not have associated actions, as they correspond to waiting places for the necessary resources to face the respective machines unloading.

After PN structural analysis (Section 2.2.2) and matrix generation (Figure 10), structural data transfer was carried out (Section 2.2.3), previously converted to the syntax of the target development environment using a proprietary application. Both the characteristic arrays declaration and their contents were transferred, i.e., there was no need to declare them in the environment. Other necessary variables were stored in data memory.

The only non-fixed part of the algorithm was coded (Figure 11), the one corresponding to PN interpretation (Section 2.2.4). For each boolean data that indicated whether or not the conditions associated with each transition had been met, the result of the corresponding combination of logical queries was assigned. Outputs status was managed according to current marking. Note that this coding can be performed in any language supported by the development environment used.

VC of the automated system (Section 2.2.5) was supported by the DT already described. The operation validation was carried out by observing, in the DT, PLC controlled signals, and PN marking evolution in the development environment itself. A test protocol was followed starting from initial conditions, i.e., temporary buffer provided with three parts:

- Temporary buffer output sensor forced failure, so that no parts were supplied to M_2. Periodic entry of parts into the process. Buffer filling, with a part waiting to be unloaded from M_1, and a part at process input.
- Sensor repair. Periodic entry of parts into the process. Full emptying of temporary buffer.
- Continuous verification that at most only one command is given to the transport system.

	T10	T11	T12	T13	T14	T20	T21	T22	T23	T24
H	0	0	0	-1	0	0	1	0	0	0
M	-1	1	0	-1	1	-1	1	0	-1	1
O	0	0	0	0	1	-1	0	0	0	0
P10	-1	0	0	0	1	0	0	0	0	0
P11	1	-1	0	0	0	0	0	0	0	0
P12	0	1	-1	0	0	0	0	0	0	0
P13	0	0	1	-1	0	0	0	0	0	0
P14	0	0	0	1	-1	0	0	0	0	0
P20	0	0	0	0	0	-1	0	0	0	1
P21	0	0	0	0	0	1	-1	0	0	0
P22	0	0	0	0	0	0	1	-1	0	0
P23	0	0	0	0	0	0	0	1	-1	0
P24	0	0	0	0	0	0	0	0	1	-1

H	M	O	P10	P11	P12	P13	P14	P20	P21	P22	P23	P24
4	1	3	1	0	0	0	0	1	0	0	0	0

Figure 10. Matrix representation detail: PIPE2-generated C and m_0^t.

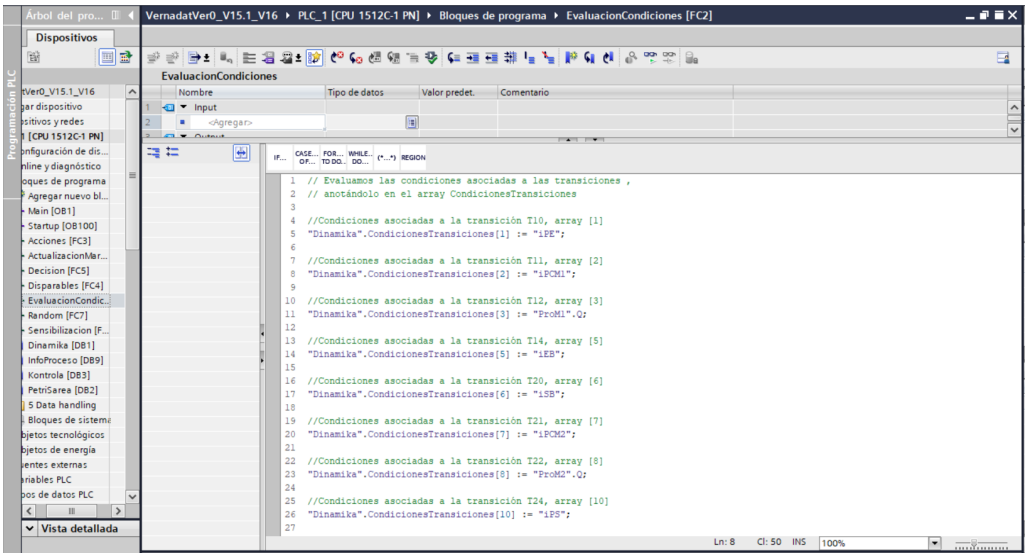

Figure 11. PN interpretation coding detail: editing of conditions associated with each transition.

3.3.2. Implementation of Other Approaches

In addition to the proposal of this paper, and referred to as PN in Section 4, the following interpreted approaches were implemented:

- ST: algorithm that processes both net structure and interpretation, by means of IF THEN structures in ST.
- LD: algorithm that processes both net structure and interpretation, by means of set and reset of variables, in LDs.
- SFC: graphic modeling and description method, suitable for GRAFCET and available in TIA Portal.

For each of the developments and control devices, scan cycle time measurements were performed, based on the connection of the physical PLC with the process DT, according to a HIL setup.

4. Results

The purpose of carrying out several implementations on controllers from different vendors was, on the one hand, to compare the scan cycle time required by each approach, and on the other hand, testing algorithm portability between development platforms. This section presents the results obtained.

4.1. Results Related to Scan Cycle Time

The scan cycle time for each of the implementations has been measured for each of the PLCs used, in order to determine whether or not the algorithm presented in this work consumes more resources than other approaches. Table 3 summarizes the results obtained. More than 100,000 scan cycles have been executed in all tests. Note that Omron´s Sysmac Studio provides the average value of the scan cycle period. Siemens' TIA Portal, on the other hand, gives the current value. In the latter case, 10 samples were observed with a 10 second interval between them. The value shown is the resulting average. Beyond the different levels of processing power offered by the CPUs used, and if relative terms are considered, implementations in ST are the fastest and in SFC the slowest. The scan cycle period required by the proposed approach is fairly similar to that of other comparable implementations.

Table 3. Measured scan cycle periods.

Implementation	CPU: Siemens 1521C-1PN	CPU: Omron NX102-1020
PN	1.029 ms	0.285 ms
ST	1.035 ms	0.276 ms
LD	1.130 ms	0.283 ms
SFC	1.361 ms	-

4.2. Results Related to Portability

After a first implementation in a Siemens PLC (see Section 3.3.1), the algorithm has been transferred into an Omron device. The algorithm, developed in ST, is easily portable to other platforms, although specific variable adaptations may be necessary.

5. Conclusions and Future Work

A novel application-independent and platform-independent methodology for implementing PNs in PLCs has been shown, in which most of the code is validated according to PN model structural analysis prior to implementation. Only net interpretation is manually coded and error-prone in the proposed approach, making it possible to generate PN-based code for PLC in a simple way. In comparison with previous work in the area of interest of this paper, analyzed in Section 1, the implementation (Section 3) of the proposed algorithm (Section 2) reveals the following:

- A PN is designed and formally analyzed, and the matrices representing its structure are obtained through a free license software tool, and these matrices are transferred to data memory of devices from different vendors. Thus, net structure is validated automatically before net implementation.
- Fixed program blocks, independent of both PLC and PN, manage net marking evolution, using the above-mentioned data structure. This is why the approach can be considered semi-compiled. ST programming makes this part standard.
- Consequently, the workload in terms of coding is reduced to associating conditions to transitions and actions to places, i.e., PN interpretation. It can be added in any language supported by the corresponding environment.

These points are based on the results achieved and already shown in Section 4:

- The scan cycle time required by the algorithm does not differ from that of similar, fully interpreted approaches, i.e., with both net marking and net interpretation management integrated into the code itself.
- The portability to the desired PLC is simple and not time-consuming. As far as the code is concerned, fixed program block contents are directly transferred. Regarding variables in data memory, it is necessary to declare and initialize those required for the algorithm to work. Arrays representing net structure are generated in a PN design and simulation software, and adapted by a proprietary parser software to PLC particular variables definition and initialization.

It is considered that the contribution of the work presented in this paper may have these implications:

- PN-based automated industrial systems are implemented in a simple way, with a reduced programming workload and, consequently, fewer potential errors in the control device code. The inclusion or modification of a PN is simple, requiring editing, validation, and transfer of new data into the PLC.
- The way to manage several nets (associated with respective subsystems) is to do it as a single one (as a single system), in which some arcs are considered to be zero weight. User-defined data types can be used if larger matrices are required than the limit set by the development tool.
- For technicians used to working with GRAFCET, the paradigm shift to a PN-based approach is not a handicap. Similar modeling skills are needed.

Note that the system under study has made it possible to test the algorithm by applying it in a case present in literature as suitable for the application of PNs. Shared resources and multiple marking make it more demanding for a GRAFCET-based approach. The existence of tokens ($\mathbf{N} = \{0, 1, 2, ...\}$) in a place in a PN versus states (bool) of steps in a GRAFCET, gives the former the capability to integrate the system's characterization into the model's structure, against further coding work in the latter. The use of a DT of the process has allowed the testing of the algorithm without availability of the necessary equipment, reproducing scenes difficult to assemble or reproduce.

The following may be some limitations of the approach presented:

- Net interpretation programming, the only part to be properly coded, is done according to the user's choice; no user-friendly or visual tool is available, yet.
- The algorithm involves an entire project in the development environment, is not parameterizable. Its portability between platforms can be simplified.
- PNs have as elements of their matrix representation natural numbers, often 0 or 1 for *ordinary* PNs. Extending this approach to high-level PNs, such as Predicate/Transition nets (PrT-nets) [34] or Coloured Petri nets (CPNs) [35], in which information is attached to each token, would increase the description capacity (making the validation process more difficult) and lead to significantly smaller nets.

Finally, and aligned with the above, the following future research directions have been identified:

- Further reduction of the coding workload by making it more systematic to add interpretation to the PN may be an interesting challenge. There is significant research on automatic code generation.
- The parameterization of a program block encapsulating the algorithm can make its handling an easier task. It can also be interesting to implement the model in MatLab, Java, and/or ADA, in order to compare both the cost of development and their performance.
- A high-level PN-based approach, applying object oriented techniques and/or featuring attributes to tokens, places (actions) and transitions (conditions), reducing the size of the model. User defined data types (UDTs) could help in PLC programming.
- Distributed PNs, in which the management of a master net is used to operate over the others, through open communication protocols such as OPC UA or message queue telemetry transport (MQTT).

Author Contributions: Conceptualization, J.C.M.; methodology, I.A., J.C.M. and L.E.; software, I.A., M.A. and J.C.M.; validation, I.A. and J.C.M.; investigation, J.C.M.; writing—original draft preparation, I.A.; writing—review and editing, I.A., M.A., J.C.M and L.E.; visualization, I.A.; supervision, L.E.; project administration, L.E. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Acknowledgments: We thank Jatsu Lizarribar, member of the Department of Basic Sciences at Mondragon Unibertsitatea, for his support in algebra topics.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

CPN	Coloured Petri net
DEDS	Discrete event dynamic system
DT	Digital twin
GRAFCET	Step transition function chart or GRAphe Fonctionnel de Commande Etape Transition
HIL	Hardware-in-the-loop
LD	Ladder diagram
MQTT	Message queue telemetry transport
OLE	Object Linking and Embedding
OPC UA	OLE for Process Control Unified Architecture
P/T	Place/Transition
PIPE2	Platform Independent Petri net Editor 2
PLC	Programmable logic controller
PN	Petri net
PrT-net	Predicate/Transition net
RG	Reachability graph
SFC	Sequential function chart
SIL	Software-in-the-loop
ST	Structured text
VC	Virtual commissioning

References

1. Ben-Naoum, L.; Boel, R.; Bongaerts, L.; De Schutter, B.; Peng, Y.; Valckenaers, P.; Vandewalle, J.; Wertz, V. Methodologies in discrete event dynamic systems. *J. A* **1995**, *36*, 3–14.
2. Sobh, T. *Discrete Event Dynamic Systems: An Overview*; Technical Reports; CIS: 1991. Available online: https://repository.upenn.edu/cis_reports/388/ (accessed on 24 January 2020).
3. Murata, T. Petri nets: Properties, analysis and applications. *Proc. IEEE* **1989**, *77*, 541–580. [CrossRef]
4. Moreira, M.V.; Basilio, J.C. Bridging the Gap Between Design and Implementation of Discrete-Event Controllers. *IEEE Trans. Autom. Sci. Eng.* **2014**, *11*, 48–65. [CrossRef]
5. Silva, M.; Teruel, E. Petri nets for the design and operation of manufacturing systems. *Eur. J. Control* **1997**, *3*, 182–199. [CrossRef]
6. Silva, M.; Teruel, E. A systems theory perspective of discrete event dynamic systems: The Petri net paradigm. In Proceedings of the CESA'96 IMACS Multiconference: Computational Engineering in Systems Applications, Lille, France, 9–12 July 1996; pp. 1–12.
7. DiCesare, F.; Harhalakis, G.; Proth, J.M.; Silva, M.; Vernadat, F. *Practice of Petri Nets in Manufacturing*; Springer: London, UK, 1993.
8. Reisig, W. *Petri Nets: An Introduction*; Springer: Berlin/Heidelberg, Germnay, 1985.
9. Peterson, J.L. *Petri Net Theory and the Modeling of Systems*; Prentice Hall PTR: Hoboken, NJ, USA, 1981.
10. David, R. Petri Nets and Grafcet for Specification of Logic Controllers. *IFAC Proc. Vol.* **1993**, *26*, 683–688. [CrossRef]
11. Michel, G. *Programmable Logic Controllers: Architecture and Applications*; John Wiley & Sons, Inc.: Hoboken, NJ, USA, 1990.
12. David, R.; Alla, H. *Petri Nets and Grafcet: Tools for Modelling Discrete Event Systems*; Prentice-Hall, Inc.: Hoboken, NJ, USA, 1992.
13. Cansever, G.; Kucukdemiral, I.B. A new approach to supervisor design with sequential control Petri-net using minimization technique for discrete event system. *Int. J. Adv. Manuf. Technol.* **2005**, *29*, 1267–1277. [CrossRef]
14. Stanton, M.; Arnold, W.; Buck, A. Modelling and Control of Manufacturing Systems Using Petri Nets. *IFAC Proc. Vol.* **1996**, *29*, 4754–4759. [CrossRef]
15. David, R.; Alla, H.L. *Du Grafcet Aux réseaux de Petri*; Hermes: London, UK, 1992.
16. Tzafestas, S.G.; Pantelelis, M.G.; Kostis, D.L. Design and Implementation of a Logic Controller using Petri Nets and Ladder Logic Diagrams. *Syst. Anal. Model. Simul.* **2002**, *42*, 135–167. [CrossRef]
17. Peng, S.S.; Zhou, M.C. Ladder diagram and Petri-net-based discrete-event control design methods. *IEEE Trans. Syst. Man Cybern. Part C* **2004**, *34*, 523–531. [CrossRef]
18. Zaytoon, J.; Riera, B. Synthesis and implementation of logic controllers—A review. *Annu. Rev. Control* **2017**, *43*. [CrossRef]
19. Uzam, M.; Jones, A. Discrete event control system design using automation Petri nets and their ladder diagram implementation. *Int. J. Adv. Manuf. Technol.* **1998**, *14*, 716–728. [CrossRef]
20. Peng, S.; Zhou, M. Sensor-based stage Petri net modelling of PLC logic programs for discrete-event control design. *Int. J. Prod. Res.* **2003**, *41*, 629–644. [CrossRef]
21. Borges, M.U.; Lima, E.J. Conversion methodologies from Signal Interpreted Petri Nets to Ladder Diagram and C language in Arduino. *Int. J. Mech. Eng. Educ.* **2018**, *46*, 302–314. [CrossRef]
22. Lee, G.B.; Zandong, H.; Lee, J.S. Automatic generation of ladder diagram with control Petri Net. *J. Intell. Manuf.* **2004**, *15*, 245–252. [CrossRef]
23. Frey, G. Automatic implementation of Petri net based control algorithms on PLC. In Proceedings of the 2000 American Control Conference, ACC (IEEE Cat. No. 00CH36334), Chicago, IL, USA, 28–30 June 2000; Volume 4, pp. 2819–2823.

24. Heiner, M.; Menzel, T. A Petri net semantics for the PLC language Instruction List. In Proceedings of the IEE Workshop on Discrete Event Systems (WODES'98), Sardinia, Italy, 26–28 August 1998; pp. 161–165.
25. Korotkin, S.; Zaidner, G.; Cohen, B.; Ellenbogen, A.; Arad, M.; Cohen, Y. A Petri Net formal design methodology for discrete-event control of industrial automated systems. In Proceedings of the 2010 IEEE 26th Convention of Electrical and Electronics Engineers in Israel, Eilat, Israel, 17–20 November 2010; pp. 431–435.
26. Markiewicz, M.; Gniewek, L. A Program Model of Fuzzy Interpreted Petri Net to Control Discrete Event Systems. *Appl. Sci.* **2017**, *7*. [CrossRef]
27. Rezig, S.; Ghorbel, C.; Achour, Z.; Rezg, N. PLC-based implementation of supervisory control for flexible manufacturing systems using theory of regions. *Int. J. Autom. Control* **2019**, *13*, 619. [CrossRef]
28. Aspar, Z.; Shaikh-Husin, N.; Khalil-Hani, M. Algorithm to Convert Signal Interpreted Petri Net Models to Programmable Logic Controller Ladder Logic Diagram Models. *Indones. J. Electr. Eng. Comput. Sci.* **2018**, *10*, 905. [CrossRef]
29. Petter, J.; Sawicki, S. A Comparison of Petri Net Simulation Tools. In Proceedings of the XXV Seminário de Iniciação Científica, Ijuí, Brazil, 25–29 September 2017; pp. 26 – 27.
30. Dingle, N.J.; Knottenbelt, W.J.; Suto, T. PIPE2: A Tool for the Performance Evaluation of Generalised Stochastic Petri Nets. *SIGMETRICS Perform. Eval. Rev.* **2009**, *36*, 34–39. [CrossRef]
31. Platform Independent Petri Net Editor 2. Available online: http://pipe2.sourceforge.net/ (accessed on 24 January 2020).
32. Johnstone, M.; Creighton, D.; Nahavandi, S. Enabling industrial scale simulation/emulation models. In Proceedings of the 2007 Winter Simulation Conference, Washington, DC, USA, 9–12 December 2007. [CrossRef]
33. Simumatik3D—Home. Available online: https://simumatik3d.github.io/ (accessed on 10 August 2020).
34. Genrich, H.J.; Lautenbach, K. System modelling with high-level Petri nets. *Theor. Comput. Sci.* **1981**, *13*, 109–135. [CrossRef]
35. Jensen, K. *Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use*; Springer Science & Business Media: Berlin/Heidelberg, Germany, 2013; Volume 1.