applied sciences

MDPI

Article

Event-Based Modeling of Input Signal Behaviors for Discrete-Event Controllers

Luis Gomes [1,2,3,*], Diogo Natário [1], Anikó Costa [1,2,3], João-Paulo Barros [2,3,4] and Rogério Campos-Rebelo [2,3,4]

[1] School of Science and Technology, NOVA University Lisbon, 2829-516 Caparica, Portugal; d.natario@campus.fct.unl.pt (D.N.); akc@fct.unl.pt (A.C.)
[2] Center of Technology and Systems, UNINOVA, 2829-516 Caparica, Portugal; joao.barros@ipbeja.pt (J.-P.B.); rcr@uninova.pt (R.C.-R.)
[3] Intelligent Systems Associate Laboratory (LASI), 4800-058 Guimarães, Portugal
[4] Instituto Politécnico de Beja, 7800-295 Beja, Portugal
* Correspondence: lugo@fct.unl.pt

Abstract: Controllers for discrete-event systems are commonly designed using state-based formalisms, like state diagrams and Petri nets. These formalisms are strongly supported by the concept of events, which, from an automation system perspective, can be associated with a simple change in the value of a signal or more complex behavioral evolutions of the signals. In this paper, the characterization of several types of events is proposed, associated with different types of signals, such as Boolean and multivalued signals. The major goal of this characterization is to improve the compactness of the model, benefiting the editing and visual interpretation of the graphical model but keeping precise execution semantics, which in turn allows for the use of computational tools covering the different stages of system development. The behavioral model of the controller is produced using a non-autonomous class of Petri nets, the IOPT nets, and the associated IOPT-Tools, which supports the specification, simulation, property verification, and automatic code generation ready to be deployed into implementation platforms. All the types of proposed events have a behavioral sub-model executed concurrently with the main model of the controller. An application example is provided to illustrate some of the advantages of the adoption of the proposed approach, encapsulating the behavioral dependencies on the evolution of input signals into events.

Keywords: Petri nets; discrete-event systems; automation systems

Citation: Gomes, L.; Natário, D.; Costa, A.; Barros, J.-P.; Campos-Rebelo, R. Event-Based Modeling of Input Signal Behaviors for Discrete-Event Controllers. *Appl. Sci.* **2024**, *14*, 5289. https://doi.org/10.3390/app14125289

Academic Editor: Eui-Nam Huh

Received: 5 April 2024
Revised: 5 June 2024
Accepted: 14 June 2024
Published: 19 June 2024

1. Introduction

When one wants to model the behavior of controllers for embedded and cyber–physical systems, including those used in automation areas, it is widely accepted that selecting a modeling formalism having both precise execution semantics, a textual representation, and a graphical representation will be an advantage, as far as it will allow for a proper support from computational tools, as well as an effective interaction between the humans involved.

Among those formalisms widely in use, state-based modeling formalisms, such as state diagrams, statecharts, and Petri nets, are common examples. However, the expressiveness of the model and its compactness, specifically in terms of its graphical representation, are major characteristics that need to be considered when selecting the modeling formalism.

Some modeling formalisms (tentatively, most of them) benefit from the usage of hierarchical structuring mechanisms, while others achieve compactness through encapsulating behavioral dependencies into the model's annotations. The latter has a strong drawback: It can be perceived as a risky approach, because to achieve compactness, one may lose the intuitive perception of the behavior that one wants to model from the graphical representation.

We argue that the desired behavior needs to be clearly perceived from the model, using the graph structure to provide a clear description of the intended behavior.

In this paper, we propose to improve the model compactness through the usage of events obtained from the analysis of the evolution of the controller's input signals. These events can, in turn, be generated by the execution of a priori available (hidden) sub-models or pre-processing at the execution code level.

Petri nets will be used as the modeling formalism to express the controller's behavior and to model the events generated from the analysis of the signal evolution. In particular, the class of IOPT nets (Input-Output Place-Transition nets) [1–3] will be considered. In addition, the associated computational tools framework IOPT-Tools [4–6], freely available at http://gres.uninova.pt/IOPT-Tools/ (accessed on 3 June 2024), will be used to support the model editing (where the proposals have been integrated).

This approach (even presented within a Petri net modeling-based approach) can be adopted by several other state-based behavioral formalisms, including state diagrams, and derived formalism, such as hierarchical and concurrent state diagrams and statecharts and other classes of Petri nets as well (as far as all the generated event sub-models, even expressed using a Petri net notation, can also be unfolded into state diagrams). It is important to emphasize that the automatic code generation possibility will be preserved, supported by a model-driven approach, while allowing for the delivery of executable code from the behavioral model directly deployable into implementation platforms (ultimately without writing a line of code).

The structure of this paper is as follows. Section 2 gives the basic information on some approaches to include events within system modeling, as well as on Petri nets and IOPT nets. Section 3 provides the rationale and summarizes the steps associated with the proposed approach. Section 4 addresses the generation of events associated with Boolean signals, while Section 5 addresses similar topics considering multivalued signals. Finally, Section 6 provides a brief discussion on the results using a small example, and Section 7 concludes and points out some future works.

2. Related Work

It is relevant to note that the use of events is transversal across several application areas, allowing for the detection of simple changes or even complex behaviors. This is the case in [7], where a graphical formalism for signal interpretation modeling is proposed, allowing for the identification of associated events and conditions, which can be used in the behavior model description of the system.

Complex Event Processing (CEP) [8] has been used in a wide range of areas, including network management, databases, and discrete-event simulation. CEP handles the composition of events from an event cloud, leading to the implementation of Event-Driven Architectures and allowing for the extraction of information from distributed systems based on events and messages.

In the context of controllers for embedded and cyber–physical systems, events, considered changes in the value of signals, are widely used by different modeling formalisms, technologies, and computational tools. Their use leads to an intrinsic reduction in the dimension of the model when graphical representations are considered. This is notably the case for some state-based computational models, such as reactive systems [9], statecharts [10], and Petri nets [11].

In [12], the approach followed in this paper was first proposed, relying on the definition of sub-models to encapsulate the detection of specific signal evolution and further execution concurrently with the main system's model. In [12], two types of signals were considered, namely, Boolean signals and multivalued signals. For Boolean signals, a set of associated events was proposed, namely, *Up*, *Down*, *UpOrDown*, *UpDown*, and *DownUp*. For multivalued signals, the set of events defined for Boolean signals was defined and extended with *UpDownHyst* and *DownUpHyst*, introducing a hysteresis dependency on the analysis of multivalue signals.

Building upon the work of [12], this paper refines their proposals on hysteresis dependencies and introduces a novel event type. This new event type incorporates an additional constraint when analyzing signal evolution, requiring a specific condition to be met for event generation.

IOPT Nets—Input-Output Place-Transition Nets—A Brief Overview

Carl Adam Petri introduced basic Petri net concepts in his 1962 Ph.D. thesis [13]. Since then, various classes of Petri nets have been proposed, making them a diverse family of languages [14]. Despite their diversity, all Petri nets can be represented through a graph with two types of nodes, named places and transitions, connected by directed arcs. Nodes of one type can only connect to nodes of the other type. Places are visually represented as circles or ellipses, while bars, squares, or rectangles denote transitions. These two types of nodes embody a dual perspective when modeling systems: places are associated with states or resources, while transitions are associated with changes in the state, actions, or resource production and consumption.

A positive integer value, which can include zero, is associated with each place, is referred to as the "place marking", and is graphically represented by tokens in the form of small dots inside the place or by a number. The current system state is composed of a set of place markings. The system state undergoes changes when one or more transitions are fired. When a transition fires, it removes tokens from its input places and generates tokens into its output places according to the weights assigned to the arcs connecting the transition and the place.

A transition in a Petri net model can only fire if it is *enabled*. For a transition to be enabled, each input place contains at least the number of tokens specified by the arc weight connecting that place to the transition. However, when the goal is to model automation systems and discrete-event controllers' behavior, it is of paramount importance to consider additional external conditions that must be met for a transition to fire. In such cases, we refer to the Petri net as a non-autonomous model, and for the transition to be fireable, it must be enabled (from the point of view of the marking) and *ready* (from the point of view of the external constraints). Several classes of non-autonomous Petri nets have been introduced over the years, namely, [15–20], including some specifically addressing manufacturing systems as the application domain [21].

Some of these Petri net classes were developed by adopting or expanding upon the concepts of synchronized and interpreted Petri nets [20] or targeting specific areas, such as Control Interpreted Petri nets (independently proposed by [22,23]).

This paper considers Input-Output Place-Transition nets (IOPT nets), which belong to the category of non-autonomous Petri nets designed for modeling discrete-event controllers [1–3]. Additionally, a suite of web-based tools, collectively known as IOPT-Tools, is freely available [4–6]. This set of tools directly supports editing, simulation, property verification, and code generation. Automatic code generators can produce both C and VHDL code directly from the models, which aligns with the goal of having execution code without writing a line of code. The C code can be directly executed on platforms like Arduino boards and several single-board computers, including Raspberry Pi and other Linux platforms. Also, VHDL code can be seamlessly deployed into FPGA-based boards.

IOPT nets are a non-autonomous extension of place-transition nets [24], being equipped with non-autonomous characteristics that build upon the principles of synchronized and interpreted Petri nets, as outlined in [20]. Specifically, IOPT nets consider input signals, which can constrain transition firing when involved in the transition guard. The analysis of signal evolution, which identifies changes in its value, can generate events, which in turn can also constrain transition firing.

In this sense, in an IOPT net, the firing of a transition needs to be *enabled* (enough tokens at input places) and *ready* from the point of view of its guard and its event (if any). We consider that a transition is *fireable* when it is enabled and ready.

In the simple model presented in Figure 1, for transition $t2$ to fire, places $p1$ and $p2$ must be marked, the guard evaluated to *true* (input signal b equals to 1), and the event ev_a (associated with input signal a) occurs. Figure 1 shows the simple IOPT net, as well as the relevant interfaces (to provide characteristics for signal a, event ev_a, and transition $t2$) provided by IOPT-Tools to support its implementation.

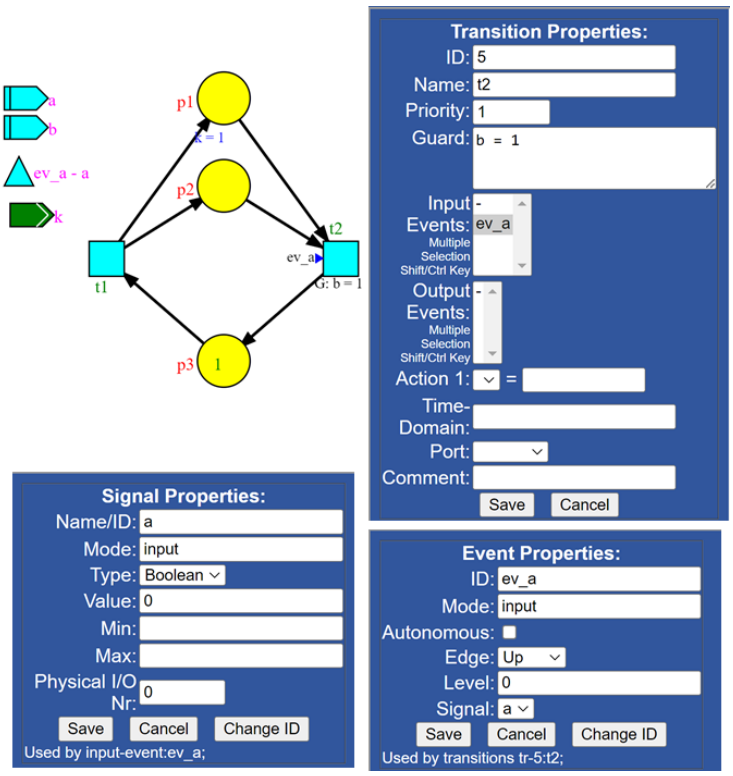

Figure 1. Small IOPT model and non-autonomous annotations.

IOPT Nets: Interface, Syntax, and Semantics

In this sub-section, a few basic definitions associated with the class of IOPT nets are presented. They are adapted from [3] and start with the characterization of the interface with the environment in terms of the inputs and outputs of the controller. After that, the syntax of the IOPT nets is presented, as well as the associated execution semantics, where the dependencies of the inputs and outputs of the system are considered.

Definition 1 (Model interface)**.** *(from [3].) The interface between the controlled system and the IOPT net is a tuple* $MI = (IS_B, IS_R, IE_{NA}, IE_A, inputSignal, OS_B, OS_R, OE_{NA}, OE_A, outputSignal)$ *satisfying the following:*

1. IS_B *is a finite set of Boolean input signals;*
2. IS_R *is a finite set of range (non-negative integers) input signals;*
3. IE_{NA} *is a finite set of non-autonomous input events;*
4. IE_A *is a finite set of autonomous input events;*
5. $inputSignal$ *is a function applying non-autonomous input events to a signal and an upper or lower edge:* $inputSignal : IE_{NA} \rightarrow (IS_B \cup IS_R) \times \{upper, lower\}$;
6. OS_B *is a finite set of Boolean output signals;*
7. OS_R *is a finite set of range (non-negative integers) output signals;*
8. OE_{NA} *is a finite set of non-autonomous output events;*
9. OE_A *is a finite set of autonomous output events;*
10. $outputSignal$ *is a function applying non-autonomous output events to a signal and an upper or lower edge:* $outputSignal : OE_{NA} \rightarrow (OS_B \cup OS_R) \times \{upper, lower\}$;
11. $IS_B \cap IS_R \cap IE_{NA} \cap IE_A \cap OS_B \cap OS_R \cap OE_{NA} \cap OE_A = \emptyset$.

Definition 2 (IOPT net). *(adapted from [3].) Given a model interface* $MI = (IS_B, IS_R, IE_{NA}, IE_A, inputSignal, OS_B, OS_R, OE_{NA}, OE_A, outputSignal)$*, an IOPT net is a tuple* $N = (P, T, A, TA, M, weight, weightTest, priority, guard, ie, oe, ta, osc)$ *satisfying the following requirements:*

1. P *is a finite set of places;*
2. T *is a finite set of transitions, such that* $P \cap C \cap T = \emptyset$*;*
3. A *is a set of arcs, such that* $A \subseteq ((P \times T) \cup (T \times P))$*;*
4. TA *is a set of test arcs, such that* $TA \subseteq (P \times T)$*;*
5. M *is the marking function:* $M : P \to \mathbb{N}_0$*;*
6. $weight$ *is the arc weight function:* $weight : A \to \mathbb{N}_0$*;*
7. $weightTest$ *is the test arc weight function:* $weightTest : TA \to \mathbb{N}_0$*;*
8. $priority$ *is a partial function applying transitions to non-negative integers,* $priority : T \rightharpoonup \mathbb{N}_0$*, and we write* $priority(t) \uparrow$ *when transition* t *has no defined priority;*
9. $guard$ *is a* guard *partial function applying transitions to Boolean expressions, where all the variables are markings or signals:* $guard : t \rightharpoonup BE$*, where* $\forall eb \in guard(t)$*,* $Var(eb) \subseteq (M \cup IS_B \cup IS_R)$*;*
10. ie *is an input event partial function applying transitions to input events:* $ie : T \rightharpoonup \mathcal{P}(IE_{NA} \cup IE_A)$*;*
11. oe *is an output event function applying transitions to output events:* $oe : T \to \mathcal{P}(OE_{NA} \cup OE_A)$*;*
12. ta *is a transition action partial function applying transitions to actions:* $ta : T \rightharpoonup (OS_B \times BES) \cup (OS_R \times IES)$ *and* $\forall bae \in BES, Var(bae) \subseteq (ML \cup OS_B \cup OS_R)$ *and* $\forall iae \in IES, Var(iae) \subseteq (ML \cup OS_B \cup OS_R)$*.*
13. osc *is an output signal condition function from places into sets of rules:* $osc : P \to \mathcal{P}(RULES)$*, where* $RULES \subseteq ((OS_B \cup OS_R) \times BES \times \mathbb{N}_0)$*, and* $\forall e \in BES, Var(e) \subseteq (ML \cup IS_R \cup IS_B \cup OS_R \cup OS_B)$*.*

The execution of the IOPT net, modeling a discrete-event controller, adheres to step-based execution semantics. Each step encompasses several stages: (1) reading the input signals; (2) computing the input events based on the current and previously read values from the input signals; (3) considering a maximal step semantics execution firing all the enabled and ready transitions in the net (taking into consideration the existing conflicts); (4) updating the output values; and (5) storing the last input values for the subsequent steps.

The concepts of fireable transition, enabled transition, and ready transition were informally introduced in the previous section; ref. [3] is recommended to obtain additional details on the associated formal definitions.

In this paper, whenever necessary, the order of a step is explicitly referred to as "n" while "$n-1$" refers to the preceding execution step, and so forth.

3. Proposed Approach

As identified in Section 1, the main goal of this work is to improve the model compactness by encapsulating the input signal evolution dependencies into events. The proposed approach can be applied to different state-based formalisms. In this paper, IOPT nets will be considered the supporting formalism for their usage.

In this context, the most simple event associated with a Boolean signal is generated by a change in the value of the signal in two consecutive execution steps, previously proposed in [1,12,25]. This leads to the common *Up* and *Down* events associated with the two possible changes in signal value. However, other types of simple events are of interest to mimic the more complex evolution of signals. Two of those events are the *UpDown* event and the *DownUp* event, initially proposed in [12]. In the *UpDown* event, a full evolution of the signal from 0 to 1 and back to 0 is necessary to be observed, while the *DownUp* event detects dual evolution. In Figure 2, those events are presented (showing the time evolution of signal *a* and the instant where the associated event is generated), along with the *UpOrDown* event, where any of the evolutions associated with the events *Up* or *Down* are detected. For all these events, a simple state diagram using a Petri net notation is presented, allowing for the detection of that evolution of the signal *a* (taking advantage

of the transition guards constrained by the signal value). To better clarify the diagrams' semantics, we now detail the working of the first diagram in Figure 2. As stated in the previous section, each transition can only fire when it is *enabled* and *ready*. A transition is *enabled* when there is one token in all of its input places (all are *marked*). In the example, it suffices to have one token in place *pa* as it is the only input place for transition *t1*; in IOPT nets, the tokens in each place are specified by a positive number in each place (*1* in Figure 2a). A transition is *ready* when the respective guard and input event are true. Absent guards or absent input events are semantically equivalent to guards and input events that evaluate to true. In the model in Figure 2a, transition *t1* only fires when signal *a* has value 0; then, transition *t1* becomes enabled (one token in place *p1*) and can only fire when signal *a* has value *1* (it becomes *ready*. The model in Figure 2a represents the behavior of the input signal *a* as depicted in the time evolution diagram below the Petri net. Therefore, the signal *a* behavior *Up(a)* can be expressed by the above time evolution diagram or by the Petri net. The same happens for each of the other diagrams in Figure 2. This dual representation has a direct correspondence to what can be performed in the implementation: (1) either the Petri net model (IOPT model) is added to the primary model to specify the signal behavior or (2) the signal time evolution is computed so that the event is made available for the Petri net modeler to use in one or more transitions.

The proposed approach intends to shrink the model collapsing sequence of nodes associated with a specific signal evolution into an event depending on the same signal evolution. In this sense, instead of using the Petri net model presented in Figure 3a associated with the *Up* event (introduced in Figure 2a), it is proposed to use the simplified model presented in Figure 3b, composed of a simplified model where transition *ta* encapsulates the sub-model composed of *t1*, *p1*, and *t2*, which will be executed concurrently with the model responsible for the analysis of the signal *a* evolution and for the generation of the event *Up(a)*.

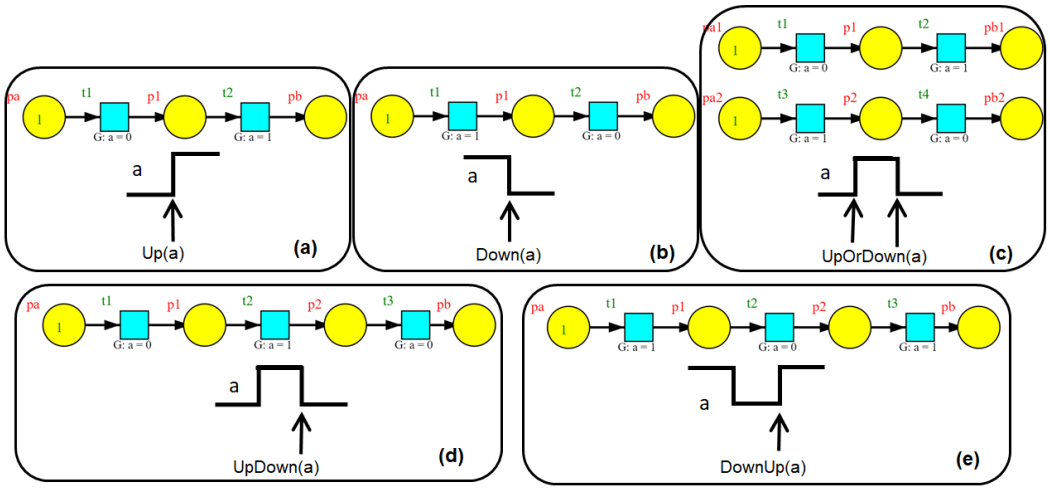

Figure 2. Different types of events associated with Boolean signals: (**a**) *Up* event, (**b**) *Down* event, (**c**) *UpOrDown* event, (**d**) *UpDown* event, and (**e**) *DownUp* event.

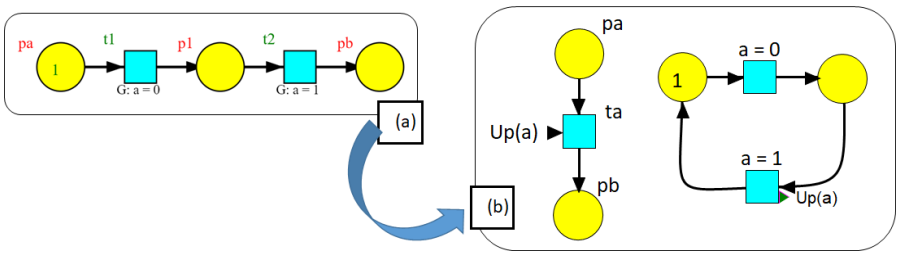

Figure 3. Restructuring the model introducing dependency on one event: (**a**) the initial model and (**b**) the model used for execution.

In this sense, the proposed strategy to improve the compactness and legibility of the model relies on encapsulating the specific evolution of the signals into the events, which will be associated with the transitions firing the sub-models characterizing the specific evolution of the signals. Those transitions will be constrained by the event's occurrence, which in turn characterizes the specific evolution of the signal and will be described through specific sub-models to be executed concurrently. The overall approach is summarized in Figure 4, where the "IOPT input event model" will be intrinsically considered available (without the need to be explicitly edited). Therefore, the model to be produced by the designer will be much smaller, as it will benefit from the encapsulation of the event dependencies.

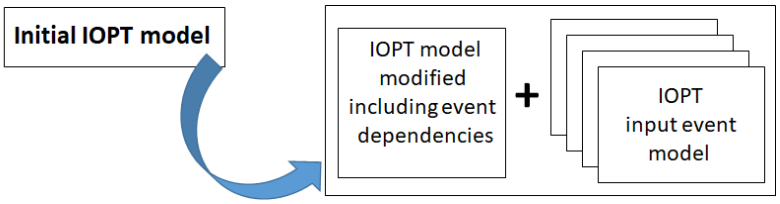

Figure 4. The overall approach relying on the introduction of events.

It is important to note that this approach will preserve the operational behavior of the initial model when executing the transformed model. This preservation aligns with the Petri net reduction techniques as presented by Murata [26], preserving the Petri net model's liveness, safeness, and boundedness properties.

In addition to the already referred events, a set of new types of events are proposed. This new type of event starts considering the already defined events but imposes a specific condition that constrains their analysis. The condition to be fulfilled is that the transition to which the event is associated needs to be enabled (from the point of view of the marking) and ready (from the point of view of the non-autonomous characteristics attached to the transition). Those new types of events will have "*Trigger*" added to the name of the five defined events, so their names will be *UpTrigger*, *DownTrigger*, *UpOrDownTrigger*, *UpDownTrigger*, and *DownUpTrigger*.

To illustrate the proposal more clearly, let us consider the UpTrigger event. Similar to the Up event, it analyzes the signal's evolution. However, this analysis occurs only after and while the associated transition is enabled and ready. This readiness is determined by the marking of the input places and the guard constrained by other signals.

Figure 5b illustrates the approach: transition $t1$ is receptive to the event *Up* associated with signal *a*, following the operational model of Figure 5c; similarly, transition $t3$ is receptive to the event *UpTrigger*, which is associated with signal *a* and also to this specific transition $t3$, following the operational model of Figure 5d. Regarding transition $t3$ firing, the analysis of the signal evolution is carried on if and while transition $t3$ is fireable. Transition $t3$ is fireable when it is enabled (places pa and pb in Figure 5b are marked) and ready (the guard composed of signal c equals to 1 and evaluates to true). In Figure 5d (and other figures to be presented in the next sections), the evaluation of this condition is included in the notation as $Fireable(t3)$ and $Not(Fireable(t3))$. In the presented case of Figure 5d, the condition $Fireable(t3)$ is evaluated by $(pa = 1 \wedge pb = 1 \wedge c = 1)$.

In the following two sections, events associated with Boolean signals and events associated with multivalued signals are presented.

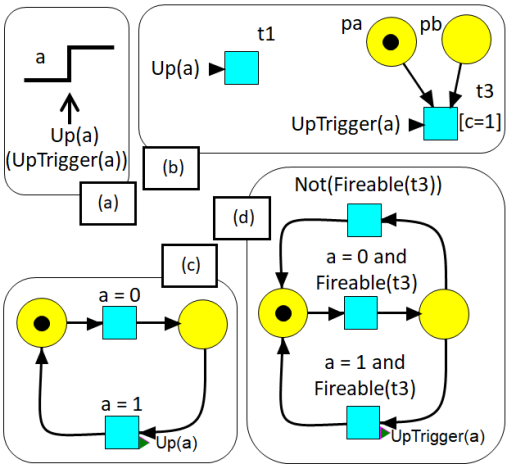

Figure 5. Modeling the *Up* and *UpTrigger* events: (**a**) the signal behavior to detect, (**b**) the main model with a transition $t1$ associated with an *Up* event and transition $t3$ with two input places, one guard and an associated *UpTrigger* event, (**c**) the sub-model for the *Up* event and (**d**) the sub-model for the *UpTrigger* event.

4. Events Depending on a Boolean Signal

Table 1 presents the definition of two sets of five events generated through the evolution analysis of Boolean signals, being the former set composed of Up, $Down$, $UpOrDown$, $UpDown$, and $DownUp$ (which updates the definitions already proposed in [12]) and the latter set constrained by the verification of conditions to fire the transition and composed of $UpTrigger$, $DownTrigger$, $UpOrDownTrigger$, $UpDownTrigger$, and $DownUpTrigger$. The timings associated with the generation of the events were previously presented in Figure 2.

Table 1. Definition of events associated with Boolean signal a.

Name	Expression
$Up(a)$	$a_{n-1} = 0 \wedge a_n = 1$
$Down(a)$	$a_{n-1} = 1 \wedge a_n = 0$
$UpOrDown(a)$	$(a_{n-1} = 0 \wedge a_n = 1) \vee (a_{n-1} = 1 \wedge a_n = 0)$
$UpDown(a)$	$a_m = 0 \wedge a_{n-1} = 1 \wedge a_n = 0$ with $m < (n-1)$
$DownUp(a)$	$a_m = 1 \wedge a_{n-1} = 0 \wedge a_n = 1$ with $m < (n-1)$
$UpTrigger(a)$	$(a_{n-1} = 0 \wedge a_n = 1)$ if and while receptive transition is enabled and ready
$DownTrigger(a)$	$(a_{n-1} = 1 \wedge a_n = 0)$ if and while receptive transition is enabled and ready
$UpOrDownTrigger(a)$	$(a_{n-1} = 0 \wedge a_n = 1) \vee (a_{n-1} = 1 \wedge a_n = 0)$ if and while receptive transition is enabled and ready
$UpDownTrigger(a)$	$(a_m = 0 \wedge a_{n-1} = 1 \wedge a_n = 0$ with $m < (n-1))$ if and while receptive transition is enabled and ready
$DownUpTrigger(a)$	$(a_m = 1 \wedge a_{n-1} = 0 \wedge a_n = 1$ with $m < (n-1))$ if and while receptive transition is enabled and ready

Figure 6 presents the models that generate the referred events. Each model is executed concurrently with the specification model where the respective event is used. In Figure 6b,d,f,h, it is assumed that those events are associated with one specific transition (a reference to $t3$ is used in this figure).

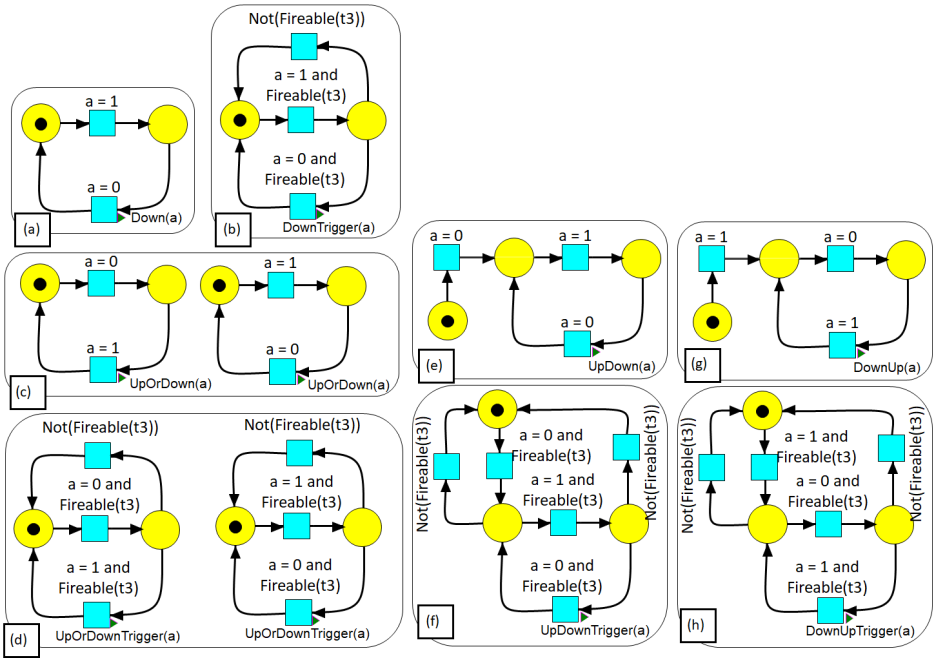

Figure 6. The operational models associated with the events listed in Table 1 to be executed concurrently with the model specified by the designer: (**a**) Down event, (**b**) DownTrigger event, (**c**) UpOrDown event, (**d**) UpOrDownTrigger event, (**e**) UpDown event, (**f**) UpDownTrigger event, (**g**) DownUp event, and (**h**) DownUpTrigger event.

5. Events Depending on Multivalued Signals

This section presents the set of events associated with multivalued signals, where a range of integer values can be read (digital inputs). Similarly to the analysis of Boolean signals, Table 2 presents the definition of four sets of five events generated through the evolution analysis of multivalued signals, providing updated definitions for the former set composed of Up, $Down$, $UpOrDown$, $UpDown$, and $DownUp$ events (which were initially proposed in [1,12,25]), where a threshold value k is used for comparison with a predefined level. A second set of events is proposed associated with similar events constrained by the verification of conditions to fire the transition with whom they are associated; this set is composed of $UpTrigger$, $DownTrigger$, $UpOrDownTrigger$, $UpDownTrigger$, and $DownUpTrigger$, also considering a value k for comparison. Figure 7 presents the timings associated with generating the referred events.

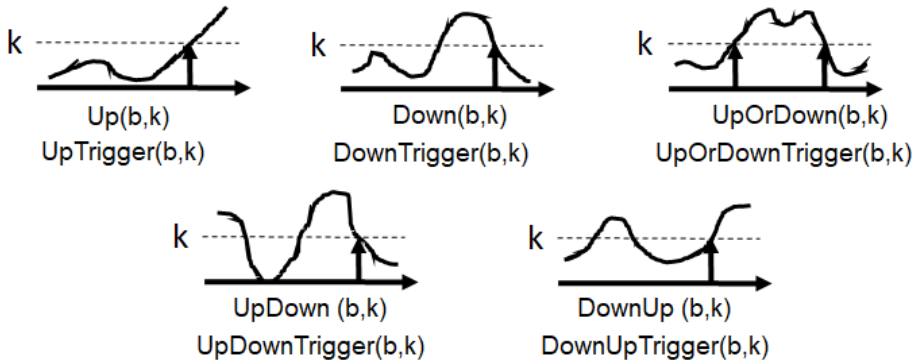

Figure 7. First set of event types associated with multivalued signals and associated timing generation.

Considering the specificity of multivalued input signals, which can be affected by noise during their acquisition, it is of paramount importance to intrinsically include an analysis of the signal evolution considering two levels of comparison, using a hysteresis strategy as proposed in [25], in order to improve the robustness to noise. With this in mind, two new sets

of events are proposed as duplicates of the two sets referred to in the previous paragraph, where two levels of comparison ($k1$ and $k2$) are used to consider a hysteresis dependency. Timings associated with the generation of the referred events are presented in Figure 8.

Table 2. The definitions of the events associated with multivalued signal b (after one activation of the event, the previous values of the signal b should be discarded).

Name	Expression
$Up(b,k)$	$b_{n-1} \leq k \wedge b_n > k$
$Down(b,k)$	$b_{n-1} > k \wedge b_n \leq k$
$UpOrDown(b,k)$	$(b_{n-1} \leq k \wedge b_n > k) \vee (b_{n-1} > k \wedge b_n \leq k)$
$UpDown(b,k)$	$b_m \leq k \wedge b_{n-1} > k \wedge b_n \leq k$ with $m < (n-1)$
$DownUp(b,k)$	$b_m > k \wedge b_{n-1} \leq k \wedge b_n > k$ with $m < (n-1)$
$UpTrigger(b,k)$	$b_{n-1} \leq k \wedge b_n > k$ if and while receptive transition is enabled and ready
$DownTrigger(b,k)$	$b_{n-1} > k \wedge b_n \leq k$ if and while receptive transition is enabled and ready
$UpOrDownTrigger(b,k)$	$(b_{n-1} \leq k \wedge b_n > k) \vee (b_{n-1} > k \wedge b_n \leq k)$ if and while receptive transition is enabled and ready
$UpDownTrigger(b,k)$	$b_m \leq k \wedge b_{n-1} > k \wedge b_n \leq k$ with $m < (n-1)$ if and while receptive transition is enabled and ready
$DownUpTrigger(b,k)$	$b_m > k \wedge b_{n-1} \leq k \wedge b_n > k$ with $m < (n-1)$ if and while receptive transition is enabled and ready
$UpHyst(b,k1,k2)$	$b_m \leq k_2 \wedge b_{n-1} \leq k_1 \wedge b_n > k_1$ with $m < n$
$DownHyst(b,k1,k2)$	$b_m > k_2 \wedge b_{n-1} > k_1 \wedge b_n \leq k_1$ with $m < n$
$UpOrDownHyst(b,k1,k2)$	$(b_m \leq k_2 \wedge b_{n-1} \leq k_1 \wedge b_n > k_1$ with $m < n) \vee$ $(b_m > k_2 \wedge b_{n-1} > k_1 \wedge b_n \leq k_1$ with $m < n)$
$UpDownHyst(b,k1,k2)$	$b_m \leq k_2 \wedge b_{n-1} > k_1 \wedge b_n \leq k_2$ with $m < (n-1)$
$DownUpHyst(b,k1,k2)$	$b_m > k_1 \wedge b_{n-1} \leq k_2 \wedge b_n > k_1$ with $m < (n-1)$
$UpHystTrigger(b,k1,k2)$	$b_m \leq k_2 \wedge b_{n-1} \leq k_1 \wedge b_n > k_1$ with $m < n$ if and while receptive transition is enabled and ready
$DownHystTrigger(b,k1,k2)$	$b_m > k_2 \wedge b_{n-1} > k_1 \wedge b_n \leq k_1$ with $m < n$ if and while receptive transition is enabled and ready
$UpOrDownHystTrigger(b,k1,k2)$	$(b_m \leq k_2 \wedge b_{n-1} \leq k_1 \wedge b_n > k_1$ with $m < n) \vee$ $(b_m > k_2 \wedge b_{n-1} > k_1 \wedge b_n \leq k_1$ with $m < n)$ if and while receptive transition is enabled and ready
$UpDownHystTrigger(b,k1,k2)$	$b_m \leq k_2 \wedge b_{n-1} > k_1 \wedge b_n \leq k_2$ with $m < (n-1)$ if and while receptive transition is enabled and ready
$DownUpHystTrigger(b,k1,k2)$	$b_m > k_1 \wedge b_{n-1} \leq k_2 \wedge b_n > k_1$ with $m < (n-1)$ if and while receptive transition is enabled and ready

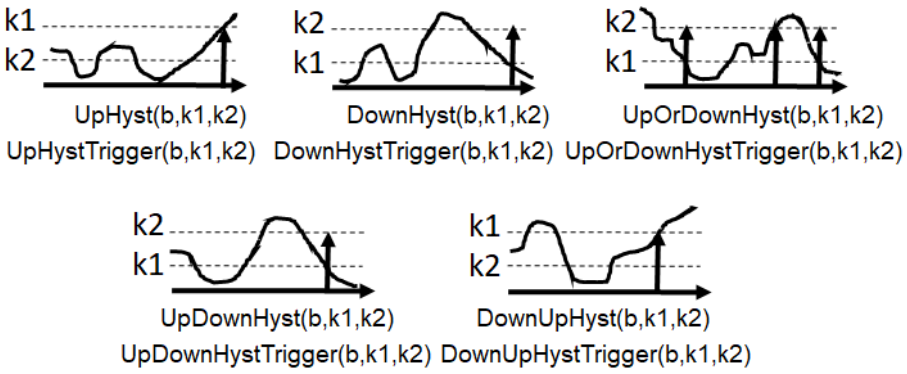

Figure 8. The second set of event types associated with the multivalued signals considering hysteresis on the signal analysis and associated timing generation.

Figure 9 illustrates the sets of the models associated with each of the events above, executed concurrently with the specification model where these events are employed. Similar to the previous presentation of Boolean signals, we also assume here that the "Trigger" events are associated with transition $t3$.

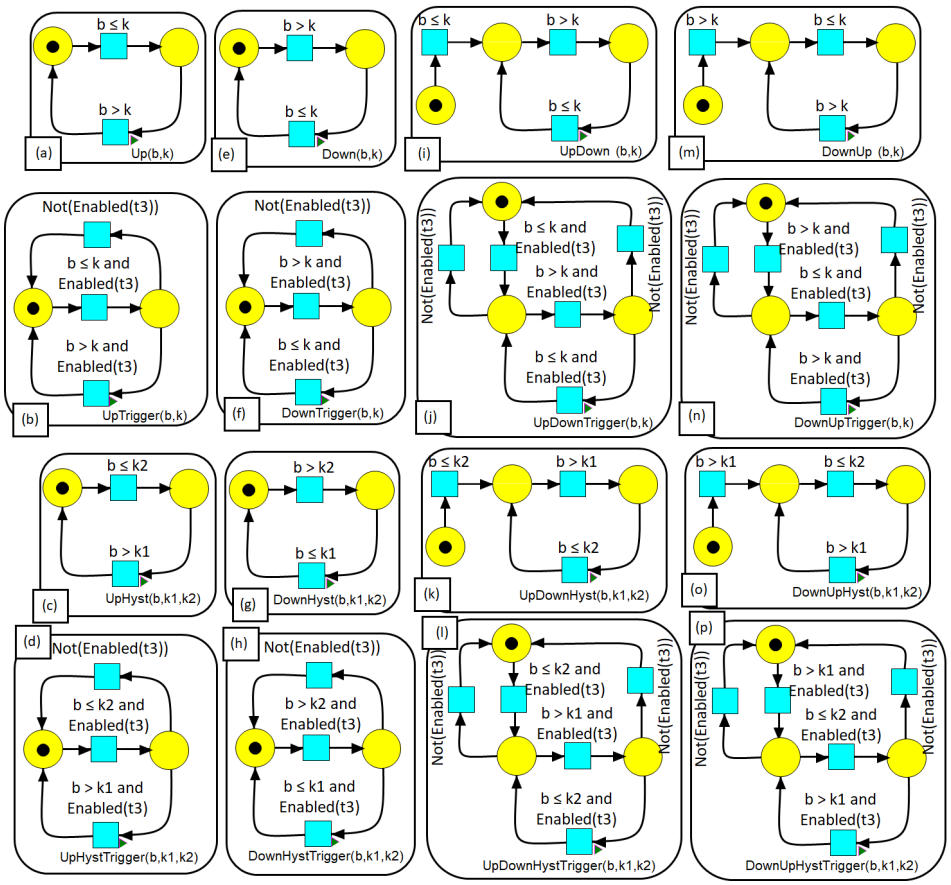

Figure 9. The operational models associated with the events listed in Table 2 to be executed concurrently with the model specified by the designer: (**a**) Up event, (**b**) UpTrigger event, (**c**) UpHyst event, (**d**) UpHystTrigger event, (**e**) Down event, (**f**) DownTrigger event, (**g**) DownHyst event, (**h**) DownHystTrigger event, (**i**) UpDown event, (**j**) UpDownTrigger event, (**k**) UpDownHyst event, (**l**) UpDownHystTrigger event, (**m**) DownUp event, (**n**) DownUpTrigger event, (**o**) DownUpHyst event, and (**p**) DownUpHystTrigger event.

6. Discussion

In order to illustrate the effectiveness of the proposed approach, a controller for a simple system responsible for packing three components coming from three different conveyors (A, B, and C) is used. Referring to Figure 10, conveyor X will be moving if the output $ConvX$ (with X belonging to A, B, and C) is active (places $p1$, $p2$, and $p3$ marked). The movement of each conveyor will stop whenever transitions $t1$, $t2$, or $t3$ fire. These transitions have an Up event associated with three dedicated events (generated by the activation of a limit switch). Whenever the three components arrive at the packing area, the operator activates a pedal associated with signal d, which in turn generates the $UpDownTrigger$ event associated with transition $t4$, launching the packaging mechanism and returning afterward to the initial state after releasing the pedal.

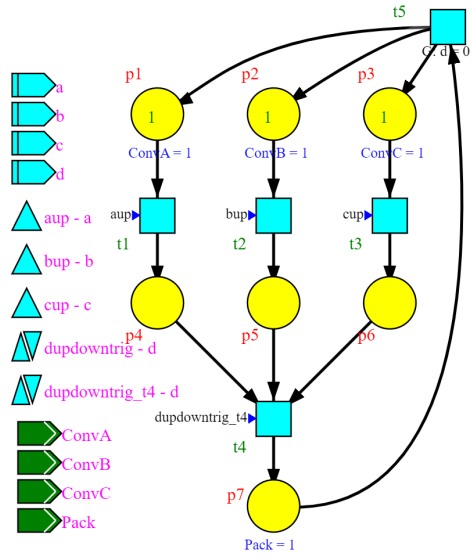

Figure 10. Packing controller.

It is also important to note that if the use of events is not available, the model of the system would be similar to the one presented in Figure 11a, composed of twelve places and ten transitions. By using events, the model of the system can be described employing seven places and five transitions, as presented in Figure 10 and the top left of Figure 11b. In this sense, the user benefits from the compactness of the model and improved readability. However, from the point of view of the execution of the model, the five models presented in Figure 11b are executed concurrently (even the user only sees the one in Figure 10).

The proposed event modeling strategy is foreseen to be supported by the IOPT-Tools framework, freely available at http://gres.uninova.pt/IOPT-Tools/ (accessed on 3 June 2024).

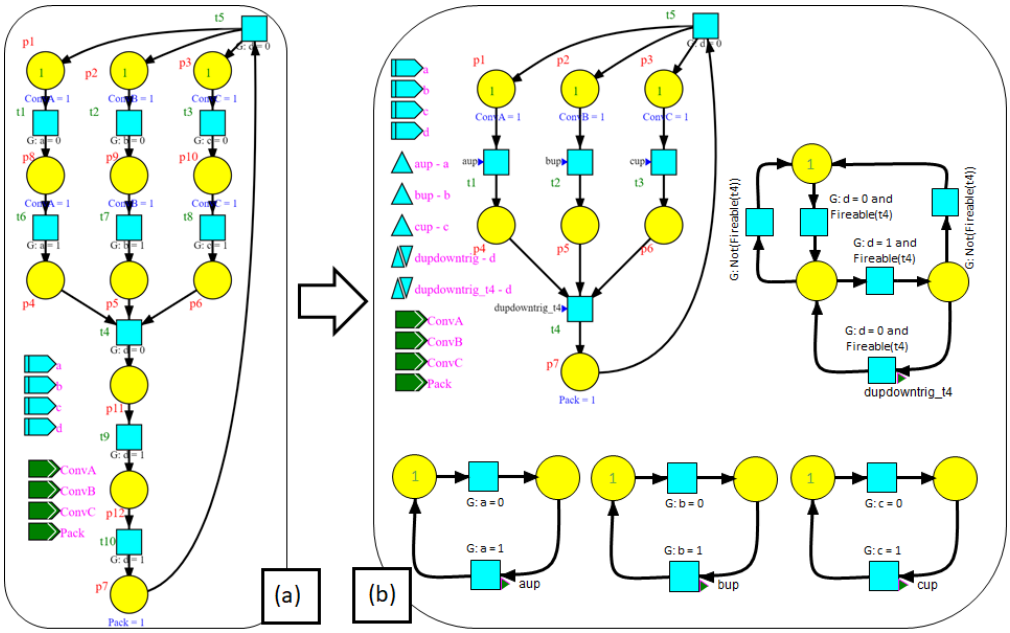

Figure 11. Comparing (**a**) the unfolded model using the conditions on the signals with (**b**) the model using events.

7. Conclusions and Future Work

When graphical representations of modeling formalisms are in use for the description of controller behaviors, relying on events to encapsulate the detection of a specific evolution

in the controller's input signals can contribute to achieving higher levels of compactness of the model, improving its expressiveness and having a positive impact on its readability.

The approach proposed in this paper, supported by (hidden) sub-models generating events associated with the specific evolution of input signals, is in line with this strategy and is entirely supported by a web-based tools framework (IOPT-Tools), which is freely available and covers all the steps of controller development, including the automatic generation of the execution code. This approach results in a more compact controller model that enhances its comprehensibility and readability. To our knowledge, no other Petri net-based tool frameworks exist that utilize events to encapsulate signal evolution behavior while maintaining the ability to automatically generate executable code for a wide range of platforms, including software implementations (C, with Python planned), hardware implementations (VHDL), and PLC platforms, benefiting from the translation of the underlying Petri net model into ladder diagrams.

It is important to note that the proposed approach, even presented within a Petri nets-based development framework, can also be used in cooperation with other discrete-event state-based modeling formalisms, namely, concurrent state machines, statecharts, and other classes of non-autonomous Petri nets. As a matter of fact, all the presented sub-models can be seen as state diagrams represented using a Petri net notation.

In future work, the definition of additional composed events based on the analysis of several signals (such as the ones suggested in this paper) is worth analyzing with a possible significant impact on the compactness of the behavioral models related to embedded controllers.

Author Contributions: Conceptualization, L.G.; methodology, L.G.; software, D.N.; validation, L.G., D.N., A.C., J.-P.B. and R.C.-R.; writing—original draft preparation, L.G., A.C. and J.-P.B.; writing—review and editing, D.N., A.C., J.-P.B. and R.C.-R. All authors have read and agreed to the published version of the manuscript.

Funding: This work was funded by the Portuguese Agency "Fundação para a Ciência e a Tecnologia" (FCT) program, in the framework of project Center of Technology and Systems (CTS) UIDB/00066/2020 / UIDP/00066/2020.

Data Availability Statement: The original contributions presented in the study are included in the article, further inquiries can be directed to the corresponding author.

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

References

1. Gomes, L.; Barros, J.P.; Costa, A.; Nunes, R. The Input-Output Place-Transition Petri Net Class and Associated Tools. In Proceedings of the 2007 5th IEEE International Conference on Industrial Informatics, Vienna, Austria, 23–27 June 2007; Volume 1, pp. 509–514. [CrossRef]
2. Gomes, L.; Moutinho, F.; Pereira, F.; Ribeiro, J.; Costa, A.; Barros, J.P. Extending Input-Output Place-Transition Petri nets for Distributed Controller Systems development. In Proceedings of the ICMC 2014—International Conference on Mechatronics and Control, Jinzhou, China, 3–5 July 2014; pp. 1099–1104. [CrossRef]
3. Gomes, L.; Barros, J.P. Refining IOPT Petri Nets Class for Embedded System Controller Modeling. In Proceedings of the IECON 2018—44th Annual Conference of the IEEE Industrial Electronics Society, Washington, DC, USA, 21–23 October 2018; pp. 4720–4725. [CrossRef]
4. Gomes, L.; Moutinho, F.; Pereira, F. IOPT-tools—A Web based tool framework for embedded systems controller development using Petri nets. In Proceedings of the 2013 23rd International Conference on Field Programmable Logic and Applications, Porto, Portugal, 2–4 September 2013. [CrossRef]
5. Pereira, F.; Moutinho, F.; Gomes, L. IOPT-Tools—Towards cloud design automation of digital controllers with Petri nets. In Proceedings of the ICMC'2014—International Conference on Mechatronics and Control, Shenyang, China, 29–31 August 2014.
6. Pereira, F.; Moutinho, F.; Costa, A.; Barros, J.P.; Campos-Rebelo, R.; Gomes, L. IOPT-Tools—From Executable Models to Automatic Code Generation for Embedded Controllers Development. In Proceedings of the PETRI NETS 2022: Application and Theory of Petri Nets and Concurrency, Bergen, Norway, 19–24 June 2022; pp. 127–138. [CrossRef]
7. Campos-Rebelo, R.; Costa, A.; Gomes, L. Graphical Formalism for Signal Interpretation Modeling. In *Recent Advances in Intelligent Engineering: Volume Dedicated to Imre J. Rudas' Seventieth Birthday*; Kovács, L., Haidegger, T., Szakál, A., Eds.; Springer International Publishing: Cham, Switzerland, 2020; pp. 139–154. [CrossRef]

8. Luckham, D. *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*; ADDISON WESLEY Publishing Company Incorporated: Boston, MA, USA, 2002.
9. Berry, G.; Cosserat, L. The ESTEREL Synchronous Programming Language and Its Mathematical Semantics. In Proceedings of the LNCS 197, Pittsburgh, PA, USA, 9–11 July 1984; Springer: Berlin/Heidelberg, Germany, 1985.
10. Harel, D. Statecharts: A visual formalism for complex systems. *Sci. Comp. Programm.* **1987**, *8*, 231–274. [CrossRef]
11. Girault, C.; Valk, R. *Petri Nets for System Engineering: A Guide to Modeling, Verification, and Applications*; Springer: Berlin/Heidelberg, Germany, 2001.
12. Gomes, L.; Campos-Rebelo, R.; Costa, A.; Barros, J.P. Input Event Modeling for Discrete-Event Controllers: A Petri Net Approach. In Proceedings of the CONTROLO'2022—15th APCA International Conference on Automatic Control and Soft Computing, Lisbon, Portugal, 26–30 June 2022.
13. Petri, C.A. Kommunikation mit Automaten. Ph.D. Thesis, Universität Hamburg, Hamburg, Germany, 1962.
14. Desel, J.; Juhás, G. "What Is a Petri Net?". In *Unifying Petri Nets, Advances in Petri Nets*; Lecture Notes in Computer Science; Ehrig, H., Juhás, G., Padberg, J., Rozenberg, G., Eds.; Springer: Berlin/Heidelberg, Germany, 2001; Volume 2128, pp. 1–25. [CrossRef]
15. Silva, M. *Las Redes de Petri: En la Automática y la Informática*; Editorial AC Madrid: Madrid, Spain, 1985.
16. Gomes, L.; Steiger-Garção, A. Programmable controller design based on a synchronized colored Petri net model and integrating fuzzy reasoning. In Proceedings of the 16th International Conference on Application and Theory of Petri Nets (ICATPN'95), Torino, Italy, 26–30 June 1995.
17. Holloway, L.E.; Krogh, B.H.; Giua, A. A Survey of Petri Net Methods for Controlled Discrete Event Systems. *Discrete Event Dyn. Syst.* **1997**, *7*, 151–190. [CrossRef]
18. David, R.; Alla, H. *Petri Nets & Grafcet; Tools for Modelling Discrete Event Systems*; Prentice Hall International (UK) Ltd.: London, UK, 1992.
19. Hanisch, H.M.; Lüder, A. A Signal Extension for Petri Nets and its Use in Controller Design. *Fundamenta Inf.* **2000**, *41*, 415–431. [CrossRef]
20. David, R.; Alla, H. *Discrete, Continuous, and Hybrid Petri Nets*, 2nd ed.; Springer: Berlin/Heidelberg, Germany, 2010.
21. Venkatesh, K.; Zhou, M.; Caudill, R.J. Comparing Ladder Logic Diagrams and Petri Nets for Sequence Controller Design through a Discrete Manufacturing System. *IEEE Trans. Ind. Electr.* **1994**, *41*, 611–619. [CrossRef]
22. Grobelna, I.; Adamski, M. Model checking of Control Interpreted Petri Nets. In Proceedings of the 18th International Conference Mixed Design of Integrated Circuits and Systems—MIXDES 2011, Gliwice, Poland, 16–18 June 2011; pp. 621–626.
23. Moreira, M.; Botelho, D.; Basilio, J. Ladder diagram implementation of Control Interpreted Petri Nets: A state equation approach. In Proceedings of the 4th IFAC Workshop on Discrete-Event System Design, Valencia, Spain, 6–8 October 2009.
24. Reisig, W. *Petri Nets: An Introduction*; Springer: Berlin/Heidelberg, Germany, 1985.
25. Campos-Rebelo, R.; Costa, A.; Gomes, L. On Structuring Events for IOPT Net Models. In *Technological Innovation for the Internet of Things*; Camarinha-Matos, L.M., Tomic, S., Graça, P., Eds.; Springer: Berlin/Heidelberg, Germany, 2013; pp. 229–238.
26. Murata, T. Petri Nets: Properties, Analysis and Applications. *Proc. IEEE* **1989**, *77*, 541–580. [CrossRef]