Refining IOPT Petri Nets class for embedded system controller modeling

Luís Gomes
Universidade Nova de Lisboa
and UNINOVA-CTS
Portugal
E-mail: lugo@fct.unl.pt

João Paulo Barros
Instituto Politécnico de Beja
and UNINOVA-CTS
Portugal
E-mail: jpb@uninova.pt

***Abstract*—Since its inception, the Input-Output Place-Transition (IOPT) class of Petri nets have changed in response to the gained experience in the use of its associated tools freely available as a cloud based toolset. Here, we informally present the current state of the IOPT net class as supported by the IOPT-Tools, publicly available at http://gres.uninova.pt/IOPT-Tools/. The corresponding formal syntax and semantics are presented, followed by an illustrative example. Finally, we give a brief presentation of the respective XML-based interchange format and conclude.**

***Index Terms*—model-driven development, Petri nets, design tools, microcontrollers.**

I. Introduction

Non-autonomous Petri nets offer a convenient and expressive language to model controllers and their connection with the environment. In spite of having been proposed in several variants along the last decades (e.g. [2]–[4], [7], [8], [15], [16]) there are still very few tools that support them, especially when we consider code generation for programmable hardware or even code to be compiled and executed on common microcontrollers [12].

The class of non-autonomous Petri nets named Input-Output Place-Transition nets have changed significantly since its inception [5]. All changes resulted from the identification of modelling needs and conveniences made possible by the availability of the associated tools, integrated in a cloud-based tool framework named IOPT-Tools [6], [11], publicly available at http://gres.uninova.pt/IOPT-Tools/. As it is common, the use of a language promotes and even forces their change and growth. Here, we present a revised definition of the IOPT nets class and illustrate it with an example model together with the respective XML-based interchange format generated by the IOPT-Tools. The following section summarizes the main changes in the IOPT nets class, as supported by the IOPT-Tools. Section III, describes and formally defines the current version of the IOPT net class. Section IV presents an illustrative example together with the XML-based format supported by IOPT-Tools and based on the PNML specification [13]. Finally, we present some conclusions and pointers for future developments.

II. IOPT nets

This section presents the formal syntax and semantics of the present version of the IOPT nets that updates an early one presented elsewhere [5].

IOPT nets are an extension of Place-Transition nets [14] with non-autonomous constructs. These allow the explicit modeling of the interface between the net model and the environment. The net models a controller and the interface with the environment is specified by the explicit modelling of input and output signals and events. This interface is based on the interpreted and synchronized nets of René David, Hassane Alla, and Manuel Silva [2], [3], [15]. Yet, non-autonomous extensions have also been proposed in more specific settings, namely on factory automation applications (e.g. [4], [8], [16]).

Compared to the IOPT nets in [5], the present version as implemented in the IOPT-Tools toolset has the following differences and additions:

1) Actions in transitions that assign values to output signals; these allow the change of output signal values when transitions fire;
2) Possibility to manually assign of physical input and output pins to input and output signals, respectively; this allows a low-level specification at model-level, relevant to automatic code generation;
3) Option to wrap limits in signal values (circular counter over the range of values in the respective domain); as output events increment/decrement associated output signal values, the circular counter behaviour can be extremely convenient;
4) Autonomous input and output events that are independent of any signal, useful for simulations and/or for inter-subsystem communication;
5) Output signals are also global variables and belong to the system state;
6) Support for modelling and automatic code generation for Globally Asynchronous Locally Synchronous (GALS) systems [1], [10], which will is not covered in this paper due to lack of space:
    a) Specify a time domain for any transition or place (the nodes associated with a specific time domain are intended to be associated with one specific implementation platform);
    b) A new type of arcs, named *channel arcs*;
    c) A new type of node having place semantics, repre-

senting Communication Channels and depicted as a small cloud, interconnected through one incoming *channel arc*, and one or more outgoing *channel arcs*. Five types of Communication Channels are available: two of them ("Synchronous Set", and "Asynchronous Set") allow the interconnection between sub-models having different time domains, while the other three ("Simple AC", "Acknowledged AC", and "Not-enabled AC") restrict the outgoing *channel arcs* to be connected to transitions having the same time domain.

The following section presents the formal definition of the IOPT nets class.

III. IOPT NETS DEFINITION

This section formally defines the syntax and semantics of IOPT nets as implemented in IOPT-Tools. It starts by the environment characterization, including the interface and input state. After, the net syntax is presented, followed by the net semantics.

A. Environment

Compared to an autonomous Petri net, from the IOPT modeller point of view, the environment imposes additional constraints on the net behaviour. This is based on input signals and events. Also, the net can actuate on the environment, and this is made possible by the specification of output signals and events. All these signals and events make the *model interface* (similar to the *system interface* in [5]). The model interface should be seen as a set of *interface shared phenomena* as presented by Michael Jackson [9]. The signals and events are shared between two domains: the net model (the controller) and the controlled system, or, using Michael Jackson's terminology, between the *machine world* and the *problem world*, respectively.

Definition 1 (Model interface): (based on [5]) 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: $inputSignal : IE_{NA} \to (IS_B \cup IS_R) \times \{upper, lower\}$;
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: $outputSignal : OE_{NA} \to (OS_B \cup OS_R) \times \{upper, lower\}$;
11) $IS_B \cap IS_R \cap IE_{NA} \cap IE_A \cap OS_B \cap OS_R \cap OE_{NA} \cap OE_A = \emptyset$.

Each non-autonomous event has an associated signal. We write $signal(e)$ to denote the signal associated to a non-autonomous event e and similarly for the set of signals associated to a set of events: $signals(ES)$, where ES is a set of non-autonomous events.

The model assumes a cycle accurate execution. Hence, in the beginning of each cycle, the input signal and input events are acquired at the same instant in time; we call the resulting data, the *system input state* as it reflects the controlled system state at a given moment (see Def. 2).

Definition 2 (System input state): Given a model interface $MI = (IS_B, IS_R, IE_{NA}, IE_A, inputSignal, OS_B, OS_R, OE_{NA}, OE_A, outputSignal)$, a system input state is defined by a tuple $SIS = (ISB_B, ISB_R, IEB_{NA}, IEB_A)$ satisfying the following requirements:

1) ISB_B is a finite set of Boolean input signal bindings: $ISB_B \subseteq IS_B \times \mathbb{B}$;
2) ISB_R is a finite set of range input signal bindings: $ISB_R \subseteq IS_R \times \mathbb{N}_0$;
3) IEB_{NA} is a finite set of non-autonomous input event bindings: $IEB_{NA} \subseteq IE_{NA} \times \mathbb{B}$;
4) IEB_A is a finite set of autonomous input event bindings: $IEB_A \subseteq IE_A \times \mathbb{B}$.

B. Syntax

The following definition assumes the use of an inscription language for algebraic expressions and variables. The set of Boolean expressions is named $BExp$ and the set of integer expressions, that evaluate to non-negative integer, is named $IExp$; $BES \subseteq BExp$ and $IES \subseteq IExp$, the function $Var(E)$ returns the set of variables in a given expression e, and ML is the set of identifiers for each place marking.

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

1) P is a finite set of places;
2) C is a finite set of channels, a special type of places for communication;
3) T is a finite set of transitions such that $P \cap C \cap T = \emptyset$;
4) A is a set of arcs, such that $A \subseteq ((P \times T) \cup (T \times P))$;
5) TA is a set of test arcs, such that $TA \subseteq (P \times T)$;
6) CA is a set of channel arcs, such that $CA \subseteq ((C \times T) \cup (T \times C))$;
7) M is the marking function: $M : P \to \mathbb{N}_0$;
8) $weight$ is the arc weight function: $weight : A \to \mathbb{N}_0$;
9) $weightTest$ is the test arc weight function: $weightTest : TA \to \mathbb{N}_0$;

10) $priority$ is a partial function applying transitions to non-negative integers: $priority : T \rightharpoonup \mathbb{N}_0$; we write $priority(t) \uparrow$ when transition t has no defined priority;
11) $guard$ is an *guard* partial function applying transitions to Boolean expressions (where all variables are markings or signals): $guard : t \rightharpoonup BE$, where $\forall eb \in guard(t), Var(eb) \subseteq (M \cup IS_B \cup IS_R)$;
12) ie is an input event function applying transitions to input events: $ie : T \rightarrow \mathcal{P}(IE_{NA} \cup IE_A)$;
13) oe is an output event function applying transitions to output events: $oe : T \rightarrow \mathcal{P}(OE_{NA} \cup OE_A)$;
14) 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 $\forall iae \in IES, Var(iae) \subseteq (ML \cup OS_B \cup OS_R)$.
15) osc is an output signal condition function from places into sets of rules: $osc : P \rightarrow \mathcal{P}(RULES)$, where $RULES \subseteq ((OS_B \cup OS_R) \times BES \times \mathbb{N}_0)$, and $\forall e \in BES, Var(e) \subseteq (ML \cup IS_R \cup IS_B \cup OS_R \cup OS_B)$.

Places have markings just like in Place-Transition nets. They also have a set of actions. Each of these, allows the assignment of values to output signals depending on the markings and signal values.

Transitions can have associated guards, actions, priorities, input, and output events. Guards are functions of markings and signals. Guards and also input events need to be true to allow transitions to fire. Non-autonomous output events can increment/decrement associated output signals (depending on the upper/lower attribute), optionally wrapping up at the higher or lower values. Autonomous output events are activated whenever associated transition(s) fire(s). Priorities allow conflict resolution among transitions in conflict.

The output signals assigned by place actions must be disjoint from the output signals assigned by transition actions and transition events: if $POS \times E1 \times \mathbb{N} = osc(P)$ and $TOS \times E2 = ta(T)$ then $signals(oe(T) \cup TOS) \cap POS = \emptyset$.

IOPT nets also have a special type of places, named *(communication) channels*. These support the specification of GALS systems [10], as well as the decomposition of the model into a set of sub-models, following the *net split* operation [1]. The connections between these and the transitions is made by a special type of arc: *channel arc*.

C. Semantics

Compared to Place-Transition nets, IOPT nets have two major semantic differences: (1) the dependency on external conditions (signals and events) and (2) the deterministic nature of each execution step.

The dependency on external conditions, i.e., the non-autonomous nature of IOPT nets is reflected in the firing rule for its transitions. Two pre-conditions are required: one related to the autonomous part and the other due to the non-autonomous component. Hence, for a transition to fire it has to be *enabled* just like in Place-Transition nets, and it also has to be *ready*, i. e. the external conditions must allow it to fire.

The non-deterministic nature of step selection in Place-Transition nets, where any subset of enabled transitions can fire, is undesirable for controller modelling. For that reason, IOPT nets have a maximal step semantics: whenever a transition is enabled and ready, the transition is fired. As all transitions fire at the same instant, we also say that the net fires all the enabled and ready transitions. The only possible exclusions are due to conflicts. Also, net firing is only possible at specific instants in time named *tics*. These are imposed by an external global clock. The period between *tics* is called *execution step*.

In the following definitions $M(p)$ denotes the marking of place p in a net with marking M, $\bullet t$ denotes the input places of a given transition t and $\bullet S$ the input places of a given set of transitions S, connected by normal arcs. Similarly, the operator $\diamond$ is used for input places connected by test arcs: $\bullet t = \{p|(p,t) \in A\}$, $\bullet S = \{p|(p,t) \in A \wedge t \in S\}$, $\diamond t = \{p|(p,t) \in TA\}$, $\diamond S = \{p|(p,t) \in TA \wedge t \in S\}$.

Definition 4 (Ready transition): Given a net $N = (P, C, T, A, TA, CA, M, weight, weightTest, priority, guard, ie, oe, ta, osc)$ and a model interface $MI = (IS_B, IS_R, IE_{NA}, IE_A, inputSignal, OS_B, OS_R, OE_{NA}, OE_A, outputSignal)$ between N and a system input state $SIS = (ISB_B, ISB_R, IEB_{NA}, IEB_A)$, a transition t, is *ready* to fire with system input state SIS, denoted $ready(t, SIS)$, iff the following conditions are satisfied:

1) The transition t input signal guard evaluates to true for the given input signal binding: $guard(t){<}ISB_B \cup ISB_R{>}$.
2) $\forall e \in ie(t), (ie(t), true) \in (IEB_{NA} \cup IEB_A)$.

Definition 5 (Enabled transition): Let $N = (P, C, T, A, TA, CA, M, weight, weightTest, priority, guard, ie, oe, ta, osc)$ be a net and $MI = (IS_B, IS_R, IE_{NA}, IE_A, inputSignal, OS_B, OS_R, OE_{NA}, OE_A, outputSignal)$ a model interface between N and a system input state $SIS = (ISB_B, ISB_R, IEB_{NA}, IEB_A)$. A transition t, with no structural conflicts, is enabled to fire with the current net marking M, denoted $enabled(t, M)$, iff the following conditions are satisfied:

1) $\forall p \in \bullet t, M(p) \geq weight(p, t)$.
2) $\forall p \in \diamond t, M(p) \geq weightTest(p, t)$.

Definition 6 (Conflict sets): A conflict set CS is a set of transitions in structural conflict: $CS \in (\mathcal{P}(T) \backslash \emptyset) \wedge \forall t_1 \in CS, \exists t_2 \in CS \backslash \{t_1\} : \bullet t_1 \cap \bullet t_2 \neq \emptyset$. We denote by SCS the set of all conflict sets in a net. All conflict sets are disjunct: $\forall CS_1, CS_2 \in SCS : CS_1 \cap CS_2 = \emptyset$. Additionally, $CS(t)$ is the conflict set that contains t: $CS(t) \in SCS \implies t \in (CS(t) \cap T)$.

Definition 7 (IOPT net step): Let $N = (P, C, T, A, TA, CA, M, weight, weightTest, priority, guard, ie, oe, ta, osc)$ be a net and $MI = (IS_B, IS_R, IE_{NA}, IE_A, inputSignal, OS_B, OS_R, OE_{NA}, OE_A, outputSignal)$ a model interface between N and a system input state $SIS = (ISB_B, ISB_R, IEB_{NA}, IEB_A)$. Let $ET \subseteq T$ be the set of all ready and enabled transitions as defined by Def. 4 and 5: $ET = \{t | t \in T \wedge ready(t, SIS) \wedge enabled(t, M)\}$. Then, Y is a IOPT net step of N iff the following condition is satisfied:

$$Y \subseteq ET \wedge \forall t_1 \in (ET \backslash Y), \exists SY \subseteq Y, (\bullet t_1 \cap \bullet SY) \neq \emptyset \wedge \exists p \in (\bullet t_1 \cap \bullet SY), (weight(p, t_1) + \sum_{t \in SY} weight(p, t) > M(p))$$

The condition states that when a ready and enabled transition is not in the maximal step that is due to insufficient markings in the input places for all the transitions in its conflict set ($CS(t)$) to be enabled.

Next, we define a IOPT net step occurrence and the respective successor marking.

Definition 8 (Step occurrence and successor marking): Let $N = (P, C, T, A, TA, CA, M, weight, weightTest, priority, guard, ie, oe, ta, osc)$ be a net and $MI = (IS_B, IS_R, IE_{NA}, IE_A, inputSignal, OS_B, OS_R, OE_{NA}, OE_A, outputSignal)$ a model interface between N and a system input state $SIS = (ISB_B, ISB_R, IEB_{NA}, IEB_A)$. The occurrence of a step Y in net N returns the net $N' = (P, C, T, A, TA, CA, M', weight, weightTest, priority, guard, ie, oe, ta, osc)$, equal to the net N except for the successor marking M' which is given by the following expression:

$$M' = \left\{ \left(p,\ m - \sum_{t \in Y \wedge (p,t) \in A} weight(p, t) + \sum_{t \in Y \wedge (t,p) \in A} weight(t, p)\right) \in (P \times \mathbb{N}_0) \;\middle|\; (p, m) \in M \right\}$$

D. Net execution

A high level specification for the net step execution is presented in the following procedure. In each step the input signals values are read and the input events updated based on the change of the value of the associated signal. The output event buffer (OEB) and the temporary marking buffer are reset and the enabled transitions are fired. The marking is updated with the new one which is then used to execute the actions in each place.

```
1:  procedure ExecuteNetSteps
2:      loop
3:          $IS_B \leftarrow readBooleanInputSignals()$
4:          $IS_R \leftarrow readRangeInputSignals()$
5:          $IE_{NA} \leftarrow computeInEvents(IS_B, IS_R)$
6:          $IE_A \leftarrow readAutonomousInputEvents()$
7:          $resetOEB()$
8:          $resetTemporaryMarkingBuffer()$
9:          $fireMaximalStep()$
10:         $netMarking \leftarrow temporaryMarkingBuffer$
11:         for $place \in P$ do
12:             $executeActions(place)$
13:         end for
14:     end loop
15: end procedure
```

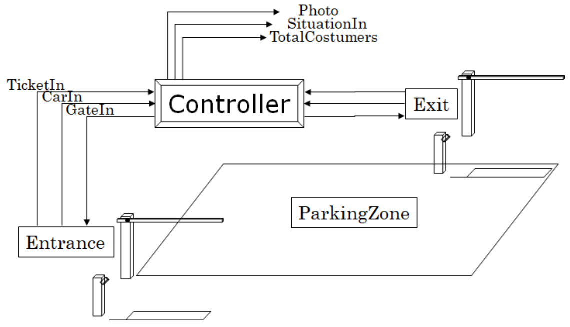

Fig. 1. Parking lot layout.

IV. Example model

A simple validation example is used to illustrate some of the modelling capabilities of IOPT nets (and benefiting from using IOPT-Tools framework). A parking lot controller is considered having one entrance and one exit, and a specific amount of parking places, as in [5]. However, for the presented example, only the entrance model is considered (similar strategy could be used to model the exit part).

Fig. 1 presents the layout of the parking lot, where the controller allows the entrance of cars activating *GateIn*, taking into consideration availability of free places inside the parking lot and the analysis of the time evolution of *CarIn* and *TicketIn* input signals.

Whenever a car is present at the entrance, input signal *GateIn* is activated, and the controller waits for the user to pick *TicketIn* to activate the *GateIn*. Considering the model presented at Fig. 2, this basic flow is modeled using the set of nodes *EntranceFree*, *ArrivingCar*, *WaitingTicket*, *EnteringRegular*, *GateOpen*, *CarEntered* (as far as there is some tokens in *FreePlaces*).

Some deviations to the described basic behavior are foreseen:

- The driver can go reverse after entering in the entrance area.
- The driver can block the entrance (due to some problem) for a period longer than expected.

To adequately handle the referred situations, the usage of events is foreseen resulting in a model much more compact with improved expressiveness. This is the case of the events *CarInEv* and *CarOutEv*, both associated with changes on the input signal *CarIn* (activation and deactivation of the signal, respectively). In particular, the event *CarOutEv* can be used to return the model to *EntranceFree* global state for the two referred situations plus the regular behavior. Also, the usage of output events, such as *photo* and *TotalCostumers*, which will be generated whenever the transitions fire, allows simple modeling of the behaviors triggering external sub-systems. Coming to output signals, they can be associated with places

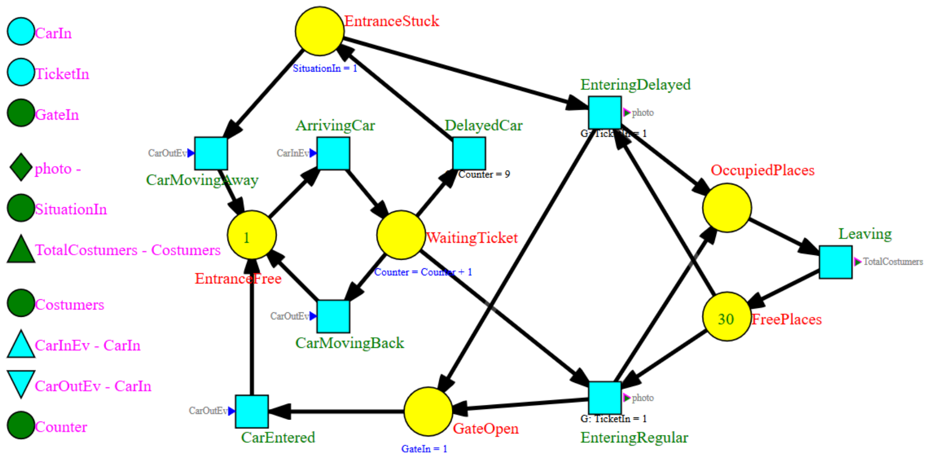

Fig. 2. Screen capture of the example model as seen in the IOPT Tools toolset.

(as in *GateOpen*, where output *GateIn* is activated, or as in *WaitingTicket*, where signal **Counter**, which is used as a global variable, is updated), or with transitions (as in *Leaving* generating the output event *TotalCostumers*).

Next, we briefly present the used interchange format.

V. Interchange Format

In this section we briefly exemplify the XML-based interchange format used by IOPT Tools. The used format is an extension of the PNML (Petri Net Markup Language) metamodel for Place-Transition nets.

Listing 1 shows part of the signals and events specification. They are between an input and an output section. Each one contains signals and events. The same listing shows the beginning of the file and an example of the `graphics` element. The remaining occurrences of those elements were omitted for brevity. In this and all the following Listings, the omitted parts are signaled with three dots.

Listing 1. Signals and events specification.

```
<?xml version="1.0" encoding="windows-1252"
standalone="no"?>
<Snoopy revision="0" version="1.1">
  <pnml>
    <net id="1" name="iecon18" type="IOPT">
      <input>
        <signal id="CarIn" type="boolean" value="0"
                gpio_nr="0">
          <graphics>
            <position page="1" x="50" y="60"/>
          </graphics>
        </signal>
        <signal id="TicketIn" type="boolean" value="0"
                gpio_nr="0">
          ...
        </signal>
        <event id="CarInEv" edge="up" level="0"
               signal="CarIn">
          ...
        </event>
         ...
      </input>
      <output>
        <signal id="Counter" type="range" value="0"
                gpio_nr="0" min="0" max="9" wrap="0">
          ...
        </signal>
        <event id="photo" autonomous="true">
          ...
        </event>
        <signal id="GateIn" type="boolean" value="0"
                gpio_nr="0">
          ...
        </signal>
         ...
        <signal id="Costumers" type="range" value="0"
                gpio_nr="0" min="0" max="255" wrap="0">
          ...
        </signal>
        <event id="TotalCostumers" edge="up" level="0"
               signal="Costumers">
          ...
        </event>
      </output>
```

Listing 2 illustrates the PNML for two places. They show examples of initial markings and actions, one of them acting on an output signal named *Counter*.

Listing 2. Places.

```
 ...
<place id="3">
    <name>
      <text>WaitingTicket</text>
       ...
    </name>
     ...
    <initialMarking>
      <text>0</text>
       ...
    </initialMarking>
    <bound>
      <text>3</text>
    </bound>
    ...
    <signalOutputActions>
      <signalOutputAction idRef="Counter">
        <value>
          <concreteSyntax language="iopt">
            <text>Counter + 1</text>
            <expression>
              <operand type="output-signal"
                       idRef="Counter" seq="1"/>
              <operation operator="addition" seq="2">
                <operand type="literal" value="1"
                         seq="3"/>
              </operation>
            </expression>
          </concreteSyntax>
        </value>
        <condition>
          <concreteSyntax language="iopt">
            <text/>
```

```
                </concreteSyntax>
              </condition>
            </signalOutputAction>
          </signalOutputActions>
        </place>

        <place id="4">
          <name>
            <text>GateOpen</text>
            ...
          <signalOutputActions>
            <signalOutputAction idRef="GateIn">
              <value>
                <concreteSyntax language="iopt">
                  <text>1</text>
                  <expression>
                    <operand type="literal" value="1"
                             seq="1"/>
                  </expression>
                </concreteSyntax>
              </value>
              <condition>
                <concreteSyntax language="iopt">
                  <text/>
                </concreteSyntax>
              </condition>
            </signalOutputAction>
          </signalOutputActions>
        </place>
        ...
```

Listing 3 illustrates the PNML for two transitions. They show examples of a guard, input event, and output event.

Listing 3. Examples of transitions specifications.

```
        ...
        <transition id="6">
          <name>
            <text>ArrivingCar</text>
            ...
          </name>
          ...
          <priority>1</priority>
          <signalInputGuards/>
          <inputEvents>
            <event idRef="CarInEv"/>
          </inputEvents>
          ...
          <outputEvents/>
          ...
        </transition>

        <transition id="5">
          <name>
            <text>EnteringRegular</text>
            ...
          </name>
          ...
          <priority>1</priority>
          <signalInputGuards>
            <signalinputguard>
              <concreteSyntax language="iopt">
                <text>TicketIn = 1</text>
                <expression>
                  <operand type="input-signal"
                           idRef="TicketIn" seq="1"/>
                  <operation operator="equal" seq="2">
                    <operand type="literal" value="1"
                             seq="3"/>
                  </operation>
                </expression>
              </concreteSyntax>
            </signalinputguard>
          </signalInputGuards>
          <inputEvents/>
          ...
          <outputEvents>
            <event idRef="photo"/>
          </outputEvents>
          ...
        </transition>
          ...
```

VI. Conclusions

The free availability of IOPT-Tools at http://gres.uninova.pt/IOPT-Tools/ has allowed the IOPT nets class and its use as a code generation platform to remain available, used, and useful. As a consequence and in response to identified needs, the corresponding metamodel has evolved leading to a significant number of differences between the initial version of the IOPT nets class and the currently available version in the IOPT-Tools. This paper updates the net class definition. As future work, the priority is to improve the usability of the user interface, to extend the code generation to additional platforms, and to simplify the adaptation to each platform specificity.

Acknowledgment

This work was partially financed by Portuguese Agency FCT Fundação para a Ciência e Tecnologia, in the framework of project UID/EEA/00066/2013.

References

[1] A. Costa and L. Gomes, “Petri net partitioning using net splitting operation,” in *2009 7th IEEE International Conference on Industrial Informatics*, June 2009.

[2] R. David and H. Alla, *Petri Nets & Grafcet; Tools for Modelling Discrete Event Systems*. Prentice Hall International (UK) Ltd, 1992.

[3] ——, *Discrete, Continuous, and Hybrid Petri Nets*, 2nd ed. Springer Publishing Company, Incorporated, 2010.

[4] G. Frey and M. Minas, “Editing, Visualizing, and Implementing Signal Interpreted Petri Nets,” in *Proceedings of the AWPN 2000, Koblenz*, Oct. 2000, pp. 57–62.

[5] L. Gomes, J. P. Barros, A. Costa, and R. Nunes, “The Input-Output Place-Transition Petri net class and associated tools,” in *2007 5th IEEE International Conference on Industrial Informatics*, vol. 1, June 2007, pp. 509–514.

[6] L. Gomes, F. Moutinho, and F. Pereira, “IOPT-tools – a web based tool framework for embedded systems controller development using Petri nets,” in *2013 23rd International Conference on Field programmable Logic and Applications*, Sept 2013, pp. 1–1.

[7] L. Gomes and A. Steiger-Garção, “Programmable controller design based on a synchronized colored Petri net model and integrating fuzzy reasoning,” in *16th International Conference on Application and Theory of Petri Nets (ICATPN'95), Torino, Italy*, jun 1995.

[8] H.-M. Hanisch and A. Lüder, “A Signal Extension for Petri Nets and its Use in Controller Design,” *Fundamenta Informaticae*, vol. 41, no. 4, pp. 415–431, 2000.

[9] M. Jackson, *Problem Frames: Analyzing and Structuring Software Development Problems*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2001.

[10] F. Moutinho and L. Gomes, *Distributed Embedded Controller Development with Petri Nets: Application to Globally-Asynchronous Locally-Synchronous Systems*, 1st ed. Springer Publishing Company, Incorporated, 2015.

[11] F. Pereira, F. Moutinho, and L. Gomes, “IOPT-Tools – towards cloud design automation of digital controllers with Petri nets,” in *ICMC'2014 - International Conference on Mechatronics and Control*, 2014.

[12] “Petri nets tool database,” http://www.informatik.uni-hamburg.de/TGI/PetriNets/tools/db.html, 2004.

[13] PNML, “PNML.org,” Accessed on 2018/06/29, http://www.pnml.org, 2015.

[14] W. Reisig, *Petri nets: an Introduction*. Springer-Verlag New York, Inc., 1985.

[15] M. Silva, *Las Redes de Petri: en la Automática y la Informática*. Madrid: Editorial AC, 1985.

[16] K. Venkatesh, M. Zhou, and R. J. Caudill, “Comparing Ladder Logic Diagrams and Petri Nets for Sequence Controller Design through a Discrete Manufacturing System,” *IEEE Transactions on Industrial Electronics*, vol. 41, no. 6, pp. 611–619, Dec. 1994.