Extending IOPT Nets with a Module Construct

José Ribeiro[1,2,3(✉)], Fernando Melício[1], and Luis Gomes[2,3]

[1] Instituto Superior de Engenharia de Lisboa, Instituto Politécnico de Lisboa, 1959-007 Lisbon, Portugal
{jribeiro,fmelicio}@deea.isel.ipl.pt
[2] Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa, 2829-516 Caparica, Portugal
lugo@fct.unl.pt
[3] UNINOVA - CTS, Monte de Caparica, 2829-516 Caparica, Portugal

Abstract. Input-output place-transition nets (IOPT nets) is a Petri net based formalism targeted for the development of embedded systems controllers. It is an extension to common place-transition Petri nets, introducing constructs to model the communication between the controller and the environment and using an execution semantics assuring a deterministic behavior. However, IOPT nets and the supporting tools framework - the IOPT-Tools - do not have a mechanism to support model structuring. Since models are flat, all the graphical components and annotations are visualized in the same page. Systems with several dozens of nodes become very difficult to manage. In this paper a modular construct for IOPT nets is presented, helping to manage large-scale systems, and the reuse of model components across projects. The algebraic specification of the model is provided and an example illustrating the concept is presented.

Keywords: Modularity · Composition · Low-level Petri nets · IOPT nets

1 Introduction

The development of embedded systems is a challenging task due to their strict requirements of safety, correctness and real time constraints, among others restrictions [1]. Embedded systems are dedicated computational devices, which controls a physical system through a communication interface. These systems are characterized by states, changing from one state to another by the occurrence of discrete events.

Among several formal methods to model discrete event systems, Petri nets have the advantage of allowing to model the structure and the dynamics of the system and are provided with powerful analysis methods [2]. Aspects as concurrent execution and synchronization of actions are naturally modeled with Petri nets. There are also a vast number of extensions that allow the modeling of distributed components or the reactive behavior of systems [3, 4]. The use of formal methods allows the analysis and validation of the system, which is crucial, in particular when the compliance with the requirements has to be assured previously to their implementation [5]. Formal methods, particularly those with a graphical representation, as Petri nets, also enable the implementation

L. Camarinha-Matos et al. (Eds.): DoCEIS 2016, IFIP AICT 470, pp. 86–95, 2016.
DOI: 10.1007/978-3-319-31165-4_9

of development approaches, based on models, that supports the entire development flow and the automation of some tasks.

The IOPT nets are a Petri net based modeling language, extending the place/transition Petri net class, addressed to the development of embedded systems controllers. Its main features are the constructs to explicitly model the communication between the net and the environment and a deterministic execution semantics. One aspect in which the Petri nets exhibit a major limitation is in the structuring of models. The representation of large-scale systems with flat models is highly inadequate. Modular mechanisms are essential for the expressivity and compactness of the models and to raise the efficiency of the modeling development process [6].

Concerning the IOPT nets, the lack of a structuring mechanism makes difficult the construction and management of large-scale and complex models. All the models must be built from scratch, with basic primitives, becoming the design task inefficient and preventing the reuse of model components into other projects.

Considering this, we state the following research question: *What structuring mechanisms should be defined to build generic subnets, reusable across different projects, allowing to manage complex and large models, while keeping all analysis capabilities?*

Hypothesis: *Modules with dynamic interfaces and configurable parameters, at instance level, allow the design of more compact models and its use in different situations. Converting a structured model into an equivalent flat model assures the use of existing analysis tools to validate the model.*

The modular mechanism presented here is an extension for the IOPT nets [7, 8] and is being implemented in the IOPT-Tools development framework [9] (available online at http://gres.uninova.pt/IOPT-Tools), the supporting tool for the IOPT nets. Figure 1 shows a diagram with the IOPT nets extensions and the IOPT-Tools features.

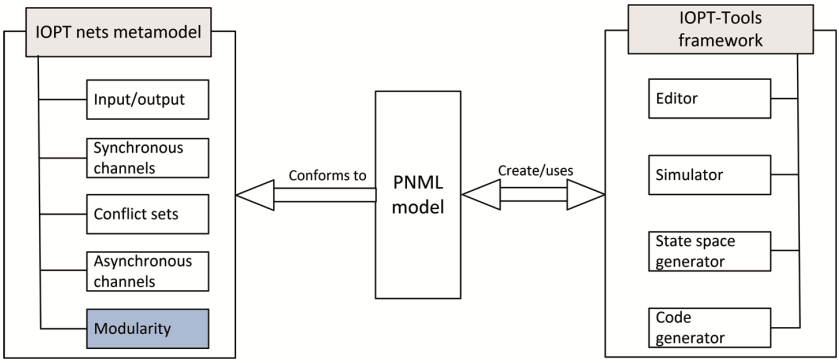

Fig. 1. IOPT Net and IOPT-Tools main features.

Concerning the proposing modular extension, the interface of the modules are nodes from the encapsulated net. Modules may be defined with more than one interface for being used in different situations. Several instances of the same module may be used in a containing net with different parameters.

The following Section of this paper cover the relationship of this work with the Cyber Physical Systems (CPS). In Sect. 3 it is reviewed some of the proposals of modular constructs that have been made in the context of Petri nets. The description of the modules for IOPT nets and its mathematical model is presented in Sect. 4. An example of a net with modules is presented in Sect. 5, followed by a discussion and conclusions.

2 Relationship to Cyber-Physical Systems

This work is about a modular extension for a modeling language (IOPT nets) addressed to the development of embedded systems controllers. An embedded system (ES) is a dedicated computational system embedded in a physical system which controls it. In [10] Cyber-Physical Systems (CPS) are defined as "the tight conjoining of and coordination between computational and physical resources". The main difference between ES and CPS is in the focus that is given to the physical system, being the CPS a more complete approach. While in embedded systems the main emphasis is on the computational part, from the point of view of a CPS an additional attention is given to the physical system, resulting in a greater integration of computational and physical systems. Although IOPT nets are suited to the development of ES controllers, the modular structuring presented here allows the modeling of the controller and the controlled physical system. Thus one can use the whole model for simulation and analysis purposes and the controller part of the model for the synthesis of the controller using a CPS development approach.

3 Related Literature

There are multiple languages for the development of embedded systems controllers [11], namely those where the communication with the controlled system has a fundamental role, as Grafcet [12] or Mark Flow Graphs [13].

In this section we overview some of the structuring mechanisms proposed in the field of Petri nets, namely those related with the modular construct presented in this work.

Many structuring mechanisms have been proposed, in order to provide Petri nets (PNs) with abstraction and composition capabilities, either for low-level PNs [14] or high-level PNs [15, 16]. Concerning composition, most of the proposals uses a module, or a similar entity, which models a small part of a system and provides an interface to communicate with the container net.

In some mechanisms the interface is composed by places or transitions, as Modular PNML [17, 18] and Hierarchical Coloured Petri nets (HCP nets) [16]. In some others it is used new elements, as events or signals in Signal Nets [19] and Net Condition Event Systems (NCES) [20].

The kind of communication can be synchronous, usually made through transition fusion, as in HCP [16] nets and modular PNML [17] or asynchronous, usually made by place fusion or by message sending, as in Object Petri nets (OP nets) [21].

Concerning instantiation it can be static or dynamic. Static instantiation has a substitution semantics, where an instance, or a macro, is substituted by the module. In this kind of instantiation the structure of the net remains unchanged during their execution and can be transformed into an equivalent unstructured net. Transition substitution in

HCP nets and modular PNML are examples of static instantiation. Dynamic instantiation consists in the creation of a module instance, during the execution of the net, and its subsequent destruction. The creation and destruction of the instance is governed by an event of the net, as the firing of a transition or a marking in a place. This change in net structure makes it very difficult to analyze. Dynamic instantiation is used in classes inspired in object-oriented paradigm as OP nets and Reference Nets [22]. The invocation transition mechanism of HCP nets also uses dynamic instantiation although it was never implemented in the CPN Tools, the supporting tool of the HCP nets.

4 Research Contribution and Innovation

This section contains the definition of the modules for IOPT nets. This mechanism uses nodes as interface and static instantiation. At first it is presented the definition of unstructured IOPT nets, their main features and semantics. Then the modular extension is presented and formalized.

4.1 IOPT Nets

As stated in the introduction, the IOPT nets have explicit constructs to model the communication with external devices. This communication is done through input and output signals and events. An IOPT net models a controller for an embedded system, thus must have a deterministic behavior. To accomplish this behavior it was adopted an execution semantics with maximal step combined with a single server semantics. Maximal step semantics means that all the transitions ready to fire, will fire in a given step. With single server semantics a transition fires only once in one step, even if it remains ready. The firing of transitions is synchronized with an external clock. A step is the occurrence of all the transitions ready to fire. A transition t, is enabled if for all its input places, $\bullet t$, $M(\bullet p) \geq w(p, t)$ with $(p, t) \in A$. An enabled transition is ready to fire if the transition guard is true and the event associated with the transition occurs.

Output events can be generated by the firing of a transition, and output signals can be updated by expressions evaluated by the firing of a transition, or by a specific marking of a place.

4.2 IOPT Net Mathematical Model

An IOPT net is defined as a tuple: $IOPT = (NG, IO, AN, Const, M0)$, where,

- NG is the net graph, $NG = (P, T, F)$ where,
  - $P \cup T$ are the set of nodes, satisfying the condition $P \bigcap T = \varnothing$,
  - F is the set of arcs which defines the flow relation. Each arc has a type from the set $AT = \{normal, test\}$, such that $F_{normal} \subseteq (P \times T) \cup (T \times P)$ and $F_{normal} \subseteq (P \times T)$.
- $IO = S_{in} \cup S_{out} \cup E_{in} \cup E_{out}$ is a finite set of input/output signals and input/output events.
- $AN = (A, TG, TIE, TOE, POS)$ is a finite set of annotations, where,

- $A : F \rightarrow \mathbb{N}$, is the weight function, assigning each arc with a positive integer.
- $TG : T \nrightarrow Exp_{(bool)}$ is a partial function that annotates transitions with boolean guard expressions.
- $TP : T \rightarrow \mathbb{N}$, is a function that annotates each transition with a priority value.
- $TIE : T \nrightarrow E_{in}$, is a partial function that annotates transitions with input events.
- $TOE : T \nrightarrow E_{out}$, is a partial function that annotates transitions with output events.
- $POS : P \nrightarrow (S_{out}, Exp_{(bool)})$ is a partial function that annotates places with output signals. The signal is updated when the place is marked and the boolean expression is true.

– *Const* is a set of symbolic constants whose values are naturals. A constant may be used as the initial marking of a place.
– $M_0 : P \rightarrow \mathbb{N}_0$, is the initial marking of the net, assigning a nonnegative integer to each place.

The transition *guard* and place *output expression* are built with operands from signal values, constants and place marking values (the number of dots), and operators from the set $O = \{O_{arithmetic}, O_{comparison}, O_{logic}\}$, with $O_{arithmetic} = \{+, -, *, /, \%\}$, being the usual arithmetic operators, $O_{comparison} = \{<, >, =, <=, >=, =\}$, the set of comparison operators and $O_{logic} = \{and, or, xor, not\}$ the set of logic operators.

4.3 Modules

An IOPT net uses the element page as the container of the net elements: graph elements, annotations and input/output declarations through which is done the communication with the environment. A module is also a container, with an encapsulated net, with one or more interfaces. The interface of a module is a subset of the module set of nodes.

A module is used in a net by their instance. An instance of a module is a copy of the module, represented in the net by a rectangle with the interface nodes. Each instance uses one interface. The interface nodes of the instance are copies of its counterpart in the module net.

An interface node may be of type input, output or, by default, input/output. Input nodes can only have incoming arcs from the container net. Output nodes can only be linked with outgoing arcs.

An IOPT net composed with modules has an equivalent at net, where the instances are substituted by their referenced modules. The interface nodes, from the instance, are fused with the respective nodes of the module. In the following definition of module it is considered a simple module, that is, a module without instances of other modules in its composition. A module is a structure $Module = (IOPT, ITF, IT)$ where,

– IOPT is a IOPT net as defined in Sect. 4.2.
– $ITF = itf_1, itf_2, \ldots$ is a set of interfaces, with, $itf_i : P \cup T \nrightarrow \{in, out, in/out\}$, each interface is a partial function that assigns nodes with an attribute *in*, *out*, *in/out*, for input, output or input/output nodes.
– IT: $S_{in} \cup E_{in} \cup Const \rightarrow \{module, instance\}$ is a function that assigns an attribute to each input signal, input event and symbolic constant declared in the module.

For signals and events, if the attribute is *module* the signal/event is unique and shared by all the instances. If the attribute is *instance*, each instance has its own input signal/event. Constants with the attribute *instance* may be overridden and have a different value in each instance. A constant with an attribute *module* cannot be overridden at instance level.

4.4 Composition of Modules

A module is used through instances. An instance is graphically represented by a rectangle, a round rectangle or an oval. The interface nodes are drawn over the instance object. The nodes in the instance are copies of the respective interface nodes of the module. The encapsulated net of a module may be composed with instance of other modules. However it is not allowed to define a module with instances of itself.

When more than one instance of the same module exists in the same net, input signals, input events and symbolic constants may be shared by all the instances (attribute *module*), or exist as separate entities in each instance (attribute *instance*). Output signals and output events are unique for each instance of the same module.

The equivalent plain net is obtained by substituting the instances by the net of the respective module. The interface nodes of the instance are fused with the nodes of the module and the element names are prefixed with the name of the instance.

5 Example

The following example illustrates the definition of modules and a net built with instances of that modules. The equivalent flat net, obtained from the conversion of the net with module instances into an equivalent net without module instances, shows the semantics of the modules. The example is adapted from [23]. It is used a parking lot with an entrance and an exit as shown in the Fig. 2. When a car is detected near the entrance gate, a ticket is printed. When the driver gets the ticket, the gate opens and he can enter

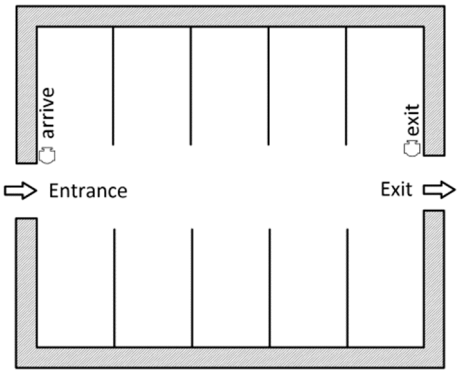

Fig. 2. Parking lot scheme.

the park. To leave the park, the car must be detected near the exit gate and after paying the ticket, the gate opens. The sensors *entrance* and *exit* detect the arrival of a car at the entrance and exit, respectively. The signal of these sensors have a positive edge trigger when a car is detected and a negative edge trigger when the detection ceases.

The models of the park entrance and park exit are implemented, respectively, in the module *Entrance* (Fig. 3), and in the module *Exit* (Fig. 4). Each of these modules have a declaration of two interfaces.

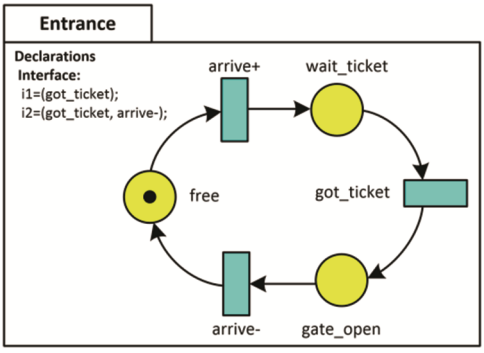

Fig. 3. Parking lot Entrance model.

A module may have multiple interfaces declared, supporting its use in different situations. For each instance of a module the designer chooses the most suited interface to communicate with the surrounding elements of the net.

In this example the interface i_1 of the module *Entrance* (composed by the node *got_ticket*) is used to decrease de number of available places and increase the occupied ones. The interface i_2 (nodes *got_ticket* and *arrive-*) could be used to, additionally, control a traffic light, using the transition *got_ticket* to switch to green and *arrive-* to switch to red.

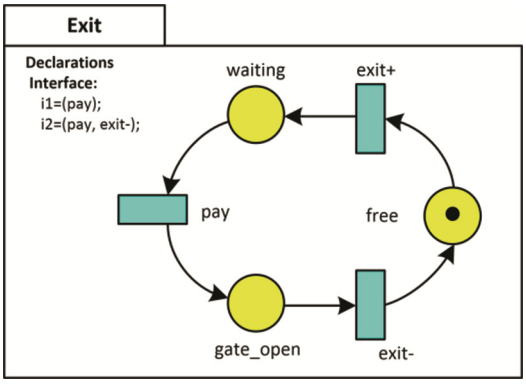

Fig. 4. Parking lot Exit model.

Figure 5 is the net composed by the instances of the modules. The places *occupied* and *free* models the number of places occupied/free in the parking lot. The equivalent plain net is shown in Fig. 6.

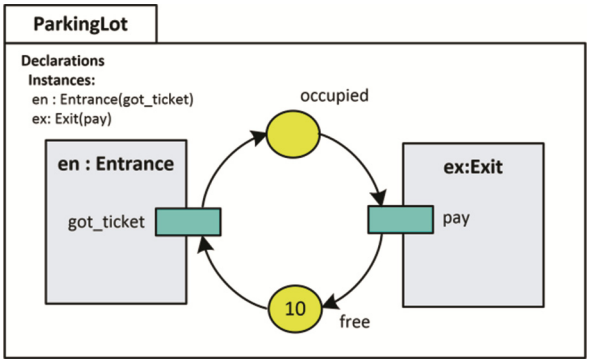

Fig. 5. Parking lot model with modules.

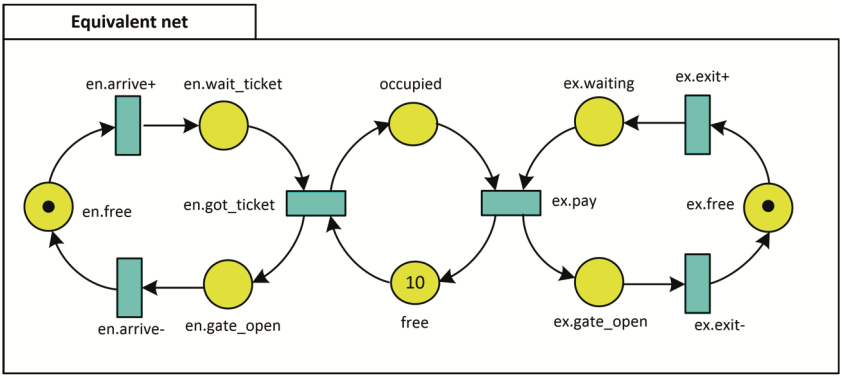

Fig. 6. Parking lot equivalent net.

6 Discussion and Conclusions

It was presented a modular extension for IOPT nets, a Petri net based language targeted for the development of embedded systems. The aim of the work was to create a structuring mechanism to represent models in a more compact way, while ensuring a greater distinction among different components of a system, whether they are physical or logical.

Comparing with similar proposals, as the substitution transition in HCP nets [16] and modular PNML [18], the mechanism presented here distinguishes by supporting multiple interfaces composed by concrete nodes (interface nodes in [18] are reference nodes). Defining multiple interfaces for the same module, supports different ways to connect the module within the container net and therefore a more flexible usage.

The parameterization of some module elements as instance elements (events, signals and constants) also increase the flexibility, enabling that different instances of the same module have its one parameters.

Although it is used the fusion of nodes as the composition mechanism between the surrounding net and an instance of the module, this is done in a transparent manner for the designer. The connections are made through arcs, or any kind of channels, as synchronous or asynchronous, which facilitates the composition.

References

1. Edwards, S., Lavagno, L., Lee, E.A., Sangiovanni-Vincentelli, A.: Design of embedded systems: formal models, validation, and synthesis. Proc. IEEE **85**(3), 366–389 (1997)
2. Murata, T.: Petri nets: properties, analysis and applications. Proc. IEEE **77**, 541–580 (1989)
3. Moalla, M., Pulou, J., Sifakis, J.: Synchronized Petri nets: a model for the description of non-autonomous sytems. In: Winkowski, J. (ed.) Internet – Technical Development and Applications. LNCS, vol. 64, pp. 374–384. Springer, Heidelberg (1978)
4. Moutinho, F., Gomes, L.: Asynchronous-Channels within Petri net-based GALS distributed embedded systems modeling. IEEE Trans. Ind. Informatics **10**(4), 2024–2033 (2014)
5. Girault, C., Valk, R.: Petri Nets for Systems Engineering. Springer, Heidelberg (2001)
6. Huber, P., Jensen, K., Shapiro, R.M.: Hierarchies in coloured Petri nets. Adv. Petri Nets **1990**, 313–341 (1989)
7. 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: ICMC 2014 International Conference on Mechatronics and Control, Jinzhou, China, pp. 1099–1104, July 2014
8. Gomes, L., Barros, J.P., Costa, A., Nunes, R.: The input-output place-transition Petri net class and associated tools. In: 2007 5th IEEE International Conference on Industrial Informatics, pp. 509–514, July 2007
9. Pereira, F., Moutinho, F., Ribeiro, J., Gomes, L.: Web based IOPT Petri net editor with an extensible plugin architecture to support generic net operations. In: IECON 2012 - 38th Annual Conference on IEEE Industrial Electronics Society, pp. 6151–6156, October 2012
10. National Science Foundation (NSF), Cyber-Physical System (CPS) (2011). http://www.nsf.gov/pubs/2011/nsf11516/nsf11516.htm
11. Gomes, L., Barros, J.P.: Models of computation for embedded systems. In: Zurawski, R. (Editor-in-Chief) The Industrial Information Technology Handbook, (Section VI – Real Time Embedded Systems; Chapter 83), pp. 83:1–83:17. CRC Press, Boca Raton (2005)
12. Thomas, B.H., McLean, C.: Using Grafcet to design generic controllers. In: International Conference on Computer Integrated Manufacturing, 1988, pp. 110–119, 23–25 May 1988
13. Hasegawa, K., Takahashi, K., Masuda, R., Ohno, H.: Proposal of mark flow graph for discrete system control. Trans. Soc. Instrum. Control Eng. **20**(2), 122–129 (1984)
14. Zuberek, W.M., Bluemke, I.: Hierarchies of place/transition refinements in Petri nets. In: 1996 IEEE Conference on Emerging Technologies and Factory Automation, 1996, EFTA 1996, Proceedings, vol. 1, pp. 355–360 (1996)
15. He., X.: A formal definition of hierarchical predicate transition nets. In: Proceedings of the 17th International Conference on Application and Theory of Petri Nets, pp. 212–229 (1996)
16. Huber, P., Jensen, K., Shapiro, R.M.: Hierarchies in coloured Petri nets. Adv. Petri Nets **1990**, 313–341 (1989)

17. Kindler, E., Petrucci, L.: Towards a standard for modular Petri nets: a formalisation. In: Franceschinis, G., Wolf, K. (eds.) PETRI NETS 2009. LNCS, vol. 5606, pp. 43–62. Springer, Heidelberg (2009)
18. Kindler, E., Weber, M.: A universal module concept for Petri nets-an implementation-oriented approach. Informatik-Bericht 150, Humboldt-Universität zu Berlin, Institut für Informatik (2001)
19. Juhás, G., Lorenz, R., Neumair, C.: Modelling and control with modules of signal nets. In: Desel, J., Reisig, W., Rozenberg, G. (eds.) Lectures on Concurrency and Petri Nets. LNCS, vol. 3098, pp. 585–625. Springer, Heidelberg (2004)
20. Rausch, M., Hanisch, H.M.: Net condition/event systems with multiple condition outputs, vol. 1, pp. 592–600 (1995)
21. Lakos, C.: Object oriented modelling with object Petri nets. In: Agha, G.A., De Cindio, F., Rozenberg, G. (eds.) Concurrent OOP and PN. LNCS, vol. 2001, pp. 1–37. Springer, Heidelberg (2001)
22. Kummer, O.: Referenznetze. Logos-Verlag, Berlin (2002)
23. Gomes, L., Barros, J.P.: Structuring and composability issues in Petri nets modeling. IEEE Trans. Ind. Inform. **1**(2), 112–123 (2005)