Mapping Between Petri Nets and DEVS Models

Sofiane Boukelkoul, Mohammed Redjimi
Université du 20 Août 1955 – 21000 Skikda
Faculté des sciences- Département d'informatique
Skikda, Algéria
{bouk.sofiane, redjimimed}@yahoo.fr

ABSTRACT: *Complex systems are characterized not only by the diversity of their components, but also by the interconnections and interactions between them. For modeling such systems, we often need more than one formalism and we must concern ourselves with the coexistence of heterogeneous models.This purpose can be achieved by using the multi-modeling. The transformation of such models in a pivot model is a technique in this context. This paper proposes a translation approach of Petri nets to DEVS "Discrete Event System Specification" models.We present mechanisms which can systematically transform the places and transitions of Petri netsto DEVS models. The coupling of these models generates a DEVS coupled model capable of running on platforms based on DEVS.*

Keywords: DEVS, Petri Nets, Multi-modeling, Coupling Model, Modeling and Simulation

Received: 1 December 2012, Revised 10 January 2013, Accepted 13 January 2013

1. Introduction

The diversity and the complexity of increasingly growing systems has forced the scientific community to implement tools for modeling and simulation more and more efficient and meet the expressed requirements and constraints and supports the heterogeneity and especially coupling systems in various disciplines. Now, it appears essential to use federative tools which offer extensive possibilities of abstraction and formalization. The multi-modeling consists of using several formalisms when one wants to model complex systems whose components are heterogeneous.The assembly models for these subsystems with the consideration of the interconnections and interactions between them consist in a set of coupled models representing the initial complex system [1].

Therefore, it has become essential to implement tools and coupling mechanisms to allow multiple models based on different formalisms to coexist.

Our work relates to the implementation of algorithmic tools that allow the transformation of formal Petri nets (PN) models in equivalent models based on the DEVS formalism. So these models can be simulated in DEVS environments.

This paper is structured as follow: First, we quote related works and justifying our motivations, followed by an introduction to multi-modeling. Then we formally present DEVS and PN formalisms. Then we proceed to the presentation of the mechanism of transformation from PN to DEVS with justifications of motivations and choices. Finally, we end with a conclusion and perspectives.

2. Related Works and Motivations

Several researches have focused on the study of the relationship between PN or other dynamic formalism and DEVS formalisms, since DEVS is considered as one of the basic modeling formalisms based on the unifying framework of general dynamic modeling formalism. Jean de Lara proposed in [4] a modeling based multi-paradigm to generate PN and State-Charts. It consists of modeling at multiple levels of abstraction implemented in AToM3 "*Tool for Multi-formalism and Meta-Modeling*". Where is presented a graphical abstraction of meta-models of Satcharts and PNs. For example the use of CD++ to develop PN [3] is closest to our work. However it only provides tools for generating PN by using library of predefining models for PN places and transitions. Therefore one may be couldn't find the appropriate model for given transition especially when it contains a big number of ports. Furthermore, in [3] we don't find a vital parallelism because firing transitions is scheduled. That means one never find more than one transition in firing state, while the parallelism is one of the fundamental PN characteristics. Thus the conflict characteristic of PNs is silently absent since without parallelism the problematic of conflict is not considered. So the value of our work is that is characterized by the development of algorithms that can automatically transform the existing PN in DEVS models. Moreover, the most important characteristics of PNs such as parallelism, concurrence and conflict are well preserved in our approach.

3. Multi-modeling

Complex systems are so far to be modeled with only one formalism because many aspects are present. And one formalism can't treat all the system frameworks. It can be semantically adequate for one or more aspect of a complex system but unlikely not for all. So multi-modeling comes to benefit of the multitude of formalisms' representation force by allowing coexisting many models based on different formalism in one model. According to Hans Vangheluwe [2], multi modeling paradigm focuses on three axes:

- The coupling and transformation of models described in different formalisms.
- Abstraction of models by defining the relationship between them at each level of abstraction.
- The meta-model which focuses on the description of the classes of models (models of models).

In [14] there is a representation of various possible transformations by using formalism transformation graph "*FTG*".

4. The DEVS Formalism

DEVS was initially introduced by B. P. Zeigler [5] in 1976 for discrete event systems modeling. It is modular, hierarchical, abstract and independent of any implementation. It is characterized by a rigorous formalization of exchange events between its models (atomic or coupled). DEVS offers a formal framework of model coupling. Further more, DEVS is closed under coupling, i.e. a coupled model can be considered as an equivalent atomic model. This characteristic is strongly required for the hierarchical model construction.

A DEVS atomic model is based on a continuous-time inputs, outputs, states and functions (exit, transition, life states). Coupled models are more complex. They are constructed by connecting several atomic models or even coupled hierarchically. Interactions are handled through the ports of entry and exit models, which promotes modularity.

4.1 Formal Specification of a DEVS atomic model

A DEVS atomic model is described by the following equation:

$$Atomic\ DEVS = (X, Y, S, \delta_{int}, \delta_{ext}, \delta_{con}, \lambda, t_a) \tag{1}$$

X is the set of external inputs.

Y is the set of model outputs.

S is the set of states.

$\delta_{int} : S \rightarrow S$: is the internal transition function that moves the system from a state to another autonomously. It depends on the time elapsed in the current state.

$\delta_{ext}: S \times X \rightarrow S$: is the external transition function occurs when model receives an external event. It returns the new state of the system based on the current state.

$\delta_{con}: X \rightarrow S \times S$: is the transition function of conflict. It occurs if an external event happens when an internal change (autonomous) system status. This feature is only present in a variant of DEVS: Parallel DEVS [6].

$\lambda: S \rightarrow Y$: is the output function of the model. It is activated when the elapsed time in a given state is equal to its life.

t_a (s) represents the life of a state "*s*" of the system. It is the time during which the model will remain in this state, if no external event occurs.

DEVS atomic model can be represented graphically as in Figure 1. It is represented by a rectangle with a plurality of inputs ports and a plurality of outputs ports, represented by filled triangles. The value v_i is the value taken by an input or output port. This value belongs to the set of possible values of p_i ports. An input port has a value in the processing of an event attached to that port. An output port has a value when the output function takes a value for this port.

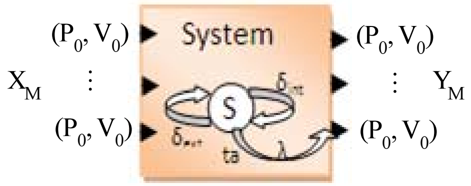

Figure 1. Representation of a DEVS atomic model

4.2 Formal Specification of a DEVS coupled model

Coupled DEVS formalism describes a system as a network of connected components. These components consist of models that are atomic or coupled. These components interact with themselves by exchanging events and values through their ports. A produced event by a model output can be received by input of other connected model. To note that the behavior of DEVS coupled model is the same to atomic one since this formalism is closed under coupling. Thus the main fruit of this characteristic is the ability to be perfectly hierarchical formalism. So coupled model can indeed be treated as atomic ones in large coupled models by encapsulating them.

To illustrate the DEVS coupled model, we propose a simple example whose graph is given in Figure 2 where the coupled model contains inside two interconnected models: *A* and *B*.

$$Coupled\ Devs = (X_{self}, Y_{self}, D, \{M_d / d \in D\}, EIC, EOC, IC) \qquad (2)$$

Self: is the model itself.

X_{self} is the set of inputs of the coupled model.

Y_{self} is the set of outputs of the coupled model.

D is the set of names associated with the components of the model, self is not in *D*.

$\{M_d / d \in D\}$ is the set of components of the coupled model.

EIC, *EOC* and *IC* define the coupling structure in the coupled model.

EIC is the set of external input couplings. They connect the model inputs coupled to those of its own components;

EOC is the external output couplings. They connect the outputs of the components to those of the coupled;

IC defines internal coupling. It connects the outputs of components with inputs from other components in the same coupled model. However, no direct feedback loops are allowed, that means no output port of a component (model) may be connected to an input port of the same component.

5. The Petri Nets (PN)

Petri Nets are a modeling formalism, developed originally by C.A Petri [7]. They are very suitable for dynamic systems modeling. *PN*s are directed graphs with two types of nodes: places and transitions connected together by arcs. An arc should never connect two nodes of the same type. Places represent in most cases the static aspect of the system, while transitions are responsible for the dynamics. Each place can contain *n* tokens ($n \in N$ with N the set of positive integers). Arcs can be labeled with a weight (positive integer), we speak of "*generalized PN* ". If the weight of all arcs is equal to1, then the *PN* is called "*ordinary*".

A transition is said to be valid if all the places up streameach contain a number of tokens ≥ the weight of the arc which connects to the transition. A valid transition can be fired. Crossing or firing a transition results in two simultaneous actions: the first is the destruction of tokens in places located up stream of the arc connecting the transition to those places as match as the weight of that arc. The second generates tokens in the places located down stream of the transition as match as the weight of the arc from this transition instead.

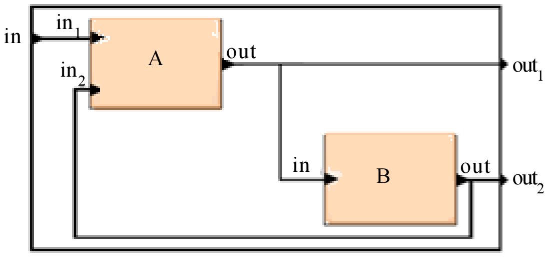

Figure 2. Graphical representation of a coupled model consisting of two models

5.1 Specifying formal Petri Nets

Several formal definitions can specify the *PN*. We provide the most appropriate to our work, which defines formally a *NP* as 5-uple:

$$PN = (P, T, PRE, POST, Mo) \tag{3}$$

P: is the set of places, *T*: is the set of transitions, PRE: the matrix generated by applying $P \times T \rightarrow N$, $PRE\,[i, j] = n\,/\,n = 0$ if the place is not upstream of the transition t_j else $n = \tau\,/\,\tau$ is the weight of the arc from p_i to t_j, *POST*: the matrix generated by applying $T \times P \rightarrow N$, $POST\,[i, j] = n\,/\,n = 0$ if the place p_i is not downstream of the transition t_j else $n = \tau\,/\,\tau$ is the weight of the arc from t_j to p_i and M_0: is the vector of initial marking; $M\,[i] = k\,/\,k$ is the number of tokens in place p_i.

Figure 3, shows a *PN* which consists of three places and one transition modeling action (*T*1) having two conditions (*P*1, *P*2) to be run. The result is put in place (*P*3). The weight of the arcs is equal to 1, it is not, therefore, necessary to label them.

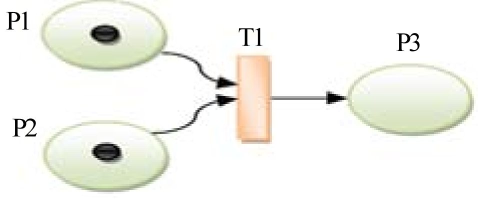

Figure 3. Example of Petri Net

6. Transformation of PN to DEVS models

The multitude of modeling tools and formalisms and the desire to reuse existing models have stimulated researchers to guide their works toward a goal of standardization of these tools. The DEVS formalism seems to be well suited for this mission.

6.1 Why DEVS?

The strength of DEVS is summed up in its ability to express, thanks to the concept of abstraction applied to each level, ranging from atomic models to a collaboration of a set of models where each interact with others. Although in dependent of the implementation, DEVS provides a modular and hierarchical vision of dynamic models. Events generated by a model can take in various values fields and can best imuli for other models. Thus, according to B. P. Zeigler [6], we can prove that there is a model DEVS for all discrete event systems. But we can go further. In fact, DEVS can be "*universal*" [8], allowing the coupling of models and formalisms described as heterogeneous paradigms. The main idea is that the models are considered as black boxes that have no link with the outside world except through input and output ports to exchange events and values. With this feature abstraction, several models can be coupled while enjoying the reuse of existing models. It is also possible to perform formal verification of DEVS models, which is a valuable aid in the design of systems [9].

In addition, several DEVS-based platforms are available as VLE "*Virtual Laboratory Environment* " [10], DEVS JAVA [11] developed in Java, Cell-DEVS "*Cellular DEVS*" which is based on the formalism of cellular automata [12].

6.2 Coupling models

Coupling models based on DEVS is a typical task. However, non-DEVS models require extra effort to be coupled. Two methods exist to incorporate a non-DEVS model in the DEVS: co-simulation and transformation [13]. The transformation of non-DEVS models (PNs in our case) to the DEVS, comes in the way to specify models in uniform language. Vangheluwe [14] represents the various possible transformations by using formalism transformation graph "*FTG*".

In the case of co-simulation: which is standardized are the communications between simulators and not the model specifications. There are several works such as HLA (High Level Architecture) [15] which fit into this framework.

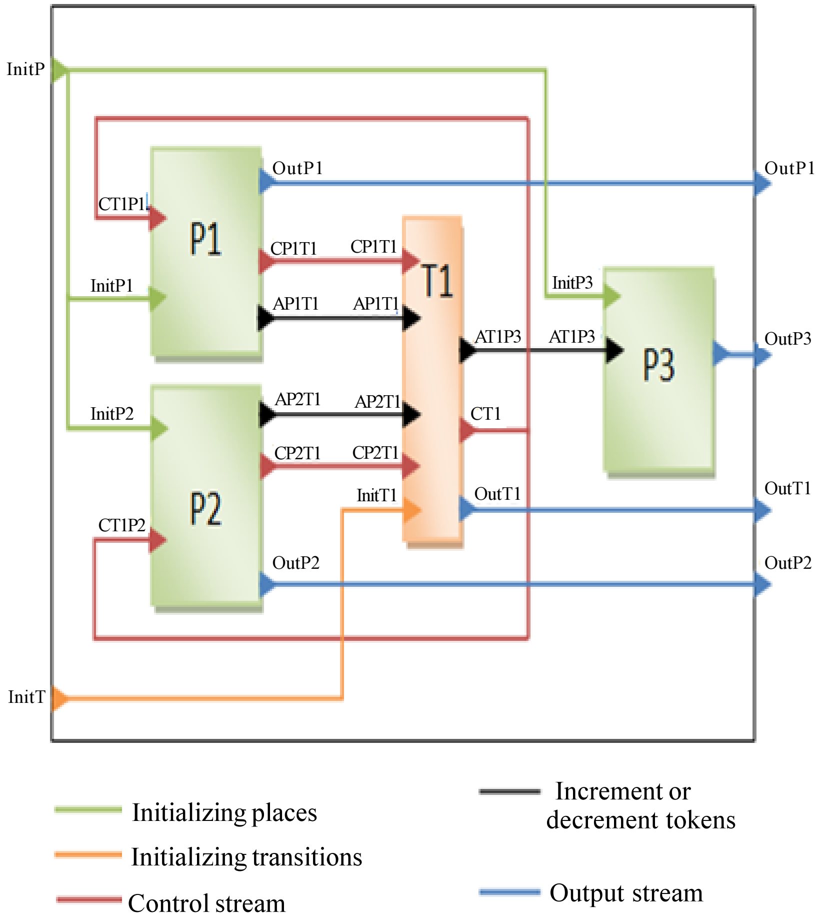

Figure 1. DEVS coupled models corresponding to the previous Petri net

6.3 Mechanisms of PN transformation to DEVS

The idea of our approach is to have as a result a DEVS coupled model (CDEVS) faithful to the input *PN*. To this aim, we propose an algorithm that automatically generates CDEVS specifying its components and their inter connections.

6.3.1 Structure of resulting DEVS model

The coupled model is composed from several DEVS atomic models with many places and transitions according to the *PN* source. Figure 4, illustrates the CDEVS model corresponding to the *PN* shown in Figure 3. DEVS model corresponding to the "*transition*" of *PN* (TDEVS for "*Transition DEVS*") is characterized by an output port "*control*" (*CT*1 in the example) whose role is to send events to places upstream, verify the number of tokens or inform them about its firing. However, TDEVS receives events from the models corresponding to places upstream (to be PDEVS "*DEVS Place*" in the following) with control ports as much as number of places (*CPiT*1). Control ports are illustrated in Figure 4, in color red.

The evolution of model CDEVS begins with an external event "*initialize*" received by the input port *InitT* (in orange) and will be broadcast on all TDEVS, as well as pause, resume and stop events that are received through the same port. Note that this port is coupled only by TDEVS because the elements responsible for the dynamics in *PN* are the transitions and not the places.

The method used in this paper is the transformation taking the *PN* formalism as source, while DEVS is the target.

TDEVS is not linked by its downstream CDEVS except by output port for each *AT*1*Pi* (in black). Via these ports transition (TDEVS) informs their places downstream about its crossing.

All TDEVS and PDEVS are provided with an output port *OutTi* and *OutPi* respectively (in blue). These ports are coupled directly with the output ports for possible CDEVS records or statistics.

All PDEVS have an input port each (*InitPi*), by which they are coupled with CDEVS via an input port *InitP* (green). This liaison role is to initialize the marking of places. Arcs from places *Pi* and arriving at a transition *Tj* are translated into output ports (*APiTj*) at PDEVS and input ports (*APiTj*) at TDEVS corresponding to *T* (black).

The creation of the structure of DEVS model corresponding to the *PN* is performed by the algorithm "*Main_PN_DEVS*" below. This algorithm takes as input a $PN = (P, T, PRE, POST, M_0)$, resulting in an output DEVS model. By the matrix PRE, the algorithm creates links corresponding to the arcs that link places by upstream transitions. The matrix *POST* is used for the coupling between TDEVS (transitions) and PDEVS (places) downstream of the transition.

```
Algorithme : Main_PN_DEVS
Input PN= (P, T, PRE, POST, M0)
Output CDEVS //coupled model
Begin :
Create CDEVS as coupled DEVS model //void model
For all transition i do create TDEVSi as atomic DEVS model  end for
for all places j do create PDEVSj as atomic DEVS model end for
for all PDEVSj do
    add 'InitPj' as intput port and join it to CDEVS.IN.InitP
//starting tokens
    add 'OutPj' as output port and join it to CDEVS.OUT.OutPj
//output stream
end for
for all TDEVSi do
     add 'InitTi' as input port //initialize, stop, pause, release
    join 'InitTi' port to CDEVS.IN. InitT port
//coupling
    add 'OutTi' as output port and join it to CDEVS.OUT.OutTi
//output stream
     add 'CTi' as output port // control: check, reserve, decrement, cancel
```

```
for all PDEVSj do
      if (PRE[i,j] > 0) //upstream place
          add to PDEVSj 'CTiPj' as input port //check, reserve, decrement, cancel
         join TDEVSi.OUT.CTi to PDEVSj.IN.CTiPj // coupling
          add to PDEVSj 'CPjTi' as output port //ok, busy ,number_of_free_tokens
add to TDEVSi 'CPjTi' as input port //ok, busy ,number_of_free_tokens
         join PDEVSj.OUT.CPjTi to TDEVSi.IN. CPjTi // coupling
         add to PDEVSj 'APjTi' as output port //arc: value = PRE[i, j]
         add to TDEVSi 'APjTi' as intput port //arc: value = PRE[i, j]
        join PDEVSj.OUT. APjTi to TDEVSi.IN.APjTi // coupling
      end if
      if (POST [i, j] > 0) //downstream places
         add to TDEVSi 'ATiPj' as output port //arc: value = POST [i, j]
         add to PDEVSj 'ATiPj' as input port //arc: value = POST [i, j]
        join TDEVSi.OUT.ATiPj to PDEVSj.IN.ATiPj // coupling
      end if
   end for
end for
end  Main_PN_DEVS
```

6.3.2 Dynamic of resulting DEVS model

In this transformation approach we model the dynamic DEVS coupled generated by the previous algorithm in accordance with the functions offered by the DEVS formalism which are δ_{int}, δ_{ext}, δ_{con} and λ. After initialization of places (PDEVS) by the initial marking and after launching the evolution of the model by the event "*initialize*" received by all transitions (TDEVS). The latter are in state "*checking*" (thanks to δ_{ext} function) to see if the number of tokens in places upstream is sufficient to achieve a crossing. Event "*check*" is sent (by the function λ). PDEVS concerned receive the event, they transmit the number of their free tokens (which are not reserved by other transition) with λ function as well. If the number of tokens is sufficient to validate the transition (TDEVS), then it will change the status of "*checking*" to make it "*reserving*" and sends the event "*reserve*" with the function λ always.Crossing does not occur directly. It must go through a reservation tokens to avoid the conflict problem (if places are upstream of several transitions), as long as the transitions are in continuous competition. In this way the property of PN in terms of dynamics and competition is faithfully preserved in our transformation approach.

PDEVS receiving the event "*reserve*" return "*ok*" if there is still enough tokens (> = the weight of the arc), "*fail*" otherwise. If TDEVS receives at least one "*fail*" it returns immediately to all PDEVS signal "*cancel*" to release the reserved tokens probably, it puts his state "*Validated* " otherwise. At this point, the transition can pass the crossing and therefore returns "*decrement*" to PDEVS which will destroy the tokens reserved by TDEVS in question. TDEVS simultaneously sends "*increment*" to PDEVS located downstream to increment the number of tokens with the value received by the input port (weight of the arc). After firing a TDEVS it rehabilitates "*checking*" and so on.

Functions δ_{int}, δ_{ext}, δ_{con} and λ, characterizing the models TDEVS are summarized in Table1. The first two columns represent the inputs, which are the events and the current state. The other columns show the outputs of each function. The table rows are grouped separately for each current state. And models PDEVS; functions are shown in Table 2. By convention, if all events have the same impact, we write "*all events*". Empty cells indicate the absence of values: for λ is the absence of events. For δ_{ext}, and δ_{int} δ_{con} meaning that the function does not produce an output state, so the state of the model is basically the same. The "&" symbol indicates that the events are simultaneous.

The evolution of CDEVS is triggered by the event "*initialize*" overview by TDEVS. So we can pause or stop the simulation by sending events "*pause*" and "*stop*" to TDEVS. Therefore, the model is in one of the following states: "*paused* " or "*stopped* ". In these states transitions (TDEVS) will ignore all events of PDEVS comers. Only events "*initialize*" or "*release*"can trigger the simulation again.

Event	Current state	δ_{ext}	δ_{int}	δ_{con}	λ (current state)
initialize	all states	checking			out
pause		paused			out
stop		stopped			out
release		checking			out
free_tokens	reserving	reserving		Reserving	reserve
Ok		validated, reserving		validated, reserving	
fail		canceling		Canceling	
all events	validated	checking			decrement & increment & out
all events	canceling	checking			cancel

Table 1.The outputs of theTDEVS model functions

Event	Current state	δ_{ext}	δ_{int}	δ_{con}	λ (current state)
initialize	all states	Checking		Checking	out
check	checking	Checking	checking	Checking	free_ tokens
reserve		Reserving		Reserving	
increment		Incrementing		incrementing	
decrement		decrementing		decrementing	
cancel		Checking		Checking	
check	reserving	Reserving	checking	Reserving	ok, fail
reserve		Reserving		Reserving	
increment		Incrementing		incrementing	
decrement		decrementing		decrementing	
cancel		Checking		Checking	
check	incrementing	Checking	checking	Checking	out
reserve		Reserving		Reserving	
increment		Incrementing		incrementing	
decrement		decrementing		decrementing	
cancel		Incrementing		incrementing	
check	decrementing	Checking	checking	Checking	out
reserve		Reserving		Reserving	
increment		Incrementing		incrementing	
decrement		decrementing		decrementing	
cancel		decrementing		decrementing	

Table 2. The outputs of the PDEVS model functions

Figure 5 illustrates the elementary transformations of PN components to their equivalent objects in DEVS. Where (a) represents a single place with the minimum of ports it has to possess. (b) illustrates a single given transition. (c) and (d) represents the minimum of IC between a place and a transition. (e) is graphical representation of IC in case of conflict between two transitions. Finally, (f) represents the IC of typical transformation where the parallelism has place.

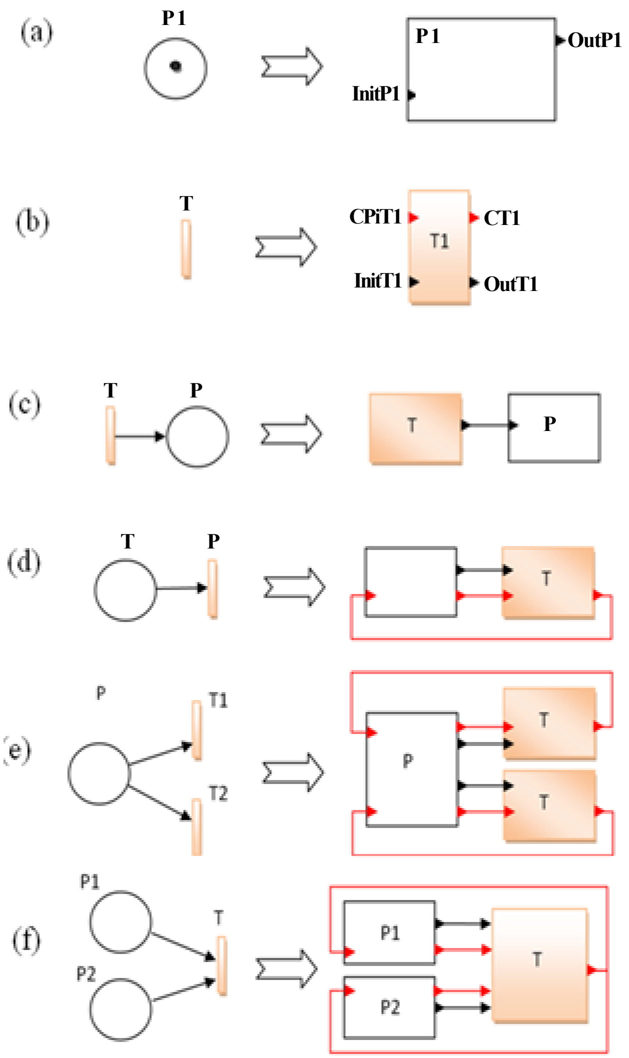

Figure 5.Graphical representation of elementary transformations and IC between generated DEVS models

Formally, the transformation is presented as follow:

$PN \rightarrow CDEVS$

$(P, T, PRE, POST, Mo) \rightarrow (X, Y, D, EIC, EOC, IC)$

Where:

$D = \{P \cup T\}$

$X = \{InitP, InitT\}$

$Y = \{OutDi \ / \ Di \text{ is atomic model representing } Pi \text{ or } Ti\}$

$EIC = \{\{(CDEVS.InitP, PDEVS.IntPi)\} \cup \{(CDEVS.initT, TDEVS.IntTj)\}$

$/ \ i \in N + \ \& \ i <= \text{Number of places, } j \in N + \& \ j <= \text{Number of transitions } \}$

$EOC = \{(Pi.OutPi, CM.OutPi), (Tj.OutTj, CM.OutTj) \ / i \in N + \ \&$

$i <= \text{Number of places, } j \ C \ N + \ \& \ j <= \text{ Number of transitions } \}$

$IC = \{$

$\{(Pi.APiTj, Tj.APiTj) \ / \ PRE\,[i,j] > 0 \}$

$\cup \{(Tj.ATjPi, Pi.ATjPi) \ / \ POST\,[i,j] > 0 \}$

$\cup \{Tj.CTj\} \ X \ \{Pi.CTjPi\} \ / \ PRE[i,j] > 0 \}$

$\cup \{(Pi.CPiTj\ ,\ Tj.CPiTj\}\ /\ PRE[i, j] > 0\ \}$

$/i \in N + \&I <= Number\ of\ places,\ j \in N + \&\ j <= Number\ of\ transitions$

}

6.3.3 Example of transformation

In this section, we present an example of transformation of one of famous case study in *PN* training field: Producer-Consumer with limited plug (7 puts) illustrated in Figure 6.

*P*1: Producer is ready to produce. *T*1: Begin of production. *P*2: Production is run.*T*2: End of production.*P*3: plug containing products (initially, plug is empty). *P*4: Consumer is ready to consume.*T*3: Begin of consummation. *P*5: Consummation is run. *T*4: End of consummation. *P*6: Number of free puts (initially, all puts in plug are free).

We prefer to illustrate the transformation graphically since both of the formalisms *PN* and DEVS offer graphical representation of their models. Figure 7, represents the coupled model faithful to the *PN* modeling Producer-Consumer. In this graphical we conserve the same color signification as shown in Figure 4.

- **Color red:** to illustrate control stream.
- **Color black:** to illustrate tokens incrementing or decrementing.
- Color green to initialize places' tokens number.
- Color orange to initialize transitions.
- Color blue for outputs.

$$PC\_PN = (P, T, PRE, POST, M_0), \mathrm{P} = \{P1, P2, P3, P4, P5, P6\},\ T = \{T1, T2, T3, T4\}$$

$$PRE = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} POST = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix} M_0 = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 7 \end{pmatrix}$$

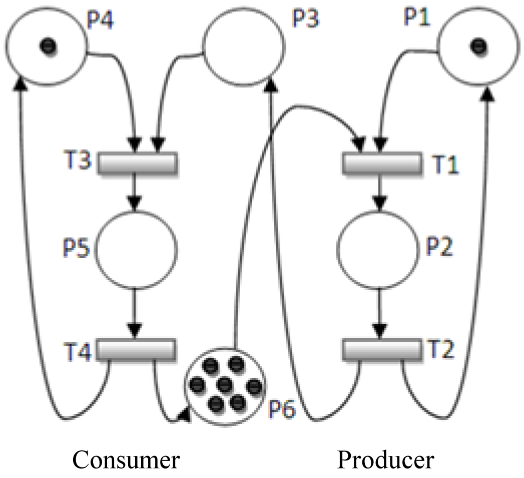

Figure 6. Producer-Consumer *PN*

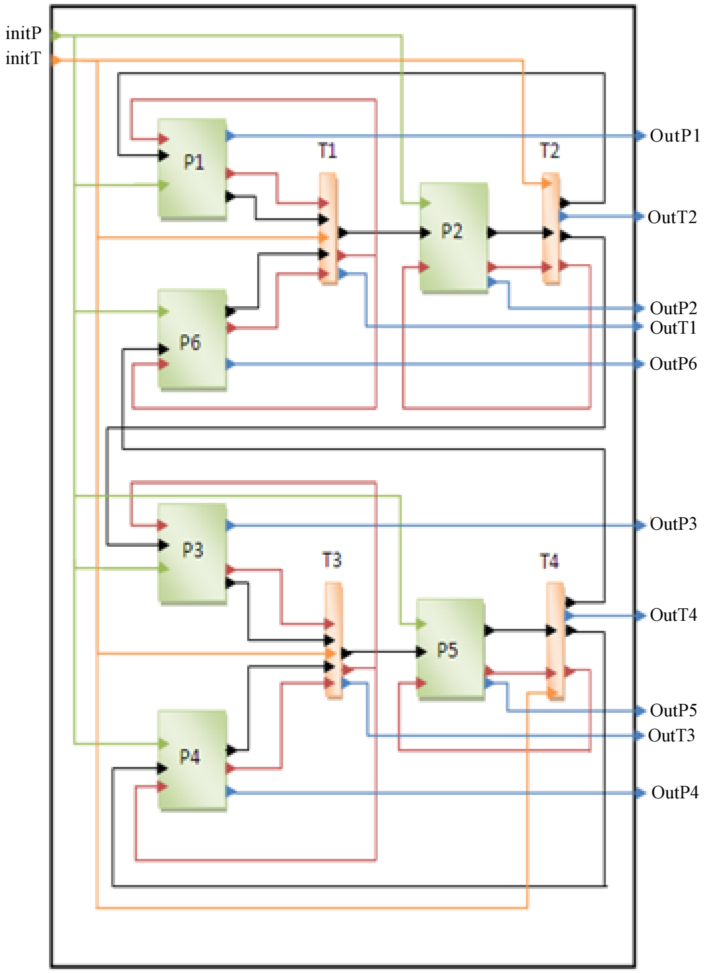

Figure 7. DEVS coupled models corresponding to Producer-Consumer *PN*

7. Discussion

Petri nets are formal tools modeling dynamic systems dealing perfectly with the aspect of competition and parallelism between processes. Therefore, they require gentle handling during mapping in order to not lose their specifications. In our approach, competition is preserved by the creation of temporary state transitions which is the reserving state. Thus, a token cannot participate at the same time, in the crossing of two transitions in conflict. However, the transition must immediately release tokens if it has reserved them and fails to be validated. In order not to paralyze other transitions which are in conflict with it.

In this paper we discussed the generalized *PN* for the reader to understand the mechanism of transformation. However, other extensions such as colored *PN* can also be processed. In this case, the tokens will no longer be trivialized. We will need to extend the type of representation of these to comprise a list with all colors. Thus, during the broadcast of the event "*check*" with a transition. Places of upstream should check the port connecting to the transition to send only the number of free tokens with the same color as specified at this port. With the same principle, the destruction of chips will be following a transition firing.

In addition, the DEVS formalism provides flexibility in the internal structure of the models [16]. Models may disappear, others can take over. Interconnections can have birth, others may change the models they couple. This aspect of dynamic structure related to DEVS models motivates modelers to simulate their *PN* models in DEVS platforms. The complexity of PN related to the representation of structural changes in systems will be simplified, as long as a DEVS model can have multiple structures, therefore, represent several *PN* at a time.

8. Conclusion and Perspectives

In this paper we have presented a transformation approach of Petri nets to DEVS models. By an algorithm that systematically built atomic DEVS models for each of the places and transitions which will be coupled to obtain a coupled DEVS model by specifying the interconnections between these models. This work falls within the framework of transformation models based multi-formalisms in a uniform one, DEVS is in our case. Our choice of this formalism was based on DEVS force in unifying and coupling models. Characterized by its abstraction, implementations independence and its ability to model complex systems in the form of a hierarchical model, DEVS is a formalism that can be the unifier of models of discrete event systems.

By the transformation presented in this paper, the *PN* can enjoy the simulation offered by multiple platforms based on DEVS. Thus the question of their validation will reply via these tools. Indeed, DEVS provides the ability to perform formal verifications of models.

Our perspectives focus on the implementation of such algorithms on complex models such as industrial processes where the notion of parallelism, interaction and distribution calculations should be considered.We are also considering the implementation of a platform for testing and validation of Petri nets while taking advantage of open source tools based on DEVS.

References

[1] Fishwick, P. A. (1995). Simulation Model Design and Execution. Prentice Hall.

[2] Vangheluwe, H. (2008). Foundations of modelling and simulation of complex systems. Electronic Communications of the EASST, 10: Graph Transformation and Visual Modeling Techniques. http://eceasst.cs.tu-berlin.de/index.php/eceasst/issue/view/19.

[3] Jacques, C. J. D. et Wainer, G. A. (2002). Using the CD++ DEVS Tookit to Develop Petri Nets. *In*: Proceedings of the SCS Summer Computer Simulation Conference, San Diego, CA. U.S.A.

[4] Lara J. et Vangheluwe, H. (2002). Computer Aided Multi-Paradigm Modelling to Process Petri-Nets and Statecharts. 2505, 239-253

[5] Zeigler, B. P. (1976). Theory of Modeling and Simulation, Wiley InterScience.

[6] Zeigler, B. P., Praehofer, H. et Kim, T. G. (2000). Theory of Modeling and Simulation, Second edition. Academic Press, ISBN 0127784551.

[7] Peterson, J. L. (1977). Petri nets, Computing Surveys, 223–252.

[8] Touraille, L., Traoré, M. K. et Hill, D. R. C. (2010). SimStudio: une Infrastructure pour la modélisation, la simulation et l’analyse de systèmes dynamiques complexes. Research Report LIMOS/RR-10-13.

[9] Freigassner, R., Praehofer H., et Zeigler, B. P. (2000). Systems approach to validation of simulation models. Cybernetics and Systems, 52–57.

[10] Quesnel, G. (2006). Approche formelle et opérationnelle de la multi-modélisation et de la simulation des systèmes complexes. Laboratoire d’Informatique du Littoral.

[11] Sarjoughian, H. et Zeigler, B. P. (1998). Devsjava : Basis for a DEVS-based collaborative ms environment. SCS International Conference on Web-Based Modeling and Simulation, San Diego, CA, 5, 29-36.

[12] Ilachinski, A. (2001). Cellular Automata, A Discrete Universe, World Scientific Publishing Co, ISBN. 981-02-4623-4.

[13] Schmidt, D. C. (2006). Model-Driven Engineering Guest Editor’s Introduction IEEE Computer, 39 (2) 25-31.

[14] Vangheluwe, H. (2000). DEVS as a common denominator for multi-formalism hybrid systems modeling Conférence IEEE International Symposium on Computer-Aided Control System Design, Alaska, p.129-134.

[15] IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)- Framework and Rules, *Institute of Electrical and Electronics Engineers*, *IEEE* 1516-2000.

[16] Baati, L. (2007). Approche de modélisation DEVS à structure hiérarchique et dynamique. LSIS UMR-CNRS 6168, Domaine Universitaire de St Jérôme.