Using the CD++ DEVS Tookit to Develop Petri Nets

Christian J.D. Jacques
Gabriel A. Wainer
Department of Systems and Computer Engineering
Carleton University
1125 Colonel By Drive
Ottawa, Ontario, K1S 5B6, Canada

Keywords: DEVS, Petri Nets, CD++ tool

Abstract
We describe an implementation where mapping of the Petri Net (PN) modeling formalism into the DEVS modeling formalism was used for the purpose of simulating PNs using a DEVS simulator. The focal point of the paper is a description of the DEVS atomic models used to model PN transitions and places along with a description of PN simulation results obtained using an unmodified DEVS simulator.

INTRODUCTION

The DEVS Formalism

The DEVS formalism [5] was originally defined as a discrete-event modeling specification mechanism. It is a systems theoretical approach that allows the definitions of hierarchical modular models that can be easily reused [Zeigler et al., 2000]. A real system modeled with DEVS is described as a composite of submodels, each of them being behavioral (atomic) or structural (coupled).

A DEVS atomic model is formally described by:

$$M = < X, S, Y, \delta int, \delta ext, \lambda, D >$$

where X is the input events set; S is the state set; Y is the output events set; int is the internal transition function; ext is the external transition function; is the output function; and D is the duration function.

A DEVS coupled model is composed of several atomic or coupled submodels. They are formally defined as:

$$CM = < X, Y, D, \{Mi\}, \{Ii\}, \{Zij\} >$$

where X is the set of input events; Y is the set of output events; D is an index for the components of the coupled model, and $\forall$ i $\in$ D, Mi is a basic DEVS (that is, an atomic or coupled model), Ii is the set of influencees of model i (that is, the models that can be influenced by outputs of model i), and $\forall$ j $\in$ Ii, Zij is the i to j translation function. We can see that coupled models are defined as a set of basic components (atomic or coupled) interconnected through the model's interfaces. The translation function is in charge of converting the outputs of a model into inputs for the others. To do so, an index of influencees is created for each model (Ii). This index defines that the outputs of the model Mi are connected to inputs in the model Mj, where j is an element of Ii.

DEVS hierarchical constructions enable multi-formalism modeling (that is, the coupling of and transformation between models described in different formalisms). Using different formalisms to represent such systems enables a modeler to chose the formalism that lends itself best to represent each sub-systems.

The CD++ Tool

CD++ [3, 4] is a tool that allows a user to implement DEVS models. The tool is built as a hierarchy of models, each of them related with a simulation entity. Atomic models can be programmed and incorporated onto a basic class hierarchy programmed in C++. A specification language allows defining the model's coupling, including the initial values and external events. The tool also enables a user to build models using graph-based notations [5], which allows visualization of the problem in a more abstract way. Therefore, we have used an extended graphical notation to allow the user define atomic models behavior. Each graph defines the state changes according to internal and external transition functions, and each is translated into an analytical definition. Our long term goal is to provide the user with a set of libraries to develop complex models based on multiformalisms. We have already included Finite State Automata, Petri Nets, DEVS graphs and DEVS atomic models written in C++, enabling the users to use different formalisms to describe different properties of a system. In this work we will focus in the implementation of Petri Nets as an example of multi-formalism model definition.

Petri Nets

Petri Nets (PNs) are a modeling formalism originally developed by C.A Petri [1]. They are especially well suited to model systems where concurrent events can take place. One of their most appealing aspect is the simplicity of their graphical representation. Static properties of PNs are can be described using three elements illustrated in Figure 1: places (large circles), transitions (bars) and arcs joining places to transitions and vice-versa.

Petri Nets have the following semantics for static definitions:

- A place may have zero or more inputs. For example, P1, P4 and P2 have zero, one and two inputs respectively.

- A place may have zero or more outputs. For example, P4, P5 and P3 have zero, one and two outputs respectively.
- A transition may have zero or more inputs. For example, t4, t3 and t2 have zero, one and two inputs respectively. A transition with no inputs is called a *source*.
- A transition may have zero or more outputs. For example, t5, t1, t4 have zero, one and two outputs respectively. A transition with no outputs is called a *sink*.

Tokens (black dots) are used to represent the model's dynamic behavior. The idea is to represent states or physical locations of the system using places and to show entities or resources using tokens inside the places. The movement of tokens from place to place models the movement of entities or resources in the real system. The following rules define the semantics for the model dynamics:

- A place may contain zero or more tokens. For example, P2, P5 and P1 contain zero, one and two tokens respectively.
- A transition is either enabled or disabled. A transition is enabled if all of its input places contain at least one token. A *source* transition is always enabled hence it can be used as an infinite source of tokens.
- A PN is executed by firing enabled transitions, one at a time, for as long as there is at least one enabled transition.
- When more than one transition is enabled at any given time, the one that fires is selected in a non deterministic manner.
- When a transition fires, a token is removed from each one of its input places and a token is deposited in each one of the output places. Source and sink transitions are exceptions to this rule.

Transition firing is instantaneous meaning that tokens are removed from input places and deposited in the output places at exactly the same time.

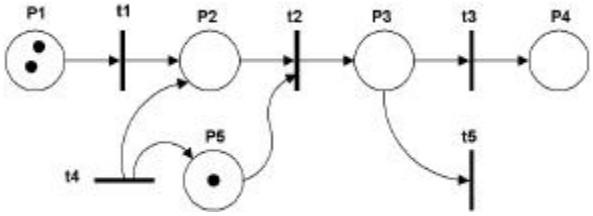

Figure 1. A Typical Petri net

Over the years extensions to PNs have been introduced to increase their modelling capabilities. Two of these extensions include the definition of inhibitor and multiple arcs. An inhibitor arc is an arc which goes from a place to a transition and it enables the transition only if the place is empty as opposed to containing at least one token. A multiple arc is an arc which goes from a place to a transition and vice-versa and indicates the number of tokens being transferred is more than one.

MAPPING PETRI NETS TO DEVS

The construction of a library of atomic models to represent PN transitions and places was straightforward since it was shown in [2] that PNs can be embedded in DEVS because any discrete event behaviour can be expressed as a DEVS model. Modeling and simulating PNs with an unmodified DEVS simulator is a concrete example showing PNs are indeed embedded into DEVS. The remaining sections are devoted to decribe how this can be achieved.

Creating the DEVS equivalent of PNs requires, at the minimum, that the PN characteristics explained previously also exist in the DEVS implementation. Furthermore, for the CD++ tool to be accepted as a useful PN modeling and simulation tool, the DEVS implementation must have the look and feel of PNs so PN modelers are comfortable using it. Furthermore, it must be as simple to use as any PN simulation tool and allow efficient analysis of simulation results. With this in mind, the chosen solution for modeling PNs was to create two DEVS atomic models. One to represent a place and one to represent a transition. This is very versatile because any PN can be constructed by coupling the two types of DEVS atomic models in a manner very similar to how places and transitions are coupled in a PN. Furthermore, it makes it easy for someone to map a PN into the proper .ma file (model definition file) necessary for the CD++ tool to execute the PN. That is, for every place or transition in the PN there is an atomic model and every arc is represented by a link between the atomic models.

The Place Model

Figure 2 illustrates the conceptual description for the DEVS model of a PN place. It has one input port and one output port described below.

Figure 2. Place conceptual model

in port: this input is used to receive tokens from zero or more PN transitions. It is also used to tell the place to loose tokens such as when a transition fires. The encoding of the messages received from this port contains information regarding the number of tokens and the operation (subtraction or addition) to be performed. This is how the model supports multiple arcs.

out port: this output is used by the place to advertise the number of tokens it contains so transitions that are connected to it can determine if they are enabled. This process is executed every time the number of tokens in the place is modified and when the model is initialized at the beginning of simulation.

The formal specification for this model is:

$M = < X, S, Y, \delta_{int}, \delta_{ext}, D, \lambda >$ where:

$X = \{IN \in \boldsymbol{N}^+\}$

$Y = \{OUT \in \boldsymbol{N}^+\}$

$S = \{\{tokens \in \boldsymbol{N_0}^+\} \cup \{id \in \boldsymbol{N}^+\} \cup \{phase \in \{active, passive\}\}$

Where *tokens* is the number of tokens contained in the place and *id* is the identifier of the place as assigned by the DEVS simulator.

```
δext (s,e,x) {
retrieve id and number of tokens from message
case id = 0    /* generic message */
increment tokens
    hold in active 0  /* to advertise the number of
tokens */

case id != 0   /* specific message */
    id matches id of this place?
         no:   disregard the message
         yes:  decrement tokens by the number of
tokens specified if there are enough. Otherwise
throw an exception.
       hold in active 0 /* to advertise the number
of tokens */
}end of external transition function


δint (s)  {
passivate      /* wait for the next external event
*/
}


λ(s) {
combine id and tokens state variables in one mes-
sage and send on the out port.
}
```

The Transition Model

Figure 3 illustrates the conceptual description for the DEVS model of a PN transition. It has five input ports and five output ports described below.

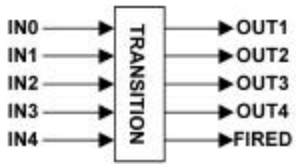

Figure 3. Transition conceptual model

in1 port: this input port is used to be notified of the number of tokens contained in the place(s) which have their out port connected to this input. Places which connect to this port do so because the connection consists of a single connecting arc. That is, if the transition fires, only one token will be removed from the input place(s).

in2, in3 and in4 ports: these input ports serve the same function as the in1 port except that the connection consists of a double, triple and quadruple connecting arc respectively. This implies the input place(s) will loose two, three or four tokens if the transition fires.

in0 port: this input port serves the same function as the in1 port except that the connection consists of an inhibitor arc. That is, the input place must contain zero token for the transition to be enabled and when it fires no token is removed from the place.

out1 port: This output is used to feed one token to all the places which have their in port connected to this port. The id of the messages sent on this port is always zero which causes all places receiving it to update the number of tokens they hold.

out2, out3, out4 ports: These ports serve the same purpose as out1 except they respectively feed two, three and four tokens to all the places which have their in port connected to these ports.

fired port: This output is used to remove tokens from the input places which must have their in port connected to this output port in addition to being connected to one of the input ports.

The formal specification of the model is:

$M = < X, S, Y, \delta_{int}, \delta_{ext}, D, \lambda >$ where:
$X = \{IN0 \in \boldsymbol{N}^+, IN1 \in \boldsymbol{N}^+, IN2 \in \boldsymbol{N}^+, IN3 \in \boldsymbol{N}^+, IN4 \in \boldsymbol{N}^+\}$
$Y = \{OUT1 = 1, OUT2 = 2, OUT3 = 3, FIRED \in \boldsymbol{N}^+\}$
$S = \{\{inputs \in \boldsymbol{N_0}^+\} \cup \{enabled \in bool\}$

Where inputs is the number of input places the transition has and enabled indicates if the transition is enabled or not. Note that the model uses a database to store the id of the input places, the number of tokens they contain and the width of the arcs connecting the places to the transition. This information does not define the state of the transition model which is why it is not included in the definition of S above.

```
δext (s,e,x) {
case port
    in0: set arcwidth (temp var) to 0.
    in1: set arcwidth (temp var) to 1.
    in2: set arcwidth (temp var) to 2.
    in3: set arcwidth (temp var) to 3.
    in4: set arcwidth (temp var) to 4.
- extract id (place of origin) and number of tokens
from the message.
First message we get from this id?
yes: increment inputs.
no: continue
- save id, arcwidth and number of tokens in data-
base.
- scan the entire database to determine if all in-
put places have enough tokens to enable the transi-
tion.
    transition is enabled ?
        yes:   set enabled to true
        hold in (active, random (0 – 60 sec))
        no:    set enabled to false
        passivate
}end of external transition function


δint (s)  {
    inputs = 0?
        yes:   /* transition is a source */
               hold in (active, random (0 – 60 sec))
        no:    passivate
}


λ(s) {
- send 1 on out1 port.
- send 2 on out2 port.
- send 3 on out3 port.
- send 4 on out4 port.

- go through the database and send a message to
every input place via the fired port.
} /* end of output function */
```

Figure 4 illustrates an example of how transition and place models are meant to be coupled to create a PN. Note that for clarity reasons two fired ports where used but the model actually only has one as mentioned previously. The figure shows a transition which is enabled when P1 has no token, P2 has at least two tokens and where P3 receives three tokens when the transition fires. Additionally, P2 looses two tokens because the fired port of the transition is connected to its in port. Even though the fired port of t1 is connected to the in port of P1, the latter does not loose tokens when t1 fires because it is connected via an inhibitor arc.

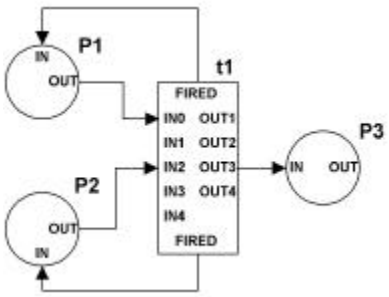

Figure 4. Coupling places and transition

Figure 5 shows the CD++ coupled model definition file equivalent to the coupled model illustrated on Figure 4. Places use the pnPlace atomic model while transitions use the pnTrans atomic model. The five links represent the five connecting arcs. By default, the place model is created with zero tokens. In the case of P2, this has been overridden by the modeler by specifying "tokens : 3" such that at the beginning of the simulation the place would contain three tokens. Another optional parameter is the *inputplaces* parameter which indicates the maximum number of input places that can connect to the transition. By default the maximum is 10 but this can be changed to a lesser or larger value. This parameter limits the amount of memory the transition model allocates to keep information about input places.

```
[top]
components : P1@pnPlace P2@pnPlace P3@pnPlace
t1@pnTrans

Link : out@P1 in0@t1
Link : out@P2 in2@t1
Link : out3@t1 in@P3
Link : fired@t1 in@P1
Link : fired@t1 in@P2

[P2]
tokens : 3

[t1]
inputplaces : 3
```

Figure 5. Model definition file for figure 4

ANALYZING SIMULATION RESULTS

A TCL tool was developed to assist modelers in analyzing test results. Given that what is of interest is the marking of the PN and the firing of the transitions, the tool provides this information in a clear and concise manner as can be seen in Figure 6. The tool does this by parsing the coupled model file to determine the names of the places and transitions used in the model and by parsing the .log file resulting from a simulation.

The first two lines in Figure 6 list the name of the places and transitions which make up the model. Then the initial marking of the PN is shown. In this case, p1 = 5, p2 = 3 and p3 = 0. Then t1 is seen to fire, which results in a (2,1,4) marking. From this one can conclude p1 and p2 are input places to t1 while p3 is an output place of t1. Furthermore, p1 has a triple arc to t1 because it lost 3 tokens due to the firing, p2 has a double arc and p3 has a quadruple arc.

```
Petri Net places: p1 p2 p3
Petri Net transitions: t1

 (5,3,0)— t1->(2,1,4)
```

Figure 6. Output of the Petri tet marking tool

VALIDATION OF MODELS

A first set of tests verified the proper interactions between a place atomic model and a transition atomic model using the simple PN shown in Figure 7. A 10 second simulation run was done without using an external event file. Figure 8 shows the PN markings resulting from the simulation which confirmed the expected results:

- Verify the transition fired continuously because it is always enabled.
- Verify the marking of the PN stayed at (1) because P1 always contained 1 token.

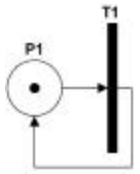

Figure 7. Coupling test 1 model

```
Petri Net places: p1
Petri Net transitions: t1

(1)-- t1 ->(1)-- t1 ->(1) … and so on to the end
of the file.
```

Figure 8. Coupling test 1 PN markings

A second group of tests was aimed to study the proper interactions between a place atomic model and a transition atomic model using the PN shown in Figure 9. A 30 minute simulation run was done without using an external event file.

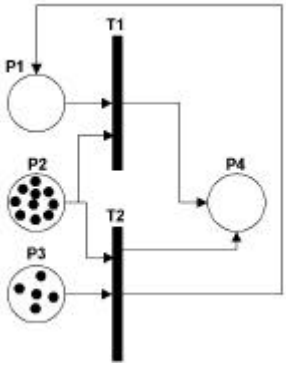

Figure 9. Coupling test 2 model

Figure 10 shows the PN markings resulting from the simulation which confirms the expected results:

- Verify t2 is the first transition to fire.
- Verify t1 and t2 fire five times each in a non deterministic manner when they are both enabled at the same time.
- Verify the initial marking is (0, 10, 5, 0)
- Verify the final marking is (0, 0, 0, 10)

```
Petri Net places: p1 p2 p3 p4
Petri Net transitions: t1 t2

(0,10,5,0)-- t2 ->(1,9,4,1)-- t1 ->(0,8,4,2)--
t2 ->(1,7,3,3)-- middle contents of file not
shown to save space->(1,1,0,9)-- t1 ->(0,0,0,10)
```

Figure 10. Coupling test 2 PN marking

Another set of tests verified the proper interactions between a place atomic model and a transition atomic model using the PN shown in Figure 11. A 30 minute simulation run was done without using an external event file. Figure 12 shows the PN markings resulting from the simulation which confirms the expected results:

- Verify initial marking is(5, 3, 0)
- Verify t1 fires only once.
- Verify P1 and P2 loose 3 and 2 tokens respectively.
- Verify P3 receives 4 tokens.
- Verify final marking is (2, 1, 4)

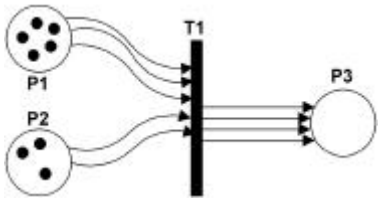

Figure 11. Coupling test 3 model

```
Petri Net places: p1 p2 p3
Petri Net transitions: t1

(5,3,0)-- t1 ->(2,1,4)
```

Figure 12. Coupling test 3 PN markings

SIMULATION EXAMPLES

After having validated the place and transition models, the next step was to simulate real systems. This section describes five simple systems which were modeled and simulated and discusses the results.

Mutual Exclusion Scenario

Figure 13 illustrates a classical mutual exclusion problem as described in [1]. Two processes have to execute critical sections of code (places P3 and P4) but are not allowed to do it at the same time. That is, P3 and P4 must be mutually exclusive. To enforce this rule the processes must therefore grab a semaphore (P5) before entering their critical section (t1 or t2) and release the semaphore after exiting their critical section (t3 or t4). The 30 minute simulation run was started with the following marking: (1, 1, 0, 0, 1).

The set of markings resulting from the simulation showed t1 fired first resulting in a marking of (0, 1, 1, 0, 0). Then t3 fired to bring the PN back to its initial marking. This was followed by the firing of t2 resulting in a marking of (1, 0, 0, 1, 0). t4 then fired bringing the net back to its initial marking. The remainder of the markings were simply a repetition of the above except for the order in which the processes successfully took the semaphore. Sometimes a process got the semaphore just after releasing it which is also expected since transitions t1 and t2 are always enabled at the same time and the decision to fire one or the other is made in a non-deterministic manner.

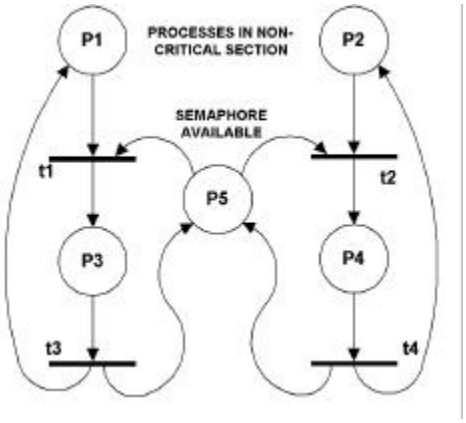

Figure 13. Mutual exclusion scenario

Two-Unit Asynchronous Pipeline

Figure 14 illustrates a two-unit asynchronous pipeline. This model is a modified version of the asynchronous pipelined control unit found in [1]. The idea is to simulate the flow of jobs in the pipeline with the assumption the processing time of each job may be different in each of the units. The 2 hour simulation started with all input and output registers empty: (0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0).

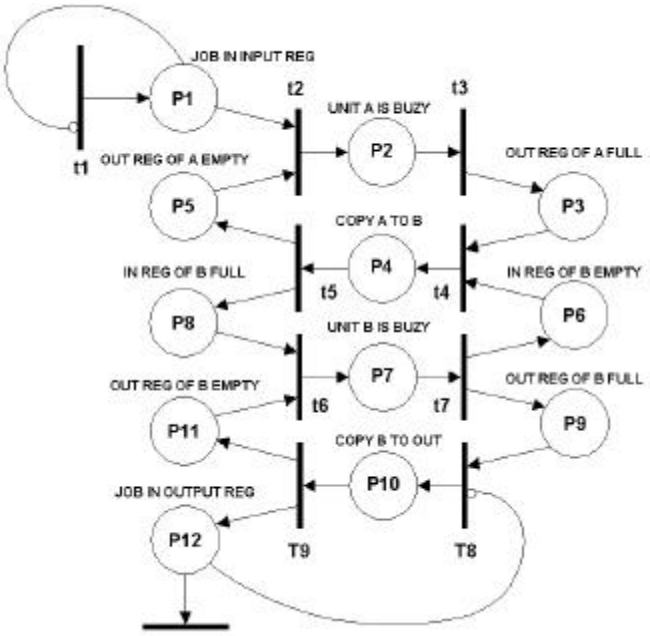

Figure 14. Two-unit asynchronous pipeline

The set of markings generated by showed jobs moving in the pipeline properly. That is they followed the P1-P2-P3-P4-P8-P7-P9-P10-P12 path. Furthermore it was found, as expected, none of the places contained more than one token at any given time and none of the units input registers were empty and full at the same time. That is P3 and P5, P6 and P8, P9 and P11 were found to be mutually exclusive respectively.

CONCLUSIONS

We described how Petri Nets can be simulated using DEVS atomic models for PN transitions and places with an unmodified DEVS simulator and also showed that DEVS simulation results can simply be filtered through a parsing tool to give them a stronger PN flavour. Many simulation examples and results were described to validate the method used and demonstrate its usefulness for implementing a PN-DEVS multi-formalism.

A possible enhancement of the transition model described in the previous sections would be to modify its internal transition function enabling timed Petri Nets modeling. The advantage of doing so with a DEVS model would lie in the fact the timing characteristics of the PN could be formally specified which is not the case with the Petri Net formalism presented earlier.

The implementation of the atomic models, the CD++ tool, the pnmark tool and a more extensive report are public domain and can be obtained in:

http://www.sce.carleton.ca/faculty/wainer/wbgraf/index.htm.

REFERENCES

[1] Peterson, James L. “Petri Nets”. *ACM Computing Surveys*, Vol 3, No. 5. September 1977. pp 221-252.

[2] Zeigler B., Praehofer H., Kim T.G., *Theory of Modeling and Simulation*, second edition, Academic Press, 2000.

[3] Rodriguez, D.; Wainer, G. "New Extensions to the CD++ tool". In *Proceedings of SCS Summer Computer Simulation Conference*, Chicago, IL. 1999.

[4] Wainer, G.; Christen, G.; Dobniewski, A. "Defining DEVS models with the CD++ toolkit". In *Proceedings of the European Simulation Symposium*, Marseilles, France. 2001.

[5] Zeigler, B.; Song, H.; Kim, T.; Praehofer, H. *DEVS Framework for Modelling, Simulation, Analysis, and Design of Hybrid Systems*. In *Proceedings of HSAC*, 1996.

CHRISTIAN J. D. JACQUES is a member of technical staff in the Embedded Technologies Business Unit of Wind River Systems Inc, in Ottawa, Ontario. He received his BEng degree (1989) in Electrical Engineering from Royal Military College of Canada in Kingston Ontario. He is currently working on his M.a.Sc. in Electrical Engineering at Carleton University in Ottawa, Ontario.

GABRIEL WAINER received the M.Sc. (1993) and the Ph.D. degrees (1998, with highest honours) at the Universidad de Buenos Aires, Argentina, and DIAM/IUSPIM, Université d'Aix-Marseille III, France. He is Assistant Professor at the Systems and Computer Engineering, Carleton University (Ottawa, Canada). He was Assistant Professor at the Computer Sciences Dept. of the Universidad de Buenos Aires, Argentina, and a visiting research scholar at the Arizona Center of Integrated Modelling and Simulation (ACIMS, University of Arizona). He has published more than 60 articles in the field of operating systems, real-time systems and Discrete-Event simulation. He is author of a book on real-time systems and another on Discrete-Event simulation. He has been the PI of several research projects, and participated in different international research programs. Prof. Wainer is a member of the Board of Directors of The Society for Computer Simulation International (SCS). He is the coordinator of a group on DEVS standardization. He is Associate Editor of the Transactions of the SCS. He is also a Co-associate Director of the Ottawa Center of The McLeod Institute of Simulation Sciences.