# Formal Framework for Discrete-Event Simulation

Vincent Albert, Clément Foucher

# Formal Framework for Discrete-Event Simulation

**Vincent Albert, Clément Foucher**

*LAAS–CNRS, Université de Toulouse, CNRS, UPS, Toulouse, France*
*(e-mail: {Vincent.Albert, Clement.Foucher}@laas.fr)*

**Abstract:** A formal framework for modelling and simulation of parallel systems named ProjectDEVS is presented. The objective of this framework is to apply a Model-Based System Engineering approach to the development of simulation products for cyber-physical embedded systems. It is intended for the design and automated deployment of virtual prototypes. Models are constructed by coupling concurrent components exchanging data through ports and executed by various simulation schemes, namely simulators. This paper focuses on the integration of a Time Petri Net implementation of a parallel simulator into the framework. The semantics of the parallel simulator is formally described using timed transition system to verify the correctness of the implementation. Then, a model with its simulator can be model checked against formal specification and be rapidly deployed on FPGA or PC via code generators.

*Keywords:* Discrete-event simulation, Parallel simulator, Time Petri Net, Formal methods, MBSE

## 1. INTRODUCTION

A cyber-physical system is a system composed of computing processes (the controller) in interaction with physical processes (the plant) for control and command. During the development cycle of a controller/plant system, different simulation platforms are used for validation purposes. We believe that a Model-Based System Engineering (MBSE) approach allows, for the development of both embedded system and their simulation products, ensuring their reliability, promoting design and test of candidate architectures, mixing real or simulated components of the controller and/or plant, and of course reducing their development time. A MBSE approach typically relies on model transformations and code generators.

At a very early stage, virtual prototyping is used to study the performance of the system and design the control algorithms with a simulated plant. Virtual prototype does not require neither the controller nor the plant to operate in real time. However, it can be executed as a software, i.e. on a desktop simulator, or instantiated on a dedicated digital hardware processing unit or a mix of both. Dedicated digital hardware processing units can be used to accelerate various computations instead of using software. With the advent of Field-Programmable Gate Arrays (FPGAs), one can design a hardware circuit and instantiate it immediately instead of going through the long process of designing an Application-Specific Integrated Circuit (ASIC). Using FPGA devices, the creation of hardware accelerators dedicated to a specific or occasional purpose becomes possible.

Discrete event simulation is widely used for the validation of parallel and distributed systems. NS3 and Omnet++, which are reference tools in the field of computer networks and VHDL simulation in the field of digital hardware processing units, use this discrete event paradigm. In the field of continuous systems simulation, there is a family of asynchronous (event-driven) numerical integrators called QSS, see Cellier et al. (2006) for an overview of this method, that shows very good results. Those simulators all use the same scheme: it is the change of a variable (or signal) value which triggers what we call an event. An event can occur at any time. A variable is updated only when a specific event occurs at a discrete point of time. We call an event a time event when the considered variable is time, or a state event when it is a variable of the model. Whereas discrete-time simulators needs a special mechanism (zero-crossing functions) to handle state event for hybrid systems simulation, in discrete-event simulation, everything is time event because an event is scheduled and it is the time of the next event that makes simulation time advance.

In that context we develop a modelling and simulation tool of parallel systems named ProDEVS based on the Discrete Event System Specification (DEVS) formalism and its simulators. DEVS, formulated by Zeigler (1976), has an abstract syntax for atomic component which is nothing else than an interface (input/output) timed automata. It provides a modular and hierarchical construction of the model with the concept of coupled component that connects atomic component output ports to atomic component input ports. DEVS also defines a set of operational semantics (the way of executing the model), called the abstract simulator in the DEVS community, that can be seen as a Model of Computation (MoC), see Ptolemaeus (2014) for MoCs in Ptolemy. For instance we have the following Discrete-Event (DE) MoCs: Classic DEVS simulator (CDEVS) where components execution is sequential (only one component is executed at a time) and conservative Parallel DEVS simulator (PDEVS), formulated by Chow (1996), where several components can be executed at the same time but causality violations are strictly avoided. Various parallel and distributed simulation researchers have implemented parallel simulators for DEVS. A time warp optimistic DEVS simulator, see Jefferson et al. (1985), where causality might be violated, detected and remedied using roll-back has been implemented by Christensen (1990). A risk-free optimistic DEVS simulator, see Ferscha (1995), where events are assessed for risk before sending has been implemented by Reisinger et al. (1995). See Zeigler et al. (2000) for pseudo-code description of these abstract simulators.

This paper focuses on the integration of a Time Petri Net (TPN) implementation of the PDEVS simulator into ProDEVS. This work results in a formal MBSE framework we called ProjectDEVS which takes a ProDEVS model and its simulator,

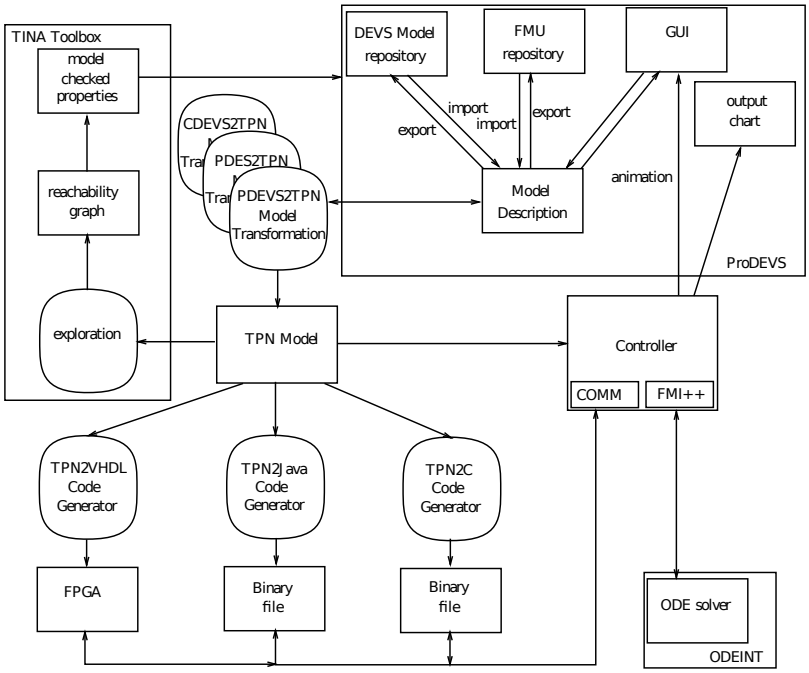


Fig. 1. ProjectDEVS Architecture

automatically transforms them into a TPN and deploy the latter as a program, as digital hardware or as a mix of both. Petri net is very efficient to describe parallelism and concurrency (resource sharing, synchronization) between tasks or processes. The advantages of using TPN as a backbone between a ProDEVS model and the platform dependant virtual prototype are : (1) the development of new simulators are not hand coded anymore, there are specified using temporal logic and designed using TPN, without any impact on the deployment phase, (2) a model can be checked against a formal specification to some extent (formal methods are subject to combinatory explosion) and we can ensure that the virtual prototype is correct, (3) formal verification can be coupled with simulation statically or dynamically (during run-time).

In the next section the architecture of the framework is detailed. Then, Section 3 defines the DEVS formalism, the principles of PDEVS simulator and the class of TPN we use. In Section 4, the rules for implementing a DEVS atomic component and a PDEVS simulator are given. Section 5 describes and illustrates the method we employ to verify the implementation and finally, perspectives and issues are given in Section 6.

## 2. ARCHITECTURE

The architecture of the ProjectDEVS framework is illustrated on figure 1. It includes ProDEVS, the model designer of ProjectDEVS, which includes a GUI offering a block-oriented view for model design. A model is constructed with concurrent components that can be imported from ProDEVS components repository or designed from scratch using input/output timed automata that we specially profiled for DEVS formalism, see Vu et al. (2015). User can create its own repository. Basically, the repository contains components for continuous systems, such as QSS integrators, or quantizers and switches for hybrid systems. We recently integrated FMI cosimulation and model exchange features, see MODELISAR (2014) for FMI specification, such that Fonctional Mockup Units (FMU) can also be imported in the model. We have developed a DEVS-FMI wrapper to synchronise discrete-time simulators with discrete-event simulators using FMI++. The FMI++ library is a utility package, implemented by Widl et al. (2013), that provides simulation functionalities for FMI model exchange and cosimulation specification. It includes a numerical integrator and a state record mechanism for roll-back. From a model description captured in the GUI and a given abstract simulator, a TPN model is generated. This TPN can be exported to the TINA toolbox, see Berthomieu et al. (2006), for model checking.

Then, a description of the structure of the Petri Net with dedicated components to implement places and transitions, and boolean or logical equations to represent enabling and firing conditions can be generated for simulation. This generated code is associated to a Time Manager which is in charge of simulation time events synchronisation and Action Managers to handle data computation in reaction of transition firing. A simulation clock provides events to make the internal TPN state evolve. Combined with a Run Manager aware of the given TPN structure, the prototype can be interfaced to a controller for simulation controls (run, step, break) and data visualisation. Data is recorded on a value change event, and stored along with the associated simulation time. A mapping between the ProDEVS model domain and the platform dependant variables is loaded into the controller for data charts. For software deployment, we have Java and C code generators that provide binary code which is interfaced with the controller. For hardware, a VHDL code is generated in a synthesizable form which can be used on a FPGA. The component representing the model is then wrapped in a register-based structure which can be adapted to most bus interfaces. Then, the simulator is interfaced with the controller through an Ethernet network using a small program running on a processor inside the FPGA. Interfaces for buses used by Xilinx and Altera, AXI and Avalon, are generated along with the model code.

## 3. DEFINITIONS

### *3.1 Discrete Event System Specification*

The following definition is taken from Zeigler et al. (2000). A DEVS atomic component is a tuple $\langle X, Y, S, \delta_{ext}, \delta_{int}, \lambda, ta\rangle$ in which :

- $X = \{(p, v)|\ p \in Iports,\ v \in X_p\}$, is a set of input ports and their values, $Y = \{(p, v)|\ p \in Oports,\ v \in Y_p\}$, is a set of output ports and their values, and $S$ is a set of sequential states,
- $\delta_{ext} : Q \times X \to S$, is the external transition function which defines how the state changes when an input event occurs:
  - $Q = \{(s, e)|s \in S, 0 \leq e \leq ta(s)\}$ is the total state,
  - $e$ is the elapsed time since the last event,
- $\delta_{int} : S \to S$, is the internal transition function which defines how the state changes when a time event occurs,
- $\lambda : S \to Y$, is the output function which defines the output to produce at a time event,
- $ta : S \to \mathbb{R}^+_{0,\infty}$, is the time advance function which is used to determine the lifespan of a state.

DEVS defines an abstract syntax, whence transition functions and/or output functions may execute simple actions described by algebraic equation or complex functions with iterative loop and branch or even a FMU step.

An informal specification of PDEVS abstract simulator is now given. Every component is in a state $s \in S$ at a given time and must be in that state for a period $e = ta(s)$ if no input event occurs. When the time $e$ has elapsed without any input event has occurred for some imminent components, an internal time event occurs, whence, they simultaneously calculate $y = \lambda(s)$ and when every output computation is finished, they simultaneously trigger their internal transition function $\delta_{int}(s)$. If instead, for some components, an input event $x \in X$ occurs before the expiration of $e$, these non imminent components trigger their external transition function $\delta_{ext}(s, e, x)$. Communications are asynchronous, i.e. non-blocking with the possibility of message loss. If upon receipt of a message on the input port $x$, a component is in a state $s$ listening on $x$ i.e., $\delta_{ext}(s, e, x) \in \delta_{ext}$, the message will be processed, otherwise it will be lost and ignored by the receiver. There may be multiple components which are candidates for internal time event at the same time. As a result of coupling, they may also receive input event at the same time. When $ta(s) = \infty$, only an input event will leave the state. When $ta(s) = 0$, the component is immediately imminent.

### *3.2 Time Petri Net*

The following definitions are taken from Berthomieu et al. (2007).

Definition 1. A Time Petri Net with priority (PrTPN) is a tuple $\langle P, T, Pre, Post, \succ, m_0, I_s\rangle$ with :

- $P, T, Pre, Post, m_0$ is a Petri Net where $P$ is the set of places, $T$ is the set of transitions, $m_0$ is the initial marking and $Pre, Post : T \times P \to \mathbb{R}^+_{0,\infty}$ are pre and post incidence matrices respectively.
- $I_s : T \to I^+$ is the static interval function with $I^+$ the set of non empty real intervals with non negative rational endpoints.
- $\succ$ is the priority relation, assumed irreflexive, asymmetric and transitive. Priority are represented by oriented arcs between transitions, the source transition having higher priority.

Moreover, we use various arcs implemented in TINA. The standard arc is written $p \to t$ with $p \in P$ and $t \in T$ gives $Pre(t, p) = 1$ or $t \to p$ with $p \in P$ and $t \in T$ gives $Post(t, p) = 1$. The inhibitor arc, written $p \multimap t$ disables $t$ if there is at least one token in $p$. The reset arc written $p \overset{*}{\to} t$ removes all tokens of $p$ when $t$ is fired. The reset arc is non blocking for a transition $t$, i.e. $Pre(t, p) = 0$. The read arc written $p \multimap\!\bullet\ t$ is blocking, i.e. $Pre(t, p) = 1$, but does not modify the marking of $p$ after firing of $t$.

Definition 2. A state of a TPN is a pair $s = (m, I)$ in which $m$ is a marking and $I$ is a function called the interval function. Function $I : T \to I^+$ associates a temporal interval with every transition enabled at $m$.

Definition 3. The semantics of a PrTPN $\langle P,T,Pre,Post,\succ,m_0,I_s\rangle$ is the timed transition system $\langle S, s_0, \rightsquigarrow\rangle$ where:

- $S$ is the set of states $(m, I)$ of the PrTPN
- $s_0 = (m_0, I_0)$ is the initial state, where $m_0$ is the initial marking and $I_0$ is the static interval function $I_s$ restricted to the transitions enabled at $m_0$.
- $\rightsquigarrow \subseteq S \times T \cup \mathbb{R}^+ \times S$ is the state transition, defined as follows $((s, a, s') \in \rightsquigarrow$ is written $s \overset{a}{\rightsquigarrow} s')$.
- we have $(m, I) \overset{t}{\rightsquigarrow} (m', I')$ iff $t \in T$ and:
  (1) $m \geq Pre(t)$, $t$ is enabled at state $m$
  (2) $0 \in I(t)$, $t$ is fireable instantly
  (3) $(\forall t' \in T)$ then $(m \geq Pre(t')$ and $(t' \succ t) \Rightarrow 0 \notin I(t'))$, there is no transition with higher priority that satisfies 1 and 2
  (4) $(\forall k \in T)(m' \geq Pre(k) \Rightarrow I'(k) =$ if $k \neq t \wedge m - Pre(t) \geq Pre(k)$ then $I(k)$ else $I_s(k))$. After the firing of $t$ then $m' = m - Pre(t) + Post(t)$, transitions that remain enabled (except $t$) preserve their interval before firing, all others transitions are associated with their static interval.
- we have $(m, I) \overset{\theta}{\rightsquigarrow} (m', I')$ iff $\theta \in \mathbb{R}^+$ and :
  (5) $(\forall k \in T)(m \geq Pre(k) \Rightarrow \theta \leq \uparrow I(k))$, a temporal transition $\theta$ is possible if $\theta$ is not larger than the right endpoint of any transition enabled.
  (6) $(\forall k \in T)(m \geq Pre(k) \Rightarrow I'(k) = I(k) - \theta, \theta$ is removed from the interval of every transition enabled before firing of the timed transition.

Every enabled transition must be fired between its associated interval. Our TPN implementation uses only punctual bounded intervals, i.e. under the form $[\theta; \theta]$. In Berthomieu et al. (2007), the authors have found a convenient abstraction of the state graph $SG = (S, s_0, \rightsquigarrow)$ which preserves Linear Temporal Logic (LTL) model checking and marking and decides state reachability. They also cite two alternate constructions of an abstraction, for the subclass of TPNs in which all transitions have bounded static intervals, which preserves Combinatory Temporal Logic (CTL) model checking and branching.

## 4. PDEVS2TPN RULES

A ProDEVS model is a composition of $N$ atomic components exchanging messages. A TPNDEVS (the TPN implementation of a ProDEVS model and its simulator) is a set of $N + 1$ TPNs sharing common places. For every atomic component we have a TPN with one place for every input and output

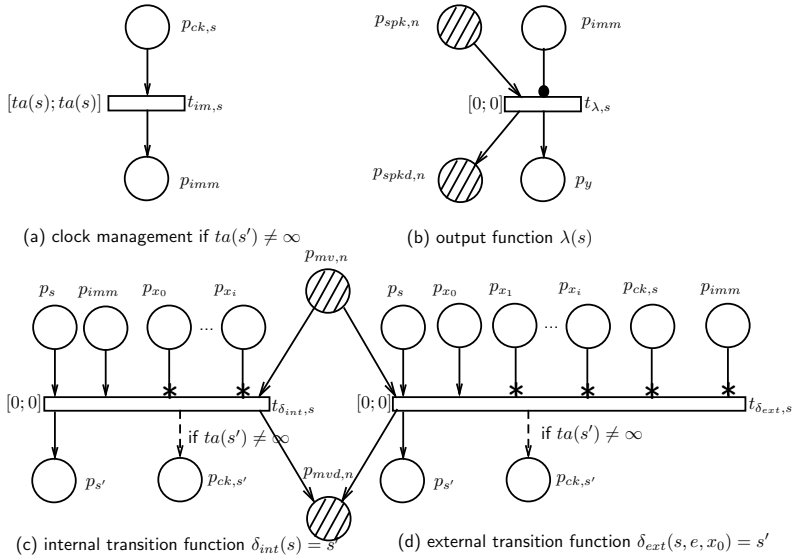


Fig. 2. TPN blocs for atomic component

port. A connection from an output port to an input port results in the fusion of the two corresponding places. Another TPN, the coordinator, is used for synchronisation and scheduling of atomic components. Atomic components and coordinator communicate via places. The figure 3 illustrates a TPNDEVS structure where an arrow represents a fusion of places.

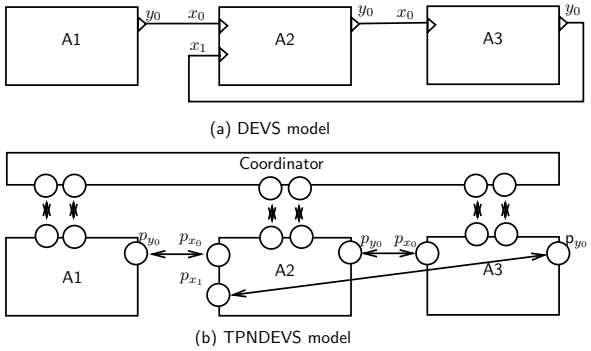


Fig. 3. TPNDEVS structure

### *4.1 TPN model of the atomic component*

An atomic component is given by 4 elementary blocs as shown in figure 2: local clocks management, outputs functions, internal and external transitions.

For every atomic component $n = \langle X, Y, S, \delta_{ext}, \delta_{int}, \lambda, ta \rangle$ we define a TPN $\langle P_n, T_n, Pre, Post, m_0, I_s \rangle$ such that:

(1) for every state $s \in S$ we have a place $p_s \in P_n$
(2) for every input port $x \in X$ we have a place $p_x \in P_n$
(3) for every output port $y \in Y$ we have a place $p_y \in P_n$
(4) we have places $p_{spk,n}, p_{spkd,n}, p_{mv,n}, p_{mvd,n} \in P_n$ which are used for communication with the coordinator
(5) we have a place $p_{imm} \in P_n$. When $p_{imm}$ is marked, it denotes that the component is imminent
(6) for every state $s \in S$ such that $ta(s) \neq \infty$ we have a place $p_{ck,s} \in P_n$ and a transition $t_{im,s} \in T_n$. We have $p_{ck,s} \rightarrow t_{im,s}$, $t_{im,s} \rightarrow p_{imm}$ and $Is(t_{im,s}) = [ta(s); ta(s)]$
(7) for every output function $\lambda(s) \in \lambda$ we have a transition $t_{\lambda,s} \in T_n$ such that $Is(t_{\lambda,s}) = [0;0]$, i.e. the firing is immediate, $p_{imm} \multimap t_{\lambda,s}$ and $t_{\lambda,s} \rightarrow p_y$, the marking of $p_y \in P$ denotes that a data is available in the output port $y \in Y$ and $t_{\lambda,s}$ denotes the firing of the output function from $s$
(8) for every internal transition function $\delta_{int}(s) \in \delta_{int}$, we have a transition $t_{\delta_{int},s} \in T_n$ such that $Is(t_{\delta_{int},s}) = [0;0]$ and $p_{imm} \rightarrow t_{\delta_{int},s}$. The marking of $p_s$ denotes that the current state is $s$. For every internal transition function $\delta_{int}(s) = s'$, we have $p_s \rightarrow t_{\delta_{int},s}$ and $t_{\delta_{int},s} \rightarrow p_{s'}$, and for every input port $x \in X$, we have $p_x \overset{*}{\rightarrow} t_{\delta_{int},s}$. Finally, if $ta(s') \neq \infty$ we have $t_{\delta_{int},s} \rightarrow p_{ck,s'}$
(9) for every external transition function $\delta_{ext}(s,e,x) \in \delta_{ext}$, we have a transition $t_{\delta_{ext},s} \in T_n$ such that $Is(t_{\delta_{ext},s}) = [0;0]$. For every external transition function $\delta_{ext}(s,e,x) = s'$, we have $p_s \rightarrow t_{\delta_{ext},s}$, $t_{\delta_{ext},s} \rightarrow p_{s'}$ and $p_x \rightarrow t_{\delta_{ext},s}$, and for every input port $x' \in X$ (except $x$), we have $p_{x'} \overset{*}{\rightarrow} t_{\delta_{ext},s}$. Finally, we have $p_{imm} \overset{*}{\rightarrow} t_{\delta_{ext},s}$, $p_{ck,s} \overset{*}{\rightarrow} t_{\delta_{ext},s}$ and if $ta(s') \neq \infty$ we have $t_{\delta_{ext},s} \rightarrow p_{ck,s'}$.

Consider an atomic component $n$ at the initial state $s \in S$, then $p_{ck,s}$ and $p_s$ are marked. The only condition for firing $t_{im,s}$ is the marking of $p_{ck,s}$. If $p_{ck,s}$ remains marked for time $ta(s)$ because no external event has occurred, then $t_{im,s}$ is fired and $p_{imm}$ is marked denoting that $n$ is imminent. An output function is then triggered by the firing of $t_{\lambda,s}$ which produces a token in $p_y$. Then, the internal transition function is triggered by firing $t_{\delta_{int},s}$. Every input port place is emptied and the component is in a new state $s'$ denoted by the marking of $p_{s'}$. If an input external event on port $x \in X$ has occurred before $ta(s)$ expired, denoted by the marking of the input port place $p_x$, the external transition function is triggered by the firing of $t_{\delta_{ext},s}$. Every input port place is emptied and the token in $p_{ck,s}$ is consumed. $n$ is now at state $s'$ denoted by the marking of $p_{s'}$ and a new cycle starts. If $n$ receives an input event on $x$, $m(p_x) = 1$ while $n$ is imminent, $m(p_{imm}) = 1$, then there is a conflict between $t_{\delta_{int},s}$ and $t_{\delta_{ext},s}$ that can be resolved by adding a priority.

### *4.2 TPN model of a ProDEVS model and its PDEVS simulator*

The semantic of a TPNDEVS model is a game, where the players are the atomic components, that takes place in two

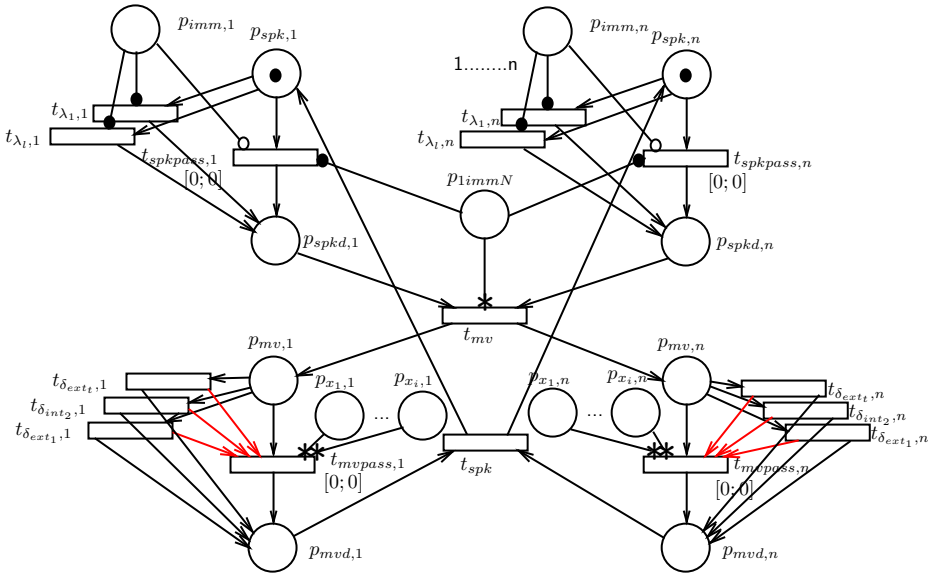


Fig. 4. PDEVS coordinator

stages for every simulation step. In the first stage, players can *speak* by triggering an output function and in the second stage they can move by triggering an internal or an external transition function. For each stage players can also pass. Each stage is modelled by a Petri Net parallel structure. For each simulation step, the players will speak in parallel, then synchronise each other, then move in parallel and finally synchronize again.

A PDEVS coordinator is a PrTPN $\langle P, T, Pre, Post, \succ, m_0, I_s \rangle$ such that:

(1) we have a place $p_{1immN} \in P$ denoting the number of imminent components at a given simulation cycle and we have transitions $t_{spk}, t_{mv} \in T$ denoting the time to speak or to move respectively.
(2) for every atomic component $n \in N$ we have:
- places $p_{spk,n}, p_{spkd,n}, p_{mv,n}, p_{mvd,n} \in P$ and transitions $t_{spkpass,n}, t_{mvpass,n} \in T$. $p_{spk,n}$ and $p_{mv,n}$ denoting that component $n$ can move and speak respectively. $p_{spkd,n}$ and $p_{mvd,n}$ denote that $n$ has spoken or moved respectively. $t_{spkpass,n}$ and $t_{mvpass,n}$ are fired if $n$ pass its turn at stage speak and move respectively
- $p_{imm,n} \multimap t_{spkpass,n}$, denotes that a component can pass its turn only if it is non imminent. For every $t_{im,s} \in T_n$ we have $t_{im,s} \succ t_{spkpass,n}$
- for every $t_{\lambda,s} \in T_n$, we have $p_{spk,n} \rightarrow t_{\lambda,s}$, $t_{\lambda,s} \rightarrow p_{spkd,n}$ and $t_{\lambda,s} \rightarrow p_{1immN}$
- for every $t_{\delta_{int}} \in T_n$, we have $p_{mv,n} \rightarrow t_{\delta_{int}}$, $t_{\delta_{int}} \rightarrow p_{mvd,n}$, $t_{\delta_{int}} \succ t_{mvpass,n}$. Idem for every $t_{\delta_{ext}} \in T_n$
- for every $p_x \in P$ we have $p_x \overset{*}{\rightarrow} t_{mvpass,n}$.

The graphical representation of the TPN model of a ProDEVS model and its PDEVS simulator is given on figure 4.

$p_{1immN}$ is used to preserve deadlock if every component is in state $s$ with $ta(s) = \infty$. It denotes that at least one component among $N$ must be imminent to continue the game. It is the only place of the all TPNDEVS which is $N$-bounded. All others places are 1-bounded.

The initial marking gives one token in places $p_{spk,1}, ..., p_{spk,n}$ denoting to the players that they can speak. $p_{spk,n}$ is consumed either by the firing of an output function transition or by $t_{spkpass,n}$ if $p_{imm,n}$ has no token. Every component can trigger an output function if it is imminent otherwise it passes. At the end of this stage, $t_{mv}$ is enabled. Then, the marking of $p_{mv,1}, ..., p_{mv,n}$ denotes it's time to move. A component $n$ can trigger an external or internal transition function or pass by the firing of $t_{mvpass,n}$. Note that input port places are emptied by the firing of $t_{mvpass,n}$. Indeed, it is possible that a non imminent component receives inputs while it is in a state that do not accept these inputs. When every component has moved a new cycle starts.

## 5. VERIFICATION

Model checking consists in applying temporal logic to semantics. To reason by model checking on a ProDEVS model, the transformation must be sound. A transformation is sound if the semantics of the ProDEVS model with its simulator, called the abstract semantics, cover all possible cases of the semantics of the corresponding TPN, called the concrete semantics. Whence, a logic formula is satisfied in the concrete only if it is satisfied in the abstract. We have manually defined abstract semantics with timed transition system $(S, s_0, \rightsquigarrow)$ as given by definition 3. The bottom graph in figure 5 shows a part of the abstract semantics for the phase *speak* with three imminent components in the model. Transition labels a, b and c mean that component A, B and C respectively have spoken, i.e. an output event has been computed. This graph says that, in PDEVS, if multiple components are candidate for time internal event, the outputs can be computed in parallel. Then, the abstract semantics is mapped on the concrete semantics as illustrated in figure 5. The top graph shows the part of the state graph given by TINA with a mapping to the abstract semantics. We can observe that the abstract semantics covers all the possible cases of the concrete semantics and that markings, states and traces are preserved.

The figure 6 shows a part of the abstract semantics for the phase *speak* then *move* with three components A, B, C. a, b and c are like before, d (respectively e) means that component A (respectively B) has moved, i.e. an external or an internal transition has been computed. f (respectively g) means that component C has computed an internal transition (respectively an external transition). This case can happen if A, B and C

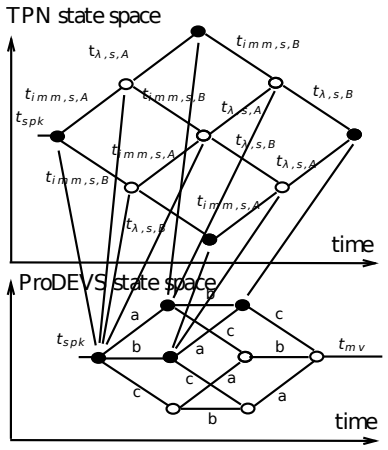


Fig. 5. Abstract semantics and mapping to concrete semantics

are all candidates to internal time event, and if C is in a state that listens on external transition g with an input connected to the output triggered by the output function a. There is an unresolved non-determinism in the ProDEVS model because an internal event time is equal to an input external event time which leads to consider two transitions that potentially brings the system to different states. Again, after the mapping, one can observe that the abstract semantics covers all the possible cases of the concrete semantics and that branching is preserved.

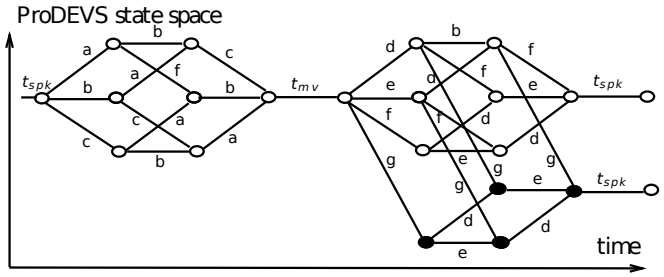


Fig. 6. Abstract semantics with conflict between internal and external events

## 6. CONCLUSION

This paper shows the integration of a Time Petri Net implementation of a parallel simulator into a modelling and simulation tool for virtual prototype development. A designed model and its simulator are automatically transformed into a correct TPN. Model checking, to verify absence of deadlock, detect non determinism, ensure reachability or safety, can be performed onto the abstract model domain using the TINA toolbox. However, as TINA only handles integers for variable and time, the data part, i.e. the action managers of the ProDEVS model are not part of the state space if it is not finite (every variable including time advance function is a bounded integer). Moreover every time advance function must be static.

The TPN model is then automatically deployed on a platform via code generators. This has at least two advantages. First, we feel more confident and comfortable in implementing simulators with TPN rather than manually code it which is error prone while TPN2Code generators are developed once for each execution platform then works for every simulator and every model. Second, in DEVS, a simulator is constructed hierarchically in order to preserve causality violation with local clock synchronisation at each level. Our transformation flattens the model so it improves the performance of the simulation by eliminating intermediate coordinators and message passing.

The other source of overall performance improvement of the simulation comes from the hardware virtual prototype. Compared to software execution, there are two major differences in the run time: the hardware generation can be very long (up to quarter an hour), where the software code compilation takes only seconds. But the hardware execution time is unrivaled by software: only a few clock cycles are used to perform a full simulation step, where the software code is dependant on a sequential execution scheme which slows it down. Thus, for small models very quickly executed on software, one will have no interest in using the hardware execution. For large models and very long simulations times however, the hardware penalty coming from the circuit generation is very quickly offset by the gain in execution time.

## REFERENCES


B. Berthomieu, F. Vernadat. Time Petri Nets Analysis with TINA. In Proceeding of *3rd Int. Conf. on The Quantitative Evaluation of Systems (QEST), IEEE Computer Society*, 2006.

B. Berthomieu, F. Peres, F. Vernadat. Model-checking Bounded Prioriterized Time Petri Nets. In Proceeding of *5th Automated Technology for Verification and Analysis Symposium (ATVA)*, 2007.

F. E. Cellier and E. Kofman. Continuous System Simulation. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

A. Chow. Parallel DEVS: a Parallel, Hierarchical, Modular Modeling Formalism and its Distributed Simulator. *SCS Transactions on Sim 13(2)*, 1996.

E. R. Christensen. Hierarchical Optimistic Distributed Simulation:Combining DEVS and Time Warp. *Doctoral Dissertation, University of Arizona*, 1990.

A. Ferscha. Probabilistic Adaptive Direct Optimism Control in Time Warp. In Proceedings of the *9th Workshop on Parallel and Distributed Simulation*, 1995.

D. Jefferson and H. Sowizral. Fast Concurrent Simulation Using the Time Warp Mechanism. In Proceedings of the *SCS Distributed Simulation Conference*, 1985.

MODELISAR Fonctional Mockup Interface specification 2.0. https://www.fmi-standard.org/. (2014)

Claudius Ptolemaeus, Editor. System Design, Modeling, and Simulation using Ptolemy II. Ptolemy.org, 2014.

G. Reisinger and H. Praehofer. Object Oriented Realization of a Parallel Discrete Event Simulator. In Proceedings of the *Eurosim. Congress*, Vienna, Austria, 1995.

L. H. Vu, D. Foures, V. Albert. ProDEVS: An Event-driven Modeling and Simulation Tool for Hybrid Systems Using State Diagrams. In Proceedings of *8th International Conference on Simulation Tools and Techniques* (SIMUTOOL), Athens, Greece, pp. 29-37, 2015

E. Widl, W. Müller, A. Elsheikh, M. Hörtenhuber, and P. Palensky. The FMI++ Library: A High-level Utility Package for FMI for Model Exchange. In Proceedings of the *IEEE Workshop on Modeling and Simulation of Cyber-Physical Energy Systems*, 2013.

B. P. Zeigler Theory of Modeling and Simulation. Academic Press, 1st edition, 1976.

B. P. Zeigler, H. Praehofer, and T. G. Kim. Theory of Modeling and Simulation. Academic Press, 2nd edition, 2000.