Second Order GDEVS Abstraction of Electronic Circuits

Nesrine Driouche, Maamar El Amine Hamri, Norbert Giambiasi

Second Order GDEVS Abstraction of Electronic Circuits

N. Driouche **M. Hamri** **N. Giambiasi**
Aix Marseille Université, CNRS, ENSAM, Université de Toulon, LSIS UMR 7296
13397, Marseille cedex 20, France
{nesrine.driouche, amine.hamri, norbert.giambiasi}@lsis.org

ABSTRACT
Generalized discrete event is a well-known paradigm since 2000 to model and simulate continuous systems. Electronic circuits is composed of individual electronic components excited by continuous signals in order to carry out complex operations, to move data, etc. However the designers do not do experimentations and analyses directly on electronic circuits, they need conceptual models. Discrete event simulation provides to designers an alternative to analyse such circuits. In this context, the GDEVS (Generalized Discrete EVent System) formalism provides an efficient framework to abstract faithfully the trajectory of continuous systems.

The approach discussed in this paper, attempts to supply to the designers of electronic circuits a framework based on piecewise polynomial functions of second order to abstract continuous signals and to allow discrete event simulations. We have already designed a software tool to validate the discussed approach, which is under development.

Author Keywords
GDEVS; Logic Gates; Piecewise polynomial signals.

ACM Classification Keywords
I.6.5 SIMULATION AND MODELING (Model Development) : Modeling methodologies.

INTRODUCTION

Nowadays, the use of electronic circuits in order to electrify objects (personal computer, cellular phone, etc.) is an obvious fact. For example, the number of transistors in central processor units (CPU) doubles each year (from 2300 transistors at 1970 to 1 million at 2010 for the Itanium 2 processor ©), smart cities grow more and employ more connected devices (Internet of Things), etc. For these reasons, the design and analysis of electronic circuits remains a topical research field.

The Modeling and Simulation (M&S) covers the field of electronic circuits. It allows designers to experiment such circuits without manufacturing; thus, a cheap but efficient solution arises to designers and manufacturers. Moreover, the literature in this field provides two kinds of simulation. The discrete time simulation widely used to compute output signals of designed circuits. But, the computation time remains considerable due to useless computations that occur each update of the time. Whereas, the discrete event simulation provides basic and efficient algorithms to compute output signals at right time with less statements.

By looking at discrete event simulation, Discrete Event system Specification (DEVS) [8] provides a general and extensible framework to design and to simulate any dynamic system, so electronic circuits are perfectly adapted to this framework. The designer may model elementary components using DEVS atomic, then he couples these components using DEVS coupled to make the network of components which represent the electronic circuit. Finally, the DEVS core simulation computes expected outputs according to injected inputs to the circuit.

Few works attempt to employ DEVS as a basis of modeling and simulation logic gates (an abstraction of electronic circuits). Studies proposed in [7, 1] , use logic gates as an application to DEVS in order to validate the proposed approaches and do not tackle the main problems of logic gates. In fact, these works use simple abstraction of input and output signals into two boolean values 0 and 1, they do not consider the unknown state that occurs often in rs-latches and flip-flop at initialization, synchronisation between concurrent subgates, etc.

In ordert to approximate accurately signals of electronic circuits, we propose to use the Generalized Discrete EVent system Specification (GDEVS) [3] to catch more faithfully the signal dynamics. GDEVS is a uniform paradigm to model and simulate continuous systems. In fact, Giambiasi & Carmona [2] showed that such a paradigm is able to model continuous functions (like integrator, etc.) using piecewise polynomial functions. On our part, we successfully used GDEVS to model and simulate logic gates based on piecewise linear functions. However, we hope to extend this previous work to piecewise polynomial functions of second order to approximate more accurately signals while computing the right time of outputs.

The paper is organized into three main parts : the first part recalls definitions on GDEVS formalism, then the second part discusses the proposed approach and highlights the contributions of a second order GDEVS abstraction for the logic gates; and finally, the third part introduces the software GDEVSLogic (currently under development) for modeling and simulation of logic gates.

SummerSim-SCSC 2016 July 24-27 Montreal, Quebec, Canada

STATE OF THE ART

Generalized Discrete Event System Formalism

For complex real-world systems that are highly dynamic, the use of piecewise constant input-output trajectories, for a given sampling time interval, may not succeed in accurately modeling the system behavior. Traditionally, under these circumstances, the sampling time interval is shortened to limit the representational error and achieve acceptable accuracy, at the cost of increased simulation execution time. GDEVS [3] adopts a radically new approach wherein it focuses on a system characteristic, namely the function that represents the system behavior in the given time period, and increases its complexity from an identity function (classical Discrete Event System models) to a higher order function. In this section, we recall the concept of generalized discrete events.

A piecewise polynomial trajectory, expressed through symbol w and shown in Figure 1, is a collection of individual segments over a continuous time base.

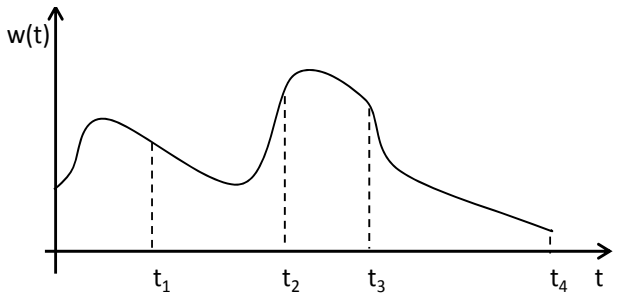


Figure 1. Piecewise polynomial trajectory.

The key characteristics include the following:

- There exists a finite number of time intervals $[te_i, te_{i+1}]$, with which tuples $(a_0, a_1, ..., a_n)$ are associated, where the a_i are constants, and
- $\forall t \in [te_i, te_{i+1}] w(t) = a_0 + a_1 t + ... + a_n t^n$
  $w_{[te_0, te_n]} = w_{[te_0, te_1]} \circ w_{[te_1, te_2]} \circ ... \circ w_{[te_{n-1}, te_n]}$ where $\circ$ represents the left concatenation operator over the individual segments.

The coefficient values of a given individual segment $w_{[te_i, te_{i+1}]}$ are defined by the tuple $(a_0, a_1, \ldots,)$. Formally, the coefficient function *Coef* associates for each polynomial a tuple of real constants over the time interval $[t_{e0}, t_{en}]$. Thus, *Coef* : $\Psi \rightarrow A^n$, where Ψ a set of polynomials, and A a set of real constants. Also, the following function holds for a given continuous polynomial segment $w_{[te_i, te_{i+1}]}$ over the time interval $[te_i, te_{i+1}]$, the components of the coefficients are $(n+1)$ constants.

In order to determine a polynomial trajectory on a time interval, given the coefficients as a function of time, the inverse function, *Coef*$^{-1}$ is defined:

$Coef^{-1} : A^n \rightarrow P$ where P is the set of polynomials of order $n+1$

Coef$^{-1}(a_0, a_1, \ldots, a_n) = a_0 + a_1 t + \ldots + a_n t^n$
and $w_{[t_i, t_j]} \circ$ *Coef*$^{-1} : A^n \rightarrow A^{'}$

Consequently, under GDEVS, events are defined for the coefficients obtained from piecewise polynomial trajectory. Otherwise, a coefficient event or a generalized discrete event is an instantaneous change of at least one of component tuple that defines the coefficient values, i.e. : in a given time interval $[t_0, t_n]$ of a piecewise polynomial trajectory, there exists a generalized discrete event at time t_i if:

$$\textit{Coef}(w_{[t_{i-1}, t_i]}) \neq \textit{Coef}(w_{[t_i, t_{i+1}]})$$

In this study, we are interested in GDEVS event of second order that means signals are approximated into piecewise polynomial functions of second order. Consequently, each GDEVS event is a triplet of constants $(a, b, c) \in \mathbb{R}^3$ and the corresponding trajectory is the polynomial function $at^2 + bt + c$. Thus, the trajectory shown in Figure 1 may be defined with the following piecewise function :

$$trajectory(t) = \begin{cases} a_0 t^2 + b_0 t + c_0 & \text{if } t \in [0, t_1[\\ a_1 t^2 + b_1 t + c_1 & \text{if } t \in [t_1, t_2[\\ a_2 t^2 + b_2 t + c_2 & \text{if } t \in [t_2, t_3[\\ a_3 t^2 + b_3 t + c_3 & \text{if } t \in [t_3, t_4[\end{cases} \quad (1)$$

This abstraction corresponds to four GDEVS events $(a_0, b_0, c_0), (a_1, b_1, c_1), (a_2, b_2, c_2)$ and (a_3, b_3, c_3) that occur at time $0, t_1, t_2$ and t_3 respectively.

M&S Software of Logic Gates

The World Wide Web (WWW) provides a lot of tools to design and simulate logic gates like simulink ©, verilog ©, etc. Some of them provide basic functionalities useful to learn simple rules on the boolean logic (truth tables), the others provides advanced functionalities like formal verification of circuits, stochastic simulations, etc. Tools like verilog and FPGA are based on programming-driven design for which knowledge on programming is indispensable for any design of a given logic gate. Such a design allows constructing large scale circuits with high number of basic and reused gates and a complex coupling; unlike tools with graphics-driven design in which only simple and small circuit can be constructed but remains a quick and efficient design.

In all these software, the design of logic gates is based essentially on interconnected basic blocks. These gates are often a composition of a basic gate *and*, *or*, or *not* in order to express the functional view of a given electronic component and a delay block to express the amount of time necessary for a given digital pulse crosses the material. A digital pulse is a series of rise and fall events well stamped according to a global clock. In addition such a pulse is limited to two voltage values low and high (by default 0 and 5.0 volts) which corresponds to 0 and 1 in boolean logic, respectively. Unfortunately, this is insufficient to obtain accurate simulations of electronic circuits.

NEW APPROACH

Most methods and tools known from the literature employ a simple abstraction of a pulse of circuits into two values 0 and 1 corresponding to low and high pulse values, respectively. In order to design more accurately a pulse, we introduce the

concept of generalized discrete event of second order and the basic logic gates *and*, *or* and *not* based on this new concept.

GDEVS Event of Second Order

In [5], we proposed the concept of first order abstraction for discrete events and we applied it successfully to design and simulate logic gates. However, a second order abstraction of pulse allows us to design it more accurately over time; whereas, first order and constant discrete events introduce more errors. In addition, by using a pulse designed with a generalized discrete event of second order, we are able to expect times of crossing with another pulse, and high and low voltages mathematically, by resolving equations of second order.

Let us consider the following pulse shown on Figure 2, and the abstractions given on Figures 3, 4 and 5.

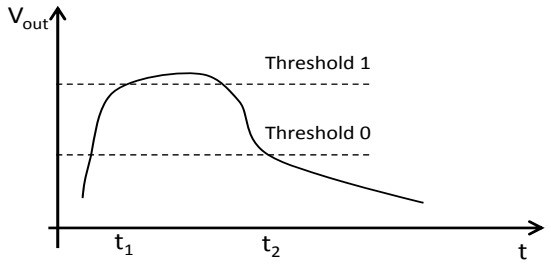


Figure 2. An electronic pulse.

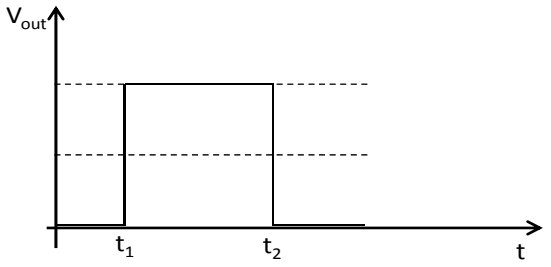


Figure 3. A boolean abstraction of a pulse.

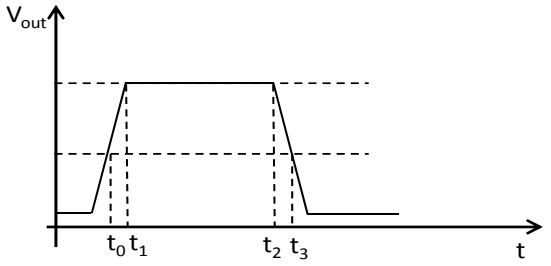


Figure 4. A linear abstraction of a pulse.

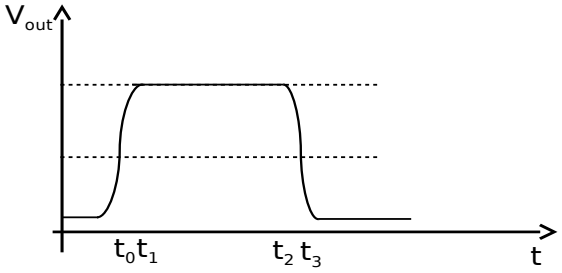


Figure 5. A second order polynomial abstraction of a pulse.

Figure 5 shows clearly that an abstraction of second order follows nearly the dynamics of the pulse shown in Figure 2 than those shown in Figures 3 and 4.

Logic Gates Based on Second Order Abstraction

In the electronic circuit field, engineers design a logic gate by a composition of three blocks in series :

- a boolean functional bloc: may be the *and*, *or* or *not* logical operator,
- a delay block: catches the input signal and delays it to send out it at right time, and
- an amplifier block: increases or decreases the power of the input signal with limiting its value between low and high voltages.

In order to design such a logic gate in optimized way, we decide to encapsulate these blocks inside a unique GDEVS atomic model of second order to design accurately output pulses and their times. The functions $\delta_{ext}()$, $\delta_{int}()$, $\lambda()$ allow representing the functional block, the function *timelife()* allows representing the delay block and the function $\lambda()$ allows taking into account the block amplifier when the output is ready to be sent out. Consequently, these three blocks are designed with only one block.

The purpose of this model is to compute the output pulse according to input ones:

$$pulse_{out}(t) = function(pulse_{in1}(t), pulse_{in2}(t), \ldots) \quad (2)$$

The boolean function is computed according to one of the following equations :

$$And_{out}(t) = min(pulse_{in1}(t), pulse_{in2}(t), \ldots) \quad (3)$$

$$Or_{out}(t) = max(pulse_{in1}(t), pulse_{in2}(t), \ldots) \quad (4)$$

$$Not_{out}(t) = V_{high-low} - (pulse_{in}(t)) \quad (5)$$

Knowing that the operator *min* (*max*) is equivalent to the *and* (*or*) logic and the output pulse corresponds to one of the input pulses, we use these operators in order to compare input pulses (in form of polynomial functions) between them and to compute output ones. For the boolean function *not*, we need only to invert the received pulse by considering the limit voltages (high and low).

Let us recall that in this study, input and output pulses are piecewise polynomial functions of second degree in order to define generalized discrete events of second order, i.e, any pulse is described with a list of three real values (a, b, c) as follows:

$$pulse(t) = at^2 + bt + c | a, b, c \in \mathbb{R} \text{ and } t \in \mathbb{R}+ \quad (6)$$

Moreover, for seek of simplicity we consider the logic gates *and* and *or* with only two input ports and one output port. Consequently, the GDEVS atomic models *and* and *or* hold :

- two input ports and one output port.
- the set of state variables $\{a_1, b_1, c_1, a_2, b_2, c_2\}$ to save the coefficients of each input pulse and the set $\{a_n, b_n, c_n\}$ to save the coefficients of the pulse to send out at crossing time t_n. Note that at crossing time the two input pulses $input_1()$ and $input_2()$ are equal.
- the functions $\delta_{ext}()$ and $\delta_{int}()$ that compute the next state for each pulse that occurs.
- the function $\lambda()$ that computes the output to send out.
- the function *lifetime()* that computes the remaining time for the next output pulse and that considers the specified delay.

Note that the resolution of the equation $input_1(t) = input_2(t)$ allows computing exactly at which times the input pulses will cross.

Example

Let us consider an ideal logic gate *and* with two inputs, one output, without delay (0 t.u) and without amplifier. The two inputs follows these trajectories :

$$input_1(t) = \begin{cases} t^2/10 - t + 3 & \text{if } t \in [0, 10[\\ t^2/9 - t - 7 & \text{if } t \in [10, \infty[\end{cases} \quad (7)$$

$$input_2(t) = \begin{cases} 0 & \text{if } t \in [0, 2[\\ t^2 - 22t + 50 & \text{if } t \in [2, 20[\\ t^2 + t + 1 & \text{if } t \in [20, \infty[\end{cases} \quad (8)$$

At initialization (time = 0), the output of this gate corresponds to the $input_2(t = 0) = 0$ because it is smaller than the $input_1(t = 0) = 3$. Then, when a new event occurs at $input_2$ at time = 2, the new output corresponds to $input_1$ ($input_1(t = 2) < input_2(t = 2)$). At first crossing time $t_{n1} = 2,49$ (after having solved the equation $input_1(t) = input_2(t)$ on the time interval $[2, \infty[$, i.e, $t^2/10 - t + 3 = t^2 - 22t + 50$), the output corresponds again to $input_1$ because at this time and after, $input_1(t)$ increases more lately than $input_2(t)$. This affirmation is based on the derivative function of each trajectory (the trajectory with the smallest derivative value will have the smallest voltage at crossing time until a new event occurs).

Note that this reasoning is applied each time a new input is received on a given port.

SIUMULATION

Core Simulation

Any classic DEVS simulator, designed in object oriented paradigm, may simulate a given electronic circuit described in GDEVS. It should allows defining logic gates in hierarchical way in order to use existing ones and more important creating GDEVS events and managing them correctly.

fwkDEVS [4] is a Java © general and extensible framework for the simulation of DEVS and GDEVS models. It allows designing both atomic and coupled models through the class `DEVSModel`. The specialization of class `DEVSCoupled` allows the instantiation of submodels and the implementation of the method `select()` that accepts one input argument the different conflicting models and identifies which one must be processed first.

For our study of electronic circuits, it is easy to specialize the class `DEVSModel` to design any logic gate. Algorithm 1 is imported into the class `GateCoupled` through the method `select()`. Consequently, the designer focuses on defining the instances of reused gates and implementing the coupling over them using the methods `addIC()`, `addEIC()` and `addEOC()` that correspond to the internal, external input and external output couplings respectively.

At the end, the designer should couple the circuit with generators (instances of the class `Generator`) to input signals and with transducers (instances of the class `Transducer`) to follow gate outputs. Then, the simulation may start after having edited the input file of each generator; consequently transducers will output files storing each signal sent out by an excited gate.

Conflicting Gates

In a given circuit, a conflict occurs when there are a lot of subgates candidate to send out a signal. This fact amplifies when the delay gate is null (0 t.u). According to how imminent subgates are processed by the simulator, they may or not send out signals which leads to errors.

Let us consider two gates and_0 and and_1 in series with a null delay. Suppose at time t the two gates are imminent, i.e. ready to fire an internal transition. By looking at the structure of the composite gate, if we handle the gate and_1 then the gate and_0, the gate and_1 will send out two signals at the same time t. However, the second scheduling which handles the gates and_0 then and_1 allows the correct processing and produces for each gate only one signal at time t.

In classic DEVS and GDEVS, the function *select()* handles conflicting events. It allows the designer to define in how manner the imminent submodels should be proceed. However, it is necessary for the designer to expect the scenarios of conflicting events and to define which of the imminent model should be selected. In simple models with less submodels and light coupling between them, the designer may expect the conflicting scenarios and may describe the useful statements to handle imminent models. However, in complex model, where there are a lot of submodels and interlaced coupling, the designer is not able to enumerate conflicting scenarios. A

realistic solution consists of designing an algorithm able to handle imminent models, according to the application nature.

In logic gates, in order to have the correct sequence of output signals, the selected imminent gate should not be influenced by no other imminent one. Thus, we propose Algorithm 1 to handle any simulation where there are imminent gates to handle.

Algorithm 1 Selection of imminent gates

Require: l: list of imminent gates l': additional list
  copy l to l'
  for each gate g ∈ l **do**
    if influencer of g ∈ l **then**
      withdraw g from l'
    end if
  end for
  return l'

Algorithm 1 computes and identifies imminent gates without imminent influencers [1]. At the end, we obtain the set of imminent gates to handle by the simulator.

While the circuit is defined with a null delay, we should generalize the definition of influencers of a subgate to include each subgate connected upstream.

GDEVSLogic: A SOFTWARE FOR M&S LOGIC GATE

GDEVSLogic is a software developed to end-users. It allows designing, checking, simulating and visualizing outputs of logic gates. We organized its architecture around four layers independent and superposed as shown in Figure 6.

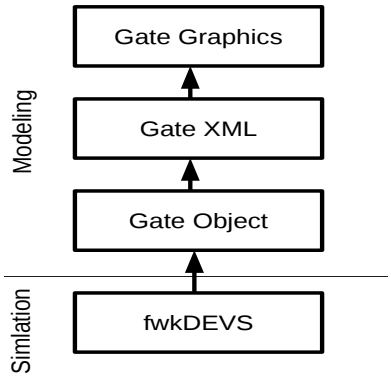


Figure 6. The four layers of GDEVSLogic.

The choice of this architecture consists on the well-known architecture in software engineering : the n-layered architecture [6]. Such an architecture provides for designers a clear and modular architecture, easy maintainability in order to fix errors and extensibility to add packages of new functionalities. Thus, each layer from our architecture assumes a limited functionalities and communicates only with the higher layer in order to to have a less coupling.

[1]In a given DEVS/GDEVS network, a model may influences another by transmitting it output via coupling. So for any model there are two coupling functions that define the sets of model- influencers and influencees.

Gate Graphics

This layer provides a user-graphical interface to design logic gates. We used the JavaFX technology which is a set of graphic software and media of Java SE 8 © and which allows the developers to conceive (design), to create, to test, to debug and to deploy customer applications rich which work in a uniform way in various platforms. The JavaFX applications may use Java API (Application Programming Interface) libraries to reach capacities of the native system and connect to the middle-ware applications.

After launching the tool will display a graphical interface (see Figure 7), where at left the user will find libraries of basic and defined gates that may reuse. Before beginning the modeling of the new compound, the user must specify the name and number of inports and outports of the model to create by going to File → New → Gate. Then after confirmation, a box appears with at left the inports and at right the outports.

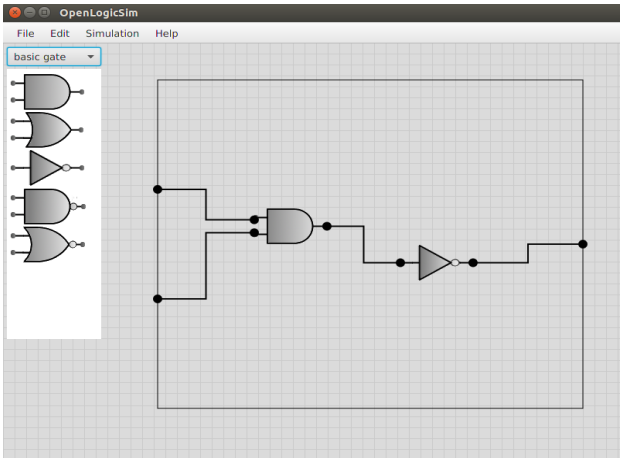


Figure 7. A screen copy from GDEVSLogic interface.

From this interface, the end-user may drag and drop existing logic gates from the library, in order to reuse them. Then, he couples dragged logic gates with connectors in drag and drop mode.

Moreover, the user-interface allows the end-user to save the circuit for further reuses. This is the role of the XML layer which ensures the functionalities of check and storage.

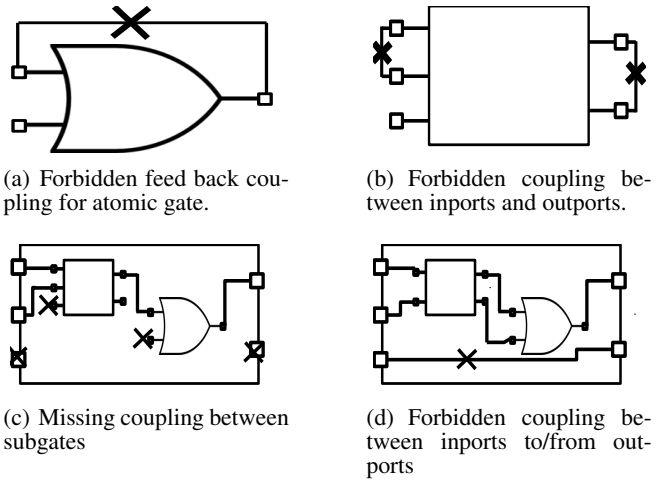
Gate XML

The modeled circuit is saved as an XML file (eXtensible Markup Language) that is somehow enhanced HTML (Hypertext Markup Language) to define new tags. It is indeed a language to format documents using tags (markup). In fact XML tags describe the content rather than the presentation. In our architecture, the XML file contains the name of the electronic circuit and the number of inports and outports, the list of reused subgates and the three different couplings (internal, external input and external output couplings). For example the tag <Internal Coupling> contains the name of the two subgate ports that connects:

```
<?xml version="1.0" encoding="UTF-8"?>
...
```

```
<Internal coupling> beginning tag
  <From>
  <Name>name of the first gate</Name>
  <Port>type of port, inport or
   outport</Port>
  </From>
  <To>
  <Name>name of the second gate</Name>
  <Port>type of port, inport or
   outport</Port>
  </To>
 </Internal coupling> end tag
```

The layer Gate XML consists on the JDOM (Java Document Object Model) API that allows parsing trees of objects into XML documents. This parser allows as well the manipulation of XML documents to add, delete, and update the elements (nodes) of documents. The main advantage is that JDOM object was specialized in our architecture to keep the structure of a given gate and then check the designed tree according to the GDEVS coupled definition in order to detect errors to fix. For that purpose, we construct a base of errors that end-users may make while they design the circuit. Figure 8 illustrates some frequent and infrequent errors that occur for composite and basic logic gates.



(a) Forbidden feed back coupling for atomic gate.

(b) Forbidden coupling between inports and outports.

(c) Missing coupling between subgates

(d) Forbidden coupling between inports to/from outports

Figure 8. Some errors occurring in designing circuit.

Note that in DEVS and GDEVS, connections between ports are unidirectional. Whereas at the layer Gate Graphics, connectors have no direction, that means an input port of a given subgate may be linked to an output port of another gate, in graphics. Obviously, this fact saves the end-users from additional errors but the simulation should take care and should invert such connectors.

However, we decide that the layer Gate XML carries out the fixture of such connections that are not mistakes but a useful facility of design for end-users; even if they provoke simulation errors. Thus, this layer should identify such connections and inverts them. At the end, the XML file of the given gate keeps a structure and a coupling valid of the graphical gate.

Gate Object

The layer Gate Object is the last layer of modeling and the starting point of simulation. In fact, the simulation is based on Java classes generated from the XML file that keeps the structure of the gate to simulate. This layer holds a second parser which transforms the Gate XML file into two Java classes ready to be compiled and plugged to the simulator fwkDEVS.

The first class keeps the structure of the designed gate; and a second class that defines the network to simulate in which each input port and each output port of the gate designed is coupled to a generator and a transducer respectively. The generator classes load edited input files from the tool and transducers classes write output signals in readable files. Finally, the end-user defines the simulation duration and may start it.

Note that the tool compiles the two generated classes thanks to the Java compiler. Then the simulator loads output class files in order to run the simulation and to print output files.

Moreover the layer Gate Object allows end-users with knowledge on programming, designing large scale of logic gates and so overcoming the limitation of the graphical design which takes into account only small-scale and simple logic gates. For that, the designer should extend the class `DEVSGate`, then he should define the repetitive statements in order to instantiate subgates and to make the coupling between them. Note that such a gate may be plugged in our tool and be reused by end-users as a black box to define new circuits.

Recall that the transformations of the circuit from the graphics until the corresponding code are free from errors. The designer may check at each step (Gates XML and Object) that the obtained specifications correspond to the initial circuit, by activating assertions and carrying out them in the background. For example, while the end-user reuses a set of gates and couples them, the tool may check (at layers Gate XML and Gate Object) that the reused gates are declared and the made coupling is in respect with that shown at the layer Gate Graphics.

CONCLUSION

In this paper, we extend our previous work to model and simulate electronic circuits based on GDEVS of first order, to GDEVS of second order. This new approach allows representing more faithfully the trajectory of signals given to and output from the electronic circuit. Note that the use of a second degree equation solver allows the designers to compute the right time at which input signals may cross. An important feature that reinforce the mathematical feature of the approach.

In order to make the GDEVS modeling and simulation of electronic circuits accessible to end-users, we developed a software tool GDEVS Logic ©. It is based on a layered architecture that consist on Graphics, XML and Object code layers, each one holds a limited functionalities. The passage from a layer to another generates a specification which is conform to the designed circuit. At the end, the simulation of circuit is carried out and results may viewed under curves.

The current approach uses the transport delay in which only output signals are delayed comparing to input ones. However, GDEVS allows us to customize the delay to inertial one by designing the right functions and so ignore shorter input signals.

One of the main future works that we aim is to supply the proposed approach with a formal verification technique. Another way to trust the simulation results.

REFERENCES

1. Chen, B., and Vangheluwe, H. Symbolic flattening of DEVS models. In *Proceedings of the 2010 Summer Computer Simulation Conference. Society for Computer Simulation International*, Society for Computer Simulation International (Orlando, USA, 2010), 209–218.
2. Giambiasi, N., and Carmona, J. Generalized discrete event abstraction of continuous systems: GDEVS formalism. *Simulation Modelling Practice and Theory 14* (2006), 47–70.
3. Giambiasi, N., Escudé, B., and Ghosh, S. GDEVS: A generalized discrete event specification for accurate modeling of dynamic systems. *Simulation: Transactions of the Society for Modeling and Simulation International 17*, 3 (2000), 120–134.
4. Hamri, M. fwkDEVS: A DEVS/GDEVS modeling and simulation framework, 2016. http://www.lsis.org/hamria/fwkdevs.html.
5. Hamri, M., Giambiasi, N., and Naamane, A. Generalized discrete events for accurate modeling and simulation of logic gates. *Concepts and Methodologies for Modeling and Simulation Part III* (2015), 257–272.
6. Microsoft Patterns & Practices Team. *Application Architecture Guide*, 2nd ed. 2009.
7. Wainer, G., Daicz, S., and Troccoli, A. Experiences in modeling and simulation of computer architectures in DEVS. *Transactions 18*, 4 (2001).
8. Zeigler, B. P., Praehofer, H., and Kim, T. G. *Theory of Modeling and Simulation*. Academic Press, 2000.

APPENDIX

GDEVSLogic SCREEN COPIES

The User Interface of GDEVSLogic

See Figure 9.

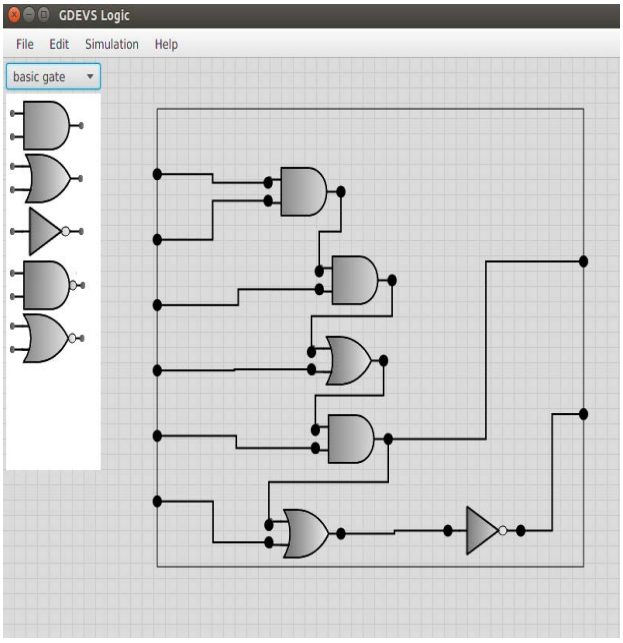


Figure 9. An example of circuit using GDEVSLogic interface.

Simulation Results of GDEVSLogic

See Figure 10.

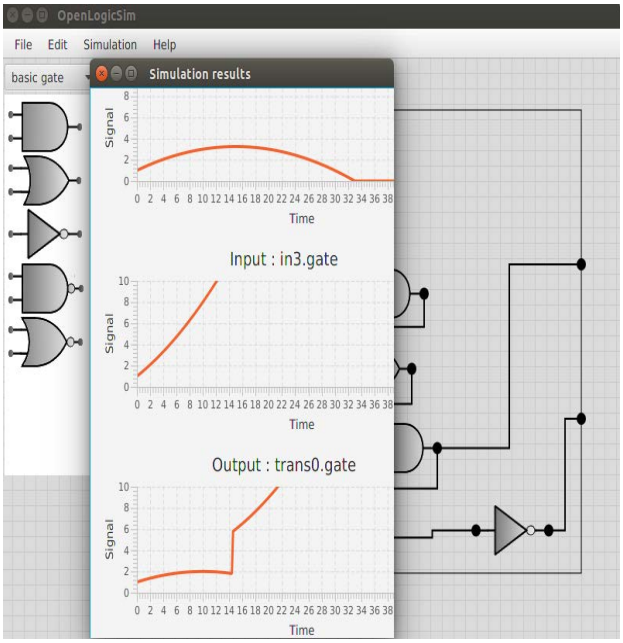


Figure 10. Simulation results using GDEVSLogic interface.