Proceedings of the 2003 Winter Simulation Conference
S. Chick, P. J. Sánchez, D. Ferrin, and D. J. Morrice, eds.

MODELS FOR CONTINOUS AND HYBRID SYSTEM SIMULATION

Mariana C. D'Abreu

Computer Science Department
Universidad de Buenos Aires
Planta Baja, Pabellón I, Ciudad Universitaria
(1428) Buenos Aires, ARGENTINA

Gabriel A. Wainer

Dept. of Systems and Computer Engineering
Carleton University
1125 Colonel By Drive. 4456 Mackenzie Bldg.
Ottawa, ON, K1S 5B6, CANADA

ABSTRACT

The DEVS formalism was defined as a method for modeling and discrete event systems. DEVS theory evolved and it was recently upgraded in order to permit modeling of continuous and hybrid systems. Here, we present a first experience on the use of two of the existing methods for defining continuous variable DEVS models (namely, the QDEVS and the GDEVS formalisms), to develop continuous and hybrid systems simulations. We show how to model these dynamic systems under the discrete event abstraction. Examples of model simulations with their execution results are included. An experimental analysis on quantization methods within models is also presented.

1 INTRODUCTION

Complex systems analysis has usually been tackled using different mathematical formalisms, Partial Differential Equations (PDE) being one of the preferred tools of choice (Taylor, 1996). In most complex systems, solutions to these equations are very difficult or impossible to find. A variety of numerical methods find approximate solutions to these equations, being successful in studying many different phenomena. The appearance of digital computers allowed enhancing previously existing numerical methods. Simulation-based approaches succeeded in providing a means of analyzing particular problems (instead of the general solutions obtained by solving PDEs).

Simulation of continuous systems on digital computers requires discretization. Classical methods as Euler, Runge-Kutta, Adams, etc., are based on discretization of time resulting in a discrete time simulation model (Press et al. 1986). Instead, methods like DEVS (Discrete EVent Specification) formalism (Zeigler et al. 2000) were built in order to allow the specification of discrete event models. The DEVS formalism was defined as a method for modeling and discrete event systems. DEVS provides means to handle explicit time, and to define complex models in a hierarchical modular fashion.

DEVS theory evolved and it was recently upgraded in order to permit modeling of continuous and hybrid systems. GDEVS (Generalized Discrete EVent Specification) (Giambiasi et al. 2000) is a generalization of constant input-output trajectories beyond DEVS abstraction; under this formalism, trajectories are organized through piecewise polynomial segments. This presents some advantages, including greater accuracy in modeling continuous systems and the ability to develop a uniform approach to model hybrid systems, i.e. composed of both continuous and discrete components. Another approach to solve these problems under the DEVS formalism is based on state variable quantization (Zeigler et al. 1998). The idea beyond this method is to provide quantization of the state variables obtaining a discrete event approximation of the continuous system. This formalism is known as Q-DEVS and quantization is done using a piecewise constant function.

In the long term, we want to attack the development of hybrid systems based on the DEVS formalism and its extensions, building libraries to make easy to use components developed on top of DEVS modeling tools. In this article we show our first results in this sense. We present the implementation of a library of Bond Graphs (Cellier 1991) based on GDEVS. Likewise, we present other components often used in continuous systems using QDEVS. One of the benefits is that for a given accuracy, the number of transitions can be reduced, decreasing the execution time of simulations. Discrete time models can be simulated under discrete event paradigm, thus allowing the development of a simulation environment for complex systems, modeled as hybrid systems, where all paradigms merge together (continuous time, discrete time, discrete event).

The experience was developed using the CD++ toolkit (Wainer 2002), a modeling and simulation framework that was developed in order to implement the theoretical concepts specified by the DEVS formalism. A hierarchy of models is introduced in order to allow a modular and simple specification of dynamic and hybrid systems. Some examples are included in order to show results of models execution. An analysis of Q-DEVS models is also pre-

sented. A set of test cases are specified and executed based on uniform quantization function equipped with hysteresis. These methods may result in poor stability and convergence. In addition, they can be inaccurate. Nevertheless, the results obtained in this experience are promising. In this first step of our research, we focus in software implementation issues. Several theoretical problems remain open, and they will be addressed in future work.

2 BACKGROUND

DEVS (Discrete EVents Systems Specification) (Zeigler et al. 2000) defines a way to specify systems whose states change either upon the reception of an input event or due to the expiration of a time delay. It allows hierarchical decomposition of the model by defining a way to couple DEVS models. A DEVS atomic model is described as:

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

X is the *set of external events*
Y is the *set of internal events*
S is the *set of sequential states*
δ_{ext}: Q x $X \rightarrow$ S is the *external state transition function*
δ_{int}: S $\rightarrow$ S is the *internal state transition function*
λ: S $\rightarrow Y$ is the *output function*
ta: S $\rightarrow R_{0+} \cup \infty$ is the *time advance function*

A DEVS coupled model is composed of several atomic or coupled submodels. It is formally defined by:

$$CM = < X, Y, D, \{M_i\}, \{I_i\}, \{Z_{ij}\}, select >$$

D is a *set of components*; for each i in D,
M_i is a *basic DEVS component*; for each i in D,
I_i is the *set of influencees* of i; for each j in I_i,
$Z_{i,j}$ is the *i-to-j output-input translation* function
select is the *tiebreaker* function.

Continuous time ODE systems with initial conditions have traditionally been simulated by discretizing the time domain, and solving the ODE over each discrete time interval. Recently quantized DEVS models (Zeigler 1998) permitted to solve this problem using a different approach, depicted in Figure 1. A curve is represented by the crossing of an equal spaced set of boundaries, separated by a *quantum* size. A *quantizer*, checks for boundary crossing whenever a change in a model takes place. Only when a crossing occurs, a new value is sent to the receiver. This operation reduces substantially the frequency of message updates.

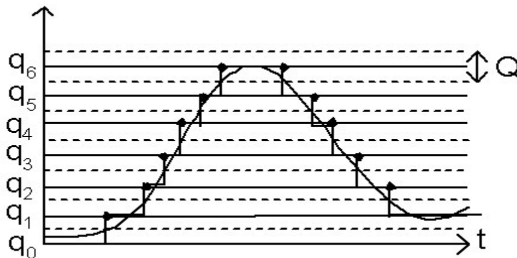


Figure 1: Signal Quantization

This approach requires a fundamental shift in thinking about the system as a whole. Instead of determining what value a dependant variable will have (it's state) at a given time, we must determine at what time a dependant variable will enter a given state, namely the state above or below it's current state. This approach may yield results as accurate as a discrete time approach.

Another approach recently applied to solve this problem is the GDEVS (Generalized Discrete EVent Specification) formalism (Giambiasi et al. 2000). GDEVS uses of polynomials of arbitrary degree to represent the piecewise input-output trajectories of a discrete event model. GDEVS adopted an approach based on a new definition of the concept of event. The target real-world system is modeled through piecewise polynomial segments translated into piecewise constant trajectories. A coefficient event is considered as an instantaneous event.

For example, let us consider a piecewise linear trajectory $w<t0;tn> \rightarrow A$ as a trajectory on a continuous time base, characterized as follows: there is a finite set of instants $\{t_0, t_1, \dots, t_n\}$ associated with constant pairs $(a_i; b_i)$ such that $\forall t \in <t_i; t_j>$, $w(t) = a_i t + b_i$, and $w<t_0; t_n> = w<t_0;t_1>*w<t_1;t_2>*\dots*w<t_{n-1};t_n>$ (where * represents the operator left concatenation of segments). This is exemplified in Figure 2, which describes the use of piecewise linear approximations of the continuous segment in Figure 2a, while Figure 2c represents a discrete event abstraction of order 1 under GDEVS with coefficient events.

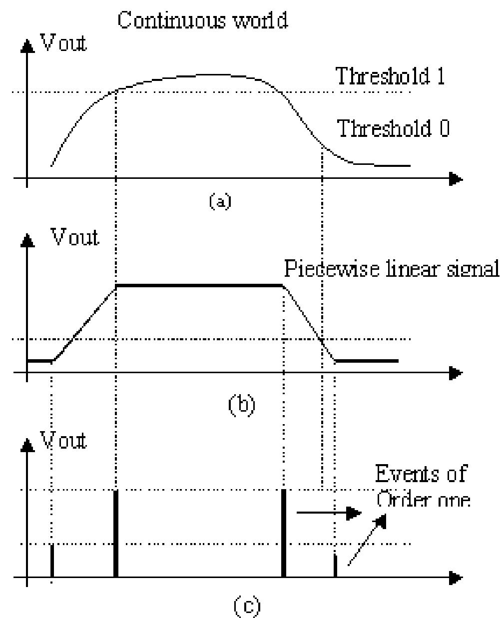
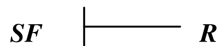


Figure 2: GDEVS Approximation

In order to build our library, we started by creating models of continuous systems to be integrated with existing DEVS models. To present a proof of concept, we fo-

cused in building a library for *Bond-Graphs* (Cellier 1991) technique. One of the reasons for choosing Bond-Graphs was the direct relation existing between the basic electrical elements integrating a circuit and the primitive Bond-Graph components. In Bond-Graphs, physical processes are represented as vertices in a directed graph and the edges represent the ideal exchange of energy between the vertices (Cellier 1991, Samantaray 2003). The *energy* (or its time derivative, the *power*), is the fundamental quantity exchanged between elements of the system. Power is the product of *flow* and *effort* (in translation mechanics, power is the product of *force* and *velocity*; in electrical systems, it is the product of *voltage* and *current*; in any system, a generalized flow and effort variable could be defined). In Bond-Graphs, flow of energy is represented by bonds, and elements exchange effort and flow through them. The exchange of power is assumed to occur through abstract entities called *energy port*; this way, the concepts of One-port and Two-Port elements are defined. One-port element, is a component that has associated one energy port represented with a bond. Two-Port elements are those that have two energy ports, represented with two bonds. Interactions between components are also restricted. Connectors implement constrained exchanges between elements. The 1-junction represents interactions between components connected serially.

As we represent the exchange of power between elements, a concept to understand how information flows between components is *causality*. No component can determine both the power variables, effort and flow. Given a pair of elements connected through a bond, causality determines which of the components causes the flow information and which causes the effort. For example, in a current source connected to a resistor, the source dictates the flow while the resistor dictates the effort. This is graphically represented as:



CD++ (Wainer 2002) is a modeling and simulation tool based on implementing DEVS theory. The tool provides a specification language that allows describing model coupling; additionally, atomic models can be developed using C++. CD++ was built as a hierarchy of classes in C++, each corresponding to a simulation entity using the basic concepts defined in (Zeigler et al. 2000). The *Atomic* class implements the behavior of an atomic component, whereas the *Coupled* class implements the mechanisms of a coupled model. We used CD++ to build a Bond-Graph library as a set of independent models in which the components are developed using GDEVS. All the components we modeled were defined using functions of order one, but higher accuracy could be achieved increasing polynomial degree. We also built a set of components using Q-DEVS. We will describe these models in the following sections.

3 A BOND-GRAPH LIBRARY IN CD++

We developed a library of Bond-Graphs on top of the DEVS formalism, and this package permits to model and simulate continuous systems within different contexts. We used the CD++ toolkit to build a library for electrical package. This library was defined using the conceptual specifications of GDEVS. Having a library of models provide a modular approach to build and simulate electrical circuits thus allowing code reusability. The use of GDEVS allows us to easily integrate continuous models with discrete event models using DEVS. All Bond-Graph primitive elements were built as atomic GDEVS models with polynomial functions of degree one. Extensions to the CD++ toolkit were introduced in order to support this new approach.

Every primitive Bond-Graph element (port component) defines one or more equations that involve the flow and/or effort variable values received by the bonds connected to it. Bonds are two-signal connections (effort and flow) that have opposite directions. Passive elements like those that Capacitors, Resistances and Inductors have a power direction pointing inwards, on the other hand, active components like Source have the power pointing outwards. This signal direction determines the bond causality, having some component equations putting demands on it.

The elements of the library were developed according a hierarchy of classes derived from DEVS atomic models, the components of this hierarchy are described bellow:

- *BG*: an abstract model, base for all the primitive Bond-Graph elements. It introduces functionality to add bonds to the components.
- *Resistance*: it calculates the effort value according the resistance equation: *effort* = *R . flow*. Here, R is the Resistance constant. The flow received by is automatically processed to obtain the effort value, which is informed to its adjacency. The time instants of new input arrival, t_1, t_2, ..., t_n, are associated to tuples (a_i,b_i), which values correspond to the coefficients used to approximate the effort curve by the polynomial function: $\text{effort}(t) = a_i\, t + b_i \quad \forall\, t <t_i,t_j>$. The model internal transition, implements the polynomial approximation of the continuous curve (effort in this case). This behavior is common to all the GDEVS models developed, where curve approximation is done using a polynomial function of order one.
- *Capacitor*: it models the static relation existing between effort and displacement. Storage elements as Capacitors impose a preferred causality. The equation defined by the Capacitor element can be expressed in two forms:

$$effort = \frac{1}{C}\int_{-\infty}^{t} flow\, dt \quad \text{or} \quad flow = C\,\frac{deffort}{dt}$$

where C is the Capacitor constant. We implemented the first equation, which is introduced in Figure 3. The implementation of the remaining components here described was defined following a similar approach than the one showed in following.

```
class Capacitor : public BG {
  public:
     Capacitor(const string &name =
                CAPACITOR_CLASS_NAME );
       ~Capacitor();
. . .
  private:
    FlowInBond &bond;  // Flow input
    RealValue  a0, a1; //input level and slope
    VList     *yout;   // output value (effort)
    RealValue   c;     // capacitor load
    RealValue   C;     // capacitor constant
    RealValue   time;  // time accounting
    RealValue   dt;    // delta
};

External transition {
// Calculates load as integral of the inputflow
  if ( state() == active ) {
   // load calculated for the duration of state
       a0 = a0 - a1 * dt;
       c = c-(a1/2 * pow( dt, 2 ) + a0 * dt );
  }
  // time since last transition
  elapsedTime=msg.time().asMsecs()-time;
  // calculates load value

c = c+a1/2*pow(elapsedTime,2)+a0*elapsedTime;
   VList *list = (VList *)msg.getValue();
  // considersinput event coefficients
  a0 = list->elementAt(1)->toReal();
  a1 = list->elementAt( 2 )->toReal();
  a0 /= C;    a1 /= C;    // Capacitor constant
  // sets coefficient of next output event
  yout->updElementAtPos( 1, c );
  yout->updElementAtPos( 2, a1/2 * dt + a0 );
  time = msg.time().asMsecs();
  holdIn( active, Time::Zero );
}

Internal transition {
// approximates load using order 1 polynomial.
  if ( a1 != 0 ) {
  // next state calculated using coefficients
       c = c + a1/2 * pow( dt, 2 ) + a0 * dt;
       a0 = a1 * dt + a0;
  // coefficient values to send when dt elapsed
   yout->updElementAtPos( 1, c );
       yout->updElementAtPos( 2, a1/2 * dt + a0
);
       holdIn( active, Time( dt ) );     }
  else {
           passivate(); // slope is null
  }
}

Output function {
 sendOutput(msg.time(),*bond.outputPort(),
*yout);
}
```

Figure 3: Implementation of Capacitor element in CD++

As we can see, when flow arrives at the component, an external transition function is activated, and the flow is integrated in order to calculate the effort value, which is send to the rest of the system through the effort port. The external transition function calculates the effort value as the integral of the input flow data, generating the Capacitor's load. We can see the implementation of the continuous curve approximation using a polynomial function of order one. If the flow input arrives during an active state, the value is computed according to the elapsed time since the last internal transition function. An internal transition is immediately scheduled, which will be in charge of computing the next state using a polynomial of order 1. Before executing the internal transition function, the output function transmits the previously computed value *yout*.

- *EffortSource and FlowSource*: these components generate signal values according to an emission frequency. EffortSource sends the effort though the output port, while the FlowSource sends the flow value. Several signals were implemented in order to provide a set of functions to use in different contexts: *Constant, Step, Ramp, Sine, Expsine, Exponential, Pulse*.
- *Inductor*: it defines the static relation existing between flow and momentum. We used the following preferred equation: $flow = \frac{1}{L}\int_{-\infty}^{t} effort\ dt$. The L value corresponds to Inertial constant. The model transition functions are similar to those listed for Capacitor model, but in this case, the Inductor load (flow) is calculated as the integral of effort value.
- *Transformer*: this model conserves power and transmits the factors with the proper scaling defined by the transformer modulus. The modulus equation defines the following relations: $f_j = r\, f_i$, and $e_j = (1/r)\ e_i$, where r is the transformer modulus, (e_i,f_i) and (e_j,f_j) are the (*effort,flow*) values transported by $bond_i$ and $bond_j$ attached to the component, respectively. New input effort data arriving at the component is processed in the external function and used to compute the outgoing effort according the modulus relation. As this element has two bonds connected to it, both output effort and flow values must be calculated by the model. This way, internal function approximates both values with polynomial functions.
- *Gyrator*: this model establishes the relationship between flow to effort and effort to flow, keeping the power unchanged. The relations are defined by: $e_j = \mu\, f_i$, $e_i = \mu\, f_j$ where μ is the gyrator modulus, (e_i,f_i) and (e_j,f_j) are the (*effort,flow*) values transported by $bond_i$ and $bond_j$ attached to the component, respectively. In a 0-junction component, all the *flows* sum up zero; in 1-junctions, the

values of all *efforts* must sum zero. Figure 4 presents a graphical representation of junctions.

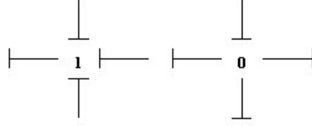


Figure 4: Graphical representation of junctions

As it can be seen in the figure, junctions are connected to several bonds respecting causality restrictions. The sign of the flows and efforts within the component are defined by the power direction, which is one of the bond's attributes. The 0-junction (1-junction) model implemented in the library, processes the arrival of new effort (flow) data in the model's external function, sending for all the output effort (flow) ports the received value. On the other hand, the arrival of new flow (effort) by one of the bonds, generates the recalculation of the equation. Once value is recalculated, flow (effort) is sent by output port.

- *Bond*: this is not a Bond-Graph primitive element, but it was included in the library in order to implement the functionality beyond component connections. Every bond is associated with an input port and an output port, which must transport the effort and flow variables between components. Attributes specify the power direction and causality restrictions. Bonds connect the different components allowing the exchange of power between them. New input effort or flow data arriving at a component, serves for the calculation of new output data using the input-output mathematical relationships defined by every component through the equations. Depending on the element type (One-port, Two-Port or Junction), it could have associated one or more bonds. The exchange of data within every bond is done through the *ciPort* and *coPort* ports defined in the class Bond.

The complete hierarchy of Bond-Graph models integrating the library is shown in in the Appendix.

4 MODEL EXECUTION EXAMPLES

Besides the Bond-Graph library presented in the previous section, we included a number of components based on the QDEVS formalism. As with GDEVS, this technique presents some advantages over classical continuous methods based on discretization of time. Here we present two quantizers and one integrator. Both quantizers were implemented in order to provide a function that converts continuous trajectories to piecewise constant segments. Obn the other hand, the Integrator model implements the functionality needed to integrate its input.

The Quantizer model provides the representation of output trajectories as piecewise constant functions through the quantization function. Two types of quantizers were implemented: *uniform* and *non-uniform* (*intervals)* quantizer. *Uniform* quantizers (depicted in Figure 5) specifies a *quantum* size that defines the length of all the intervals. In contrast, *Intervals* (not uniform) quantizer, permits to set different interval lengths according the following restriction:

$$Quantization_domain = \bigcup_i r_i$$

where r_i belongs to quantization intervals verifying $r_{i-1} < r_i$

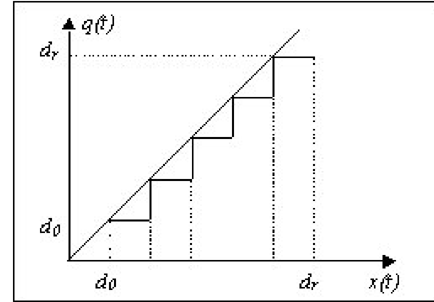


Figure 5: Uniform Quantization

A second model uses quantization with hysteresis (Kofman and Junco 2001), as shown in Figure 6. The inclusion of hysteresis in the function allows the discrete-event simulation of the continuous system. Let $D = \{d_0, d_1,..., d_n\}$ be a set of numbers where $d_i \in \Re$, $d_{i-1} < d_i$ with $1 \le i \le r$ and let $x \in \Omega$ be a continuous trajectory where $x : \Re \rightarrow \Re$. Let $b : \Omega \rightarrow \Omega$ be a mapping and q the trajectory defined by $q = b(x)$. A fundamental property that function b must establish is given by the following inequality: $d_0 \le x(t) \le d_r \Rightarrow |q(t) - x(t)| = |b(x(t)) - x(t)| \le max_{1 \le i \le r}(d_i - d_{i-1}, \varepsilon)$ where ε is the size of the hysteresis window.

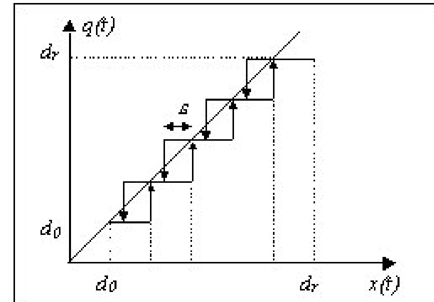


Figure 6: Quantization with hysteresis

Finally, we built an integrator as a DEVS model. The model outputs the integrated value of its input. It implements the Euler integration algorithm, a first order method that provides numerical solution to integral calculus. The algorithm is defined by the following iteration step:

$$x^*_{n+1} = x^*_n + hx'(n)$$

where x^* is the approximation of function x and x' is given by the following equation

$$x'(t) \approx \frac{x(t+h) - x(t)}{h}$$

5 MODEL EXECUTION EXAMPLES

We show how we used the libraries to execute some examples of application. We first will introduce an electrical circuit (presented in Figure 7), which is composed of components connected in serial and parallel. The intention is to measure the current value of the circuit according the evolution of time (Banerjee 2003).

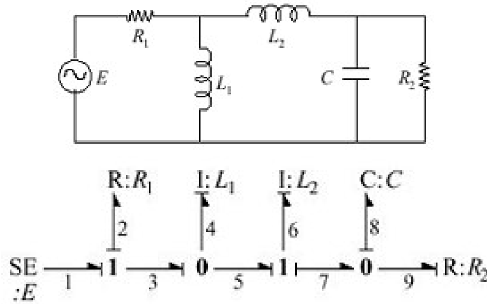


Figure 7: Circuit diagram and Bond-Graph

In order to simulate the circuit within the CD++ toolkit, the components in the diagram had to be replaced by the corresponding atomic models developed in the Bond-Graph library, explained in the previous section. All the components were connected using input/output (effort/flow) ports, according the causality defined by every element, generating a GDEVS coupled model. The structure of the coupled model associated to the circuit in Figure 8 is shown below.

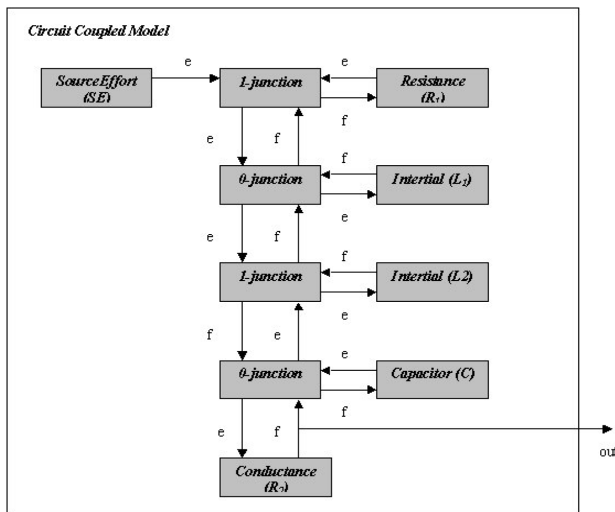


Figure 8: DEVS coupled model associated to the circuit

As it can be seen in Figure 8, every Bond-Graph element was replaced with its functionally equivalent library component, a DEVS atomic model. The components have been associated to the following parameters for simulation:

- period: 1 ms.
- Resistance (R_1)=1
- Inductors: $L_1 = 48$; $L_2 = 48$.
- Capacitance: C = 65.
- Conductance: $R_2 = 0.001$
- EffortSource: emits a pulse with a period of 2500 ms and duration of 2 ms. Pulse amplitude= 220 V.

Figure 9 shows the simulation results for this example.

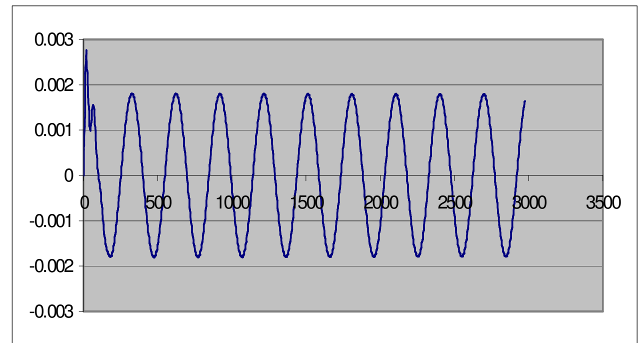


Figure 9: Circuit current

Another example was based on building a mechanical system representing a mass-spring-damper system (Santamaray 2003), presented in Figure 10. The objective of this simulation is to observe the movements of the mass M when we subject the system to the application of regular forces. The translation from the mechanical system to CD++ components was done as in the previous example. Every Bond-Graph element was replaced by its related atomic model, which, in turn, was connected to its adjacency trough the ports, generating a coupled model.

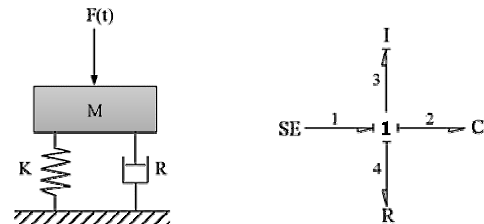


Figure 10: Mechanical Circuit and Bond-Graph

Two cases where tested, modifying the frequency of models, in order to measure the time consumed by the simulation toolkit in every case.

- M: mass of the element (L = 40).
- K: stiffness of damper spring (capacitance constant C=2)

- R: resistance coefficient (resistance constant R=1.5)
- EffortSource: emits a pulse with a period of 200msec and duration of 2 ms. Pulse amplitude was set to 1 and model frequency to 1/10 ms.

Finally, different tests were done to compare the results of applying uniform quantization method with and without hysteresis technique. To do so, models Quantizer and HQ (quantizer with hysteresis) were tested. The following table presents the cases executed and analyzed:

Table 1 – Test cases

Function	*Interval*	*Q*	*Hyst.*
$f(x) = \text{sine}(x)$	$[0,2\pi]$	0.1	0.1
$f(x) = 0.9–10^5.(x \bmod 2)$	[1,10001]	0.001	0.001
$f(x) = x$	[0,1]	0.1	0.1
$f(x) = 10^{-2} x^2 + 10^{-2} x + 1$	[-100,100]	0.1	0.1
$f(x) = (1/ x^2) . \text{cosine}(x)$	[0.0001,1]	0.001	0.001
$f(x) = - \log x$	[0.1,1.1]	0.01	0.01
$f(x) = 1/x$	[0,1000]	0.0001	0.0001
$f(x) = (-1)^x . 1/(\ln(x))^4$	[2,100]	0.0001	0.0001

The CD++ simulator evolves by message passing between the models. In order to analyze the execution times, both models were tested against the same input function in every case. Execution results are shown in Figure 11.

The values obtained represent the percentage of output messages sent by the models in relation to the input messages received. Message reduction in case (1) is less significant than in the rest of cases because of the characteristic of the function considered. The quantity of points evaluated over the interval, which determines the distance between them, makes no substantial difference in using one model over the other. A similar situation is presented in case (3). Here the function is strictly growing, making no sense the use of hysteresis technique.

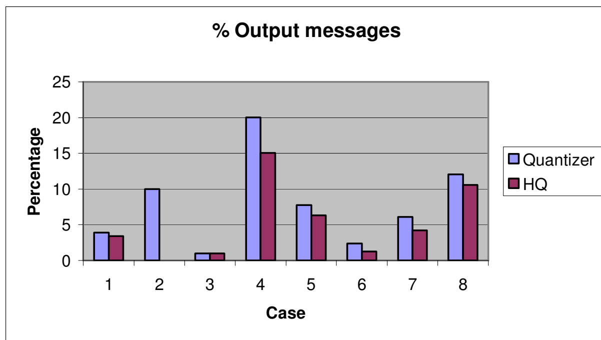


Figure 11: Percentage of output messages

Case (2) shows a situation where the use of hysteresis for quantization gives good results. Here, the function oscillates between two values that are separated by a distance less than hysteresis window size. The number of output messages generated by *Quantizer* corresponds to 100%, meaning that every message arrived to the model was emitted. The percentage of output messages produced by *HQ* model corresponds to 0.02%, producing a significant reduction on the number of messages involved in the simulation. In case (4), a reduction of 25% on the quantity of output messages was obtained by model *HQ* in reference to the number generated by *Quantizer*. Here, the reduction is consequence of applying quantization with hysteresis to values belonging to the first half of the evaluation interval, where function decreases. Case (5) presents a similar situation that case (4) but here, the function values that make significant the reduction of output messages, are those belonging to the last part of the evaluation interval. This is because function decreases faster as evaluation points get bigger. A reduction of 50% of output messages was obtained in case (6) and 30% in case (7) by model *HQ*. In both cases, inputs correspond to functions that decrease across the entire interval.

The size of quantum chosen and the quantity of points evaluated help to give significant results, making distance between values less than hysteresis window size in a considerably number of points. That reduction shows that little variations on the input values are not taken into account by model *HQ*, decreasing significantly the number of messages emitted. Case (8) presents a function that oscillates between a range of values that get closer by time passes. Here, function period length is greater than in case (2), making model *HQ* increment the number of messages emitted compared to case (2). Distance between values evaluated, greater to hysteresis window size for most of the points belonging to the first part of evaluation interval, also contributes on making results not so good as in case (2).

The error obtained depends on the quantum size and the quantization function. This behavior can be defined according the following equation:

$$e_q = \frac{\sum_{i=1}^{N} | x_q - x |}{| x | .N}$$

where x_q is the quantized value, x is the original value and N is the number of points considered. Results are shown in Figure 12. It can be seen, analyzing graphic results, that most of the cases present similar results. The error introduced by quantization process in each case differs in unsubstantial way between models.

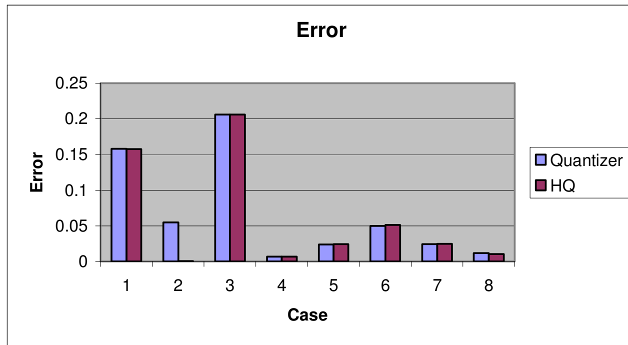


Figure 12: Quantization error

The exception is case (2), it is consequence of using hysteresis to quantize functions with slight oscillations. In this case, the output generated by *HQ* model is not altered by variations on input signal if they are less than hysteresis window size. The window can be considered as a variation tolerance threshold. That is not the case of *Quantizer*, which only considers quantization interval values, introducing significant error.

6 CONCLUSION

A hierarchy of models was developed within CD++ to give support in the simulation of continuous and hybrid systems. A GDEVS base and *Bond-Graph* elements were implemented allowing the simulation of dynamic systems in a context-independent way. The use of quantization methods was also presented. Tests were executed and results were analyzed in order to determine when these methods could yield the appropriate approach. The use of hysteresis technique in quantization function decreases, in most of the cases, the number of messages involved during simulation in comparison with classical quantization. In some cases, this reduction can be very significant. The error introduced by hysteresis quantization is almost equal to that produced by classical quantization, making the technique a good alternative to accelerate simulation without obtaining greater error.

Different open topics must be considered for future research in this are. First, an exhaustive comparison between the simulation models and the corresponding analytical solutions must be faced. Model complexity must be considered when using polynomial approximations. Stability and convergence properties must be analyzed. Using these approaches, we can benefit of better performance for a given accuracy, which decreases the execution time of simulations. Discrete time models can be simulated under a discrete event paradigm, thus allowing the development of a simulation environment for complex systems, modeled as hybrid systems, where all paradigms merge together (continuous time, discrete time, discrete event).

ACKNOWLEDGMENTS

This work was partially funded by the Natural Sciences and Engineering Research Council of Canada (NSERC) and the Institute of Robotics and Intelligent Systems (IRIS, Canada). I would like to thank to the anonymous referees and their comments to this article.

REFERENCES

Banerjee, S. 2003. Dynamics of Physical Systems – The Language of Bond Graphs. Available online via http://www.ee.iitkgp.ernet.in/~soumitro/dops/chap4.pdf [Accessed April 6 2003].

Cellier, F. 1991. *Continuous System Modeling*. Springer-Verlag, New York.

Giambiasi, N.; Escude, B.; Ghosh, S. 2000. GDEVS: A Generalized Discrete Event Specification for Accurate Modeling of Dynamic systems. *Transactions of the SCS,* 17(3) pp. 120-134.

Kofman, E.; Junco, S. 2001. Quantized State Systems. A DEVS Approach for Continuous Systems Simulation. *Transactions of the SCS*, 18(3), pp. 123-132.

Press, W.H.; Flannery B.P.; Teukolsky, S.A.; Vetterling, W.T. 1986. *Numerical Recipes*. Cambridge University Press, Cambridge

Samantaray, A. 2003. About Bond Graph–The system modeling world [online]. Available online via http://www.bondgraphs.com/about.html [Accessed April 6 2003].

Taylor, M. 1996. *Partial Differential Equations: Basic Theory*. Springer Verlag, NY.

Wainer, G. 2002. CD++: a toolkit to define discrete-event models. *Software, Practice and Experience*. Wiley. Vol. 32, No. 3. pp. 1261-1306.

Zeigler, B. DEVS Theory of Quantization. 1998. *DARPA Contract N6133997K-007: ECE Dept., University of Arizona*, Tucson, AZ.

Zeigler, B.; Kim T.; Praehofer, H. 2000. *Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems*. Academic Press.

Zeigler, B.P., Cho, H.; Lee, J.; Sarjoughian, H. 1998. The DEVS/HLA Distributed Simulation Environment and Its Support for Predictive Filtering. *DARPA Contract N6133997K-0007: ECE Dept., University of Arizona*, Tucson, AZ.

AUTHOR BIOGRAPHIES

MARIANA D'ABREU is a M. Sc. student in the Computer Sciences Department of the Universidad de Buenos Aires, Argentina. She has worked in the IT industry in Argentina for the past 7 years. Her e-mail address is <mdabreu@dc.uba.ar>.

GABRIEL WAINER received the M.Sc. (1993) and Ph.D. degrees (1998, with highest honors) of the Universidad de Buenos Aires, Argentina, and Université d'Aix-Marseille III, France. He is Assistant Professor in the Dept. of Systems and Computer Engineering, Carleton University (Ottawa, ON, Canada). He was Assistant Professor at the Computer Sciences Dept. of the Universidad de Buenos Aires, and a visiting research scholar at the University of Arizona and LSIS, CNRS, France. He is author of a book on real-time systems and another on Discrete-Event simulation and more than 70 research articles. He is Associate Editor of the Transactions of the SCS. He is Associate Director of the Ottawa Center of The McLeod Institute of Simulation Sciences and a coordinator of an international group on DEVS standardization. His email and web addresses are <gwainer@sce.carleton.ca> and <www.sce.carleton.ca/faculty/wainer/>.

APPENDIX:

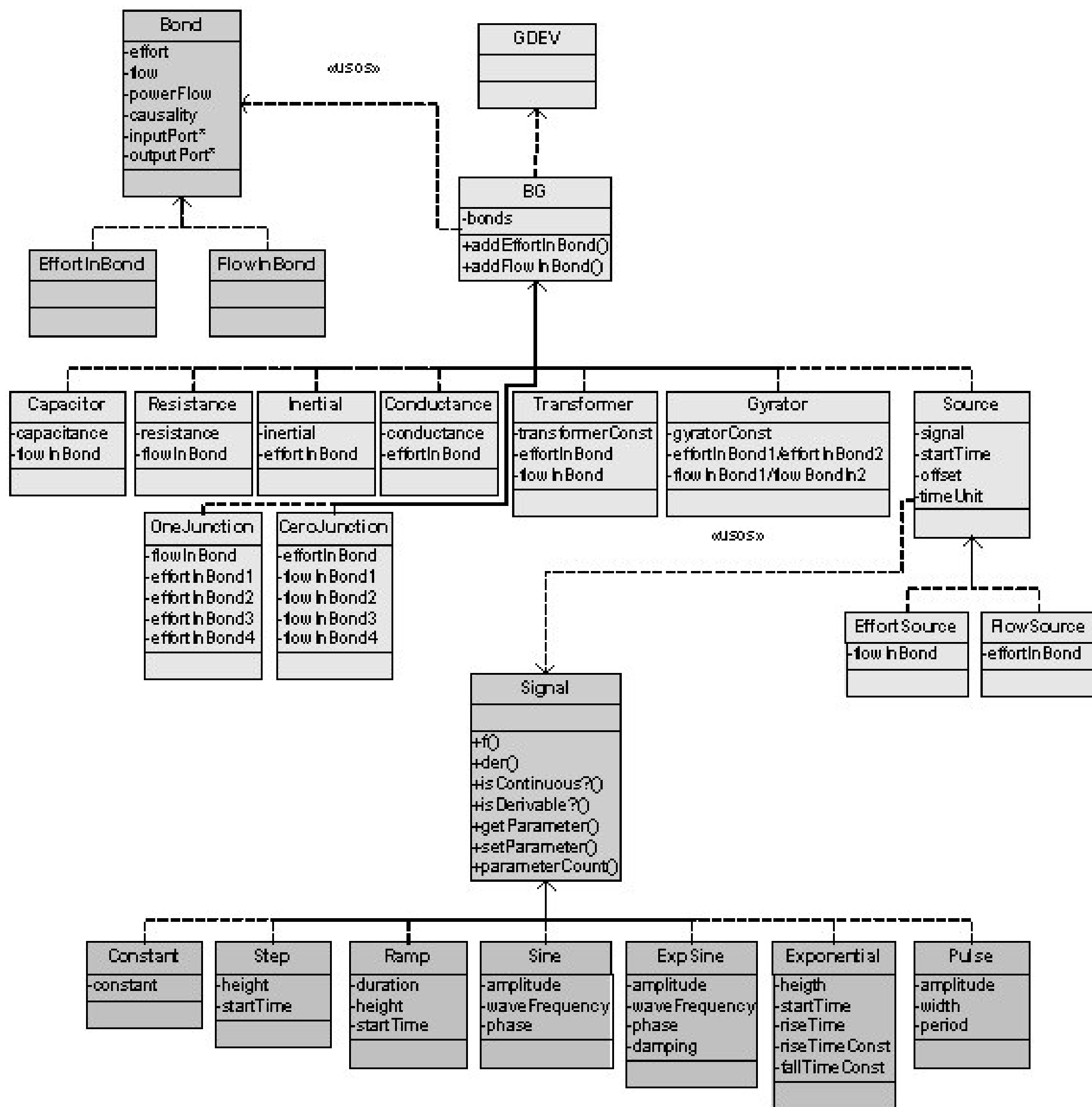


Figure 13: Bond-Graph Library models