Experimental Results On the Use of Modelica/CD++

Laurentiu Checiu

School of Information
Technology and Engineering
University of Ottawa.
800 King Edward Avenue
Ottawa, ON, K1N 6N5, Canada
Email: lchec097@uottawa.ca

Gabriel Wainer

Department of Systems and
Computer Engineering
Carleton University
1125 Colonel By Drive
Ottawa, ON, K1S 5B6, Canada
Email: gwainer@sce.carleton.ca

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. We have built a compiler of a subset of Modelica, and built a translator into DEVS models. 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 the Modelica models created is also presented in this paper.

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. Continuous dynamic systems are usually analyzed in terms of Differential Algebraic Equations (DAEs), Ordinary Differential Equations (ODEs), or Partial Differential Equations (PDEs). In most complex systems, solutions to these equations are very difficult or impossible to find. A variety of numerical methods finds approximate solutions to these equations, being successful in studying many different phenomena. Simulation-based approaches succeeded in providing a means of analyzing particular problems (instead of the general solutions obtained by solving PDEs).

Modelica [2] is an object-oriented language that permits attacking these problems. Modelica was created for modeling physical systems, designed to support library development and model exchange. Models in Modelica are mathematically described by differential, algebraic and discrete equations. Modelica has many libraries of standard components in different ODEs, block diagrams, electrical and mechanical models.

Simulation of continuous systems on digital computers requires discretization. Classical methods are based on discretization of time resulting in a discrete time simulation model. Instead, methods like DEVS (Discrete Event System specification) formalism [3] 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 theory evolved and it was recently upgraded in order to permit modeling of continuous and hybrid systems [4] [5]. 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. 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 on the creation of a Modelica compiler able to create Q-DEVS models [6][7]. 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 [8], a modeling and simulation framework that was developed in order to implement the theoretical concepts specified by the DEVS formalism.

2. BACKGROUND

DEVS (Discrete EVents Systems Specification) [3] 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:

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

X is the *set of external events*
Y is the *set of internal events*
S is the *set of sequential states*
d_{ext}: Q x $X \rightarrow$ S is the *external state transition function*
d_{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:

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

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 Ii,
$Z_{i,j}$ 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 [4] 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.

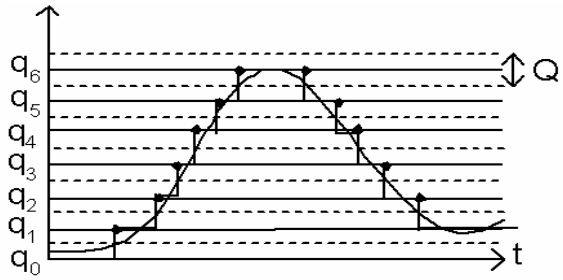


Figure 1: Signal Quantization

This approach requires a fundamental shift in thinking about the system as a whole. Instead of determining what value a dependant variable will have (its 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.

CD++ [8] 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 [3]. 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 [9] library as a set of independent models in which the components are developed using Q-DEVS [6].

Modelica is an object-oriented language, which was created for modeling physical systems, designed to support library development and model exchange. Models in Modelica are mathematically described by differential, algebraic and discrete equations. Modelica has many libraries of standard components in different ODEs, block diagrams, electrical and mechanical models. A model in Modelica is defined using classes, as described following:

```
class A
outer Real T0;
...
end A;
class B
inner Real T0;
A a1, a2; // B.T0, B.a1.T0 and B.a2.T0 is the same
variable
...
end B;
```

Figure 2. A model in Modelica

For instance, the representation of model of an electrical circuit is hierarchical; the electrical circuit can be decomposed into standard (basic) sub-models defined in the electrical library. The electrical library we defined for Modelica/CD++ starts by converting Modelica models into Bond Graphs [9]. Bond Graphs represent continuous systems as a set of elements that can interact with each other by exchanging energy and information, and this exchange determines the dynamics of the system. Energy is the fundamental feature that is exchanged between elements of a system during interaction. Power (energy time derivative) is the product of two factors: the effort and the flow. In electrical systems, power is the product of voltage and current; in hydraulic systems, power is the product of pressure and volume flow rate and so on. A generalized effort variable and a generalized flow variable can be defined whose product gives the power exchanged by the elements of a system. The idea of bond graphs came up because the exchange of energy and information between elements can be represented in a graphical form. Finally, Bond Graphs can be used to prove properties of the model, and then to translate them into DEVS models, following the method presented in [10].

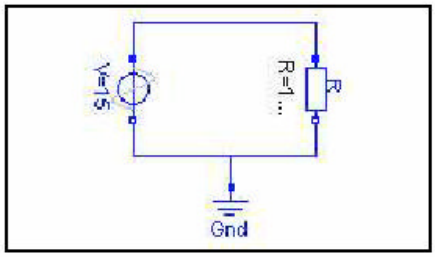
The user must provide a source code file as input to the Modelica compiler. The result is the corresponding model of the circuit in a Bond Graph representation. In this Bond Graph, we check for algebraic loops and singularities (elements that have discontinuities e.g. diode): if there is any, the process is stopped. Then, we generate an optimized Bond Graph corresponding to the electrical circuit, which is used to generate a coupled DEVS model specification according to the rules used by CD++.

There are different Bond Graph atomic models that we invoke with this purpose: *SerialJunction*, *ParallelJunction*, *SourceEffort* (constant, step, pulse, sine), *SourceFlow* (constant, step, pulse, sine), *Capacitor*, *Inductance*, *Resistance*.

3. CREATING MODELS IN MODELICA/CD++

In this section, we will present the definition of different Modelica/CD++ test cases, and their simulation execution results. We will present several simulations of electrical circuits (R, RC, RL) using sinusoidal and pulse voltage source. The results of our simulations are consistent with the real behaviour of the electrical circuits in discussion.

Our first example will consider the simulation of an electrical circuit with sine voltage. This model is presented in Figure 3 (where V generates a sine voltage).




```
Model circuit
Modelica.Electrical.Analog.Sources.SineVoltage
          V(V=15,freqHz=60);
Modelica.Electrical.Analog.Basic.Resistor
          R1(R=10);
Modelica.Electrical.Analog.Basic.Ground Gnd;
equation
  connect(V.p, R1.p);
  connect(R1.n, V.n);
  connect(R1.n, Gnd.p);
end circuit1;
```

Figure 3. An electrical model, and its corresponding representation in Modelica source code.

After the complete model is generated and executed, we obtain the following results:

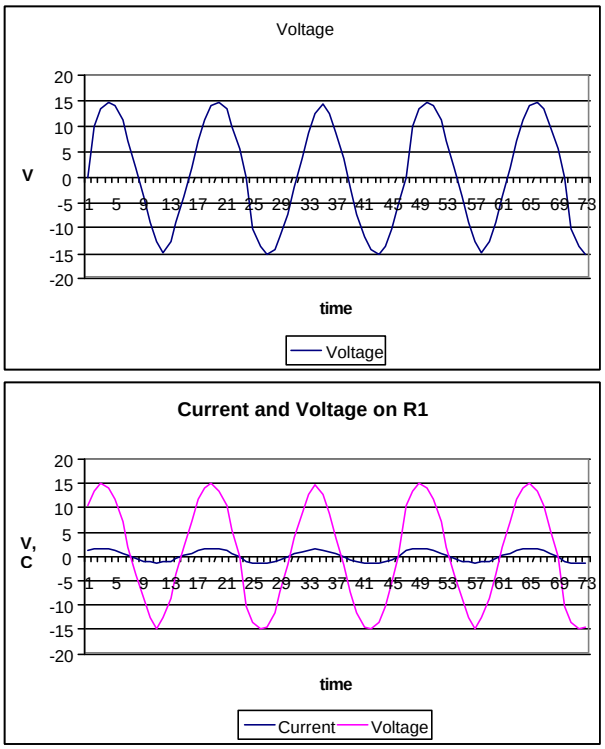


Figure 4. Electrical model with sine signal: (a) voltage; (b) current and voltage on R1.

The first graph (a) shows the voltage on the voltage source V, which is a sine wave with amplitude 15V. The second graph (b) shows the voltage and the current on resistor R. The resistor is a passive element in the oscillating electrical circuit so the current and the voltage on the resistor are in phase; the amplitude of electrical current is I=V/R1.

Our second example defines a simulation of an electrical circuit with pulse voltage. The structure of the model is similar to the one presented in Figure 4. Here, we show the Modelica/CD++ specification of the model (where V now provides a pulse voltage and we also added a capacitor).

```
model circuit
Modelica.Electrical.Analog.Sources.PulseVoltage
V(V=10,width=50,period=2);
  Modelica.Electrical.Analog.Basic.Resistor
R1(R=10);
  Modelica.Electrical.Analog.Basic.Capacitor
C(C=15);
  Modelica.Electrical.Analog.Basic.Ground Gnd;
equation
  connect(V.p, R1.p);
  connect(R1.n, C.p);
  connect(C.n, V.n);
  connect(V.n, Gnd.p);
end circuit;
```

Figure 5. An electrical model, and its corresponding representation in Modelica source code.

After the complete model is generated and executed, we obtain the following results:

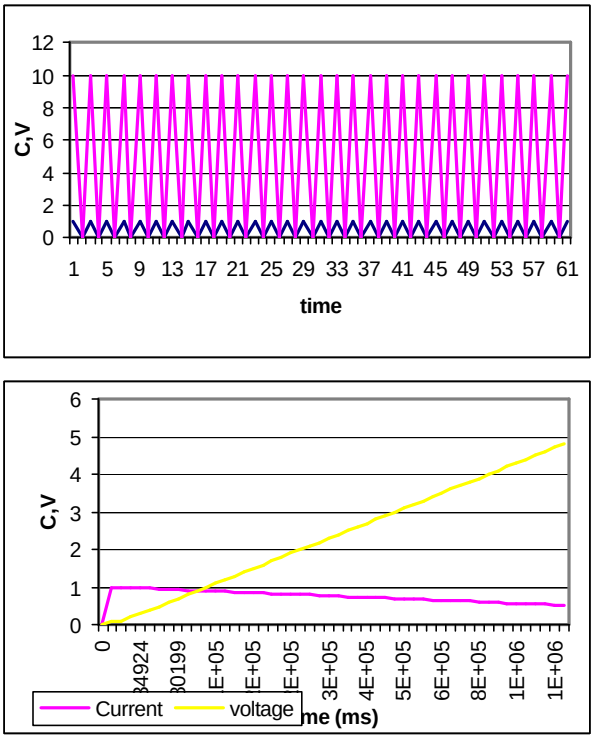


Figure 6. Electrical model with pulse signal: (a) Current and voltage on R1; (b) current and voltage on the Capacitor.

The resistor is a passive circuit element and, as described in previous example, current I=V/R1- in case (a). The voltage on capacitor – case (b) - will increase till the maximum value (10V) and the current will become zero, the capacitor will be fully charged. The voltage source provides only positive pulses so capacitor will not be discharged and the current on the capacitor remains zero.

In our next test, we added an inductor to the original circuit, using a pulse voltage again:

```
model circuit
  Modelica.Electrical.Analog.Sources.PulseVoltage
V(V=10,width=50,period=2);
  Modelica.Electrical.Analog.Basic.Resistor
R1(R=10);
  Modelica.Electrical.Analog.Basic.Inductor
I1(L=48);
  Modelica.Electrical.Analog.Basic.Ground Gnd;
equation
  connect(V.p, R1.p);
  connect(R1.n, I1.p);
  connect(I1.n, V.n);
  connect(V.n, Gnd.p);
end circuit;
```

Figure 7. An electrical model and its corresponding representation in Modelica source code.

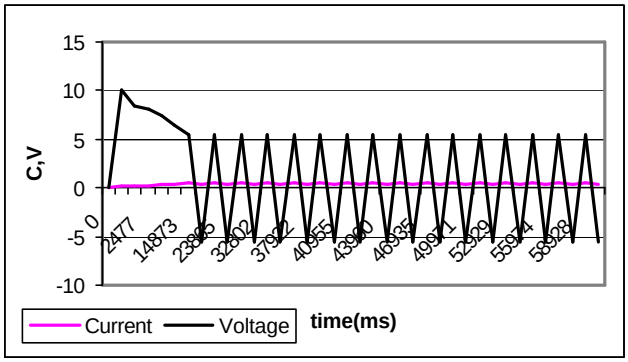


Figure 8. Electrical model with pulse signal and Inductor: Current and voltage on the Inductor.

The inductor is an active circuit element. When the current varies from zero to its nominal value (transient regime of the electrical circuit) there is an induction phenomenon on the inductor - the inductor generates voltage in this transition stage. The current on the circuit varies then from its nominal value to zero because the source voltage provides pulses, so there is again an induction phenomenon on the inductor which generates voltage. In conclusion, the voltage on the inductor varies in (-6V, 6V) interval as the source voltage V varies in the interval [0V, 10V].

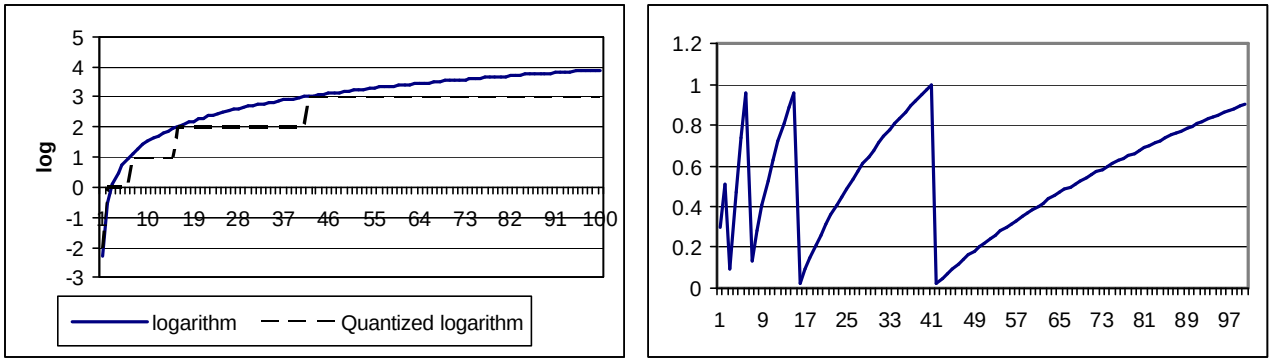
4. TESTING Q-DEVS MODEL EXECUTION

In this section, we introduce different models generated after the results of our intermediate steps. As mentioned in the Introduction, our Modelica/CD++ models are translated into Bond Graphs, which are used to test model properties. These models are then translated into DEVS model specifications in CD++. Here, we will show different experimental results obtained through varied simulations using different quantum sizes for the quantized versions of the models finally generated. Our first example shows a model generated by the tool, in which a hybrid model, mixing a logarithmic, and an attenuated sine signal (sin(x)/x) are defined. The following figure shows the model specification generated by Modelica/CD++ for this case.

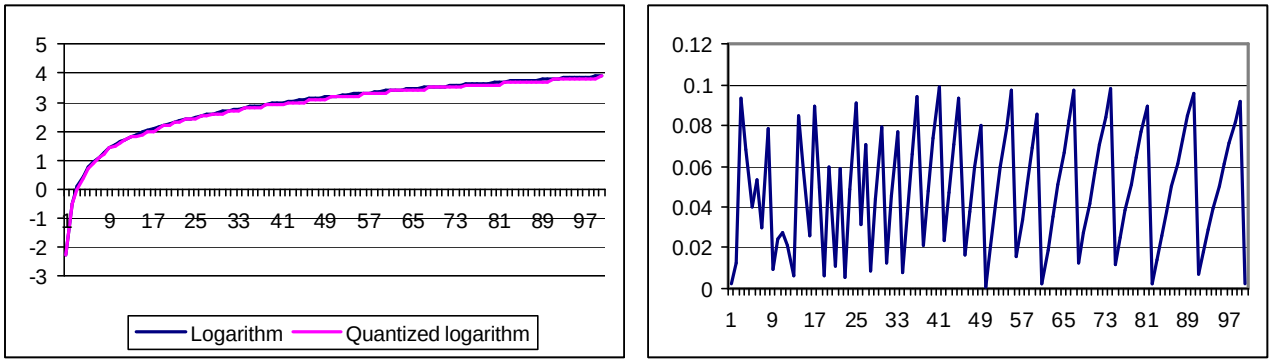
```
[top]
components : function@FunctionEvaluator
        quantizer@Quantizer  integrator@Integrator
in : in
out : out
Link : out@function in@quantizer
Link : out@quantizer in@integrator

[function]
initialValue   : 0.1   lastValue     : 50
qValues        : 100   function  : log
evalFrequency  : 0.001
```

Figure 9. Q-DEVS model generated: Integrator, Quantizer and FunctionEvaluator: logarithmic

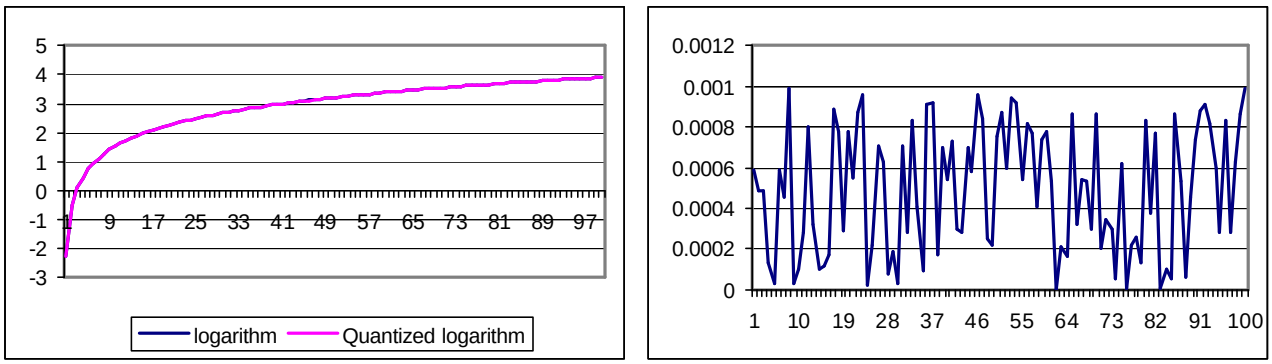
As we can see, Modelica/CD++ generates a coupled model (*top*) consisting of three atomic components: *Integrator*, *Quantizer*, and *FunctionEvaluator*. These models are part of Modelica/CD++ library, and have been built as DEVS models executable in CD++. The *FunctionEvaluator* receives different parameters (*function)*, in this case, the type of function to be evaluated (*log*arithm), the initial/last values we want to generate, and the evaluation frequency. The *quantizer* simply quantizes the input signal using a predefined value (in this case, q=1). Finally, the integrator computes the integration of the input value, using the Euler methods with a time scale of 0.05 time units. Finally, the *link* directive shows how the three models are coupled to each other.



(a)



(b)



(c)

Figure 10. Q-DEVS model execution: Logarithm vs. Quantized version and simulation Error charts. (a) q=1; (b) q=0.1; (c) q=0.01.

In the previous figure, we show the execution results of the model, and the error obtained at each step of the execution. As we can see, the approximation improves highly while the quantum size is reduced. This is because more messages are involved in the simulation, and higher precision can be obtained. In the worst possible case, the error is 0 when compared to the original signal, but we have to pay the cost of increased execution time for the model).

We executed this model varying the size of the quantum size, and obtained the results presented following.

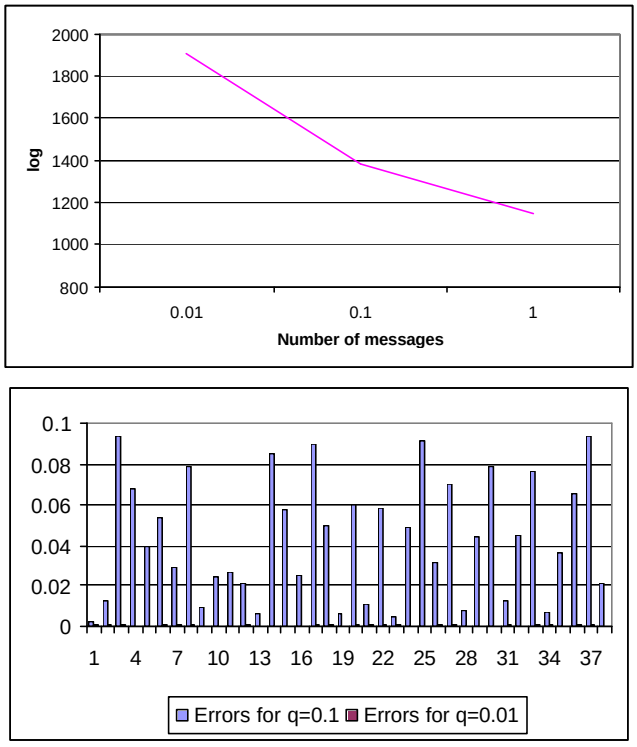


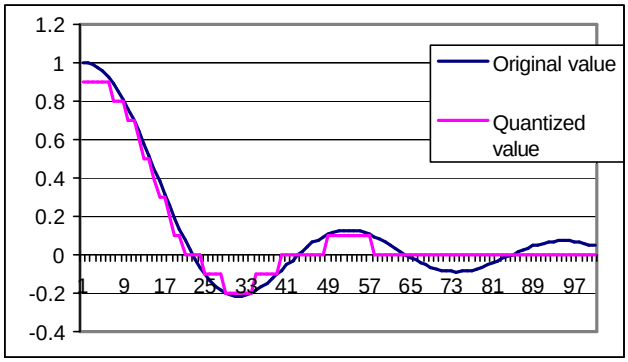
Figure 11. Q-DEVS model execution: Number of messages in the simulation, and comparative error

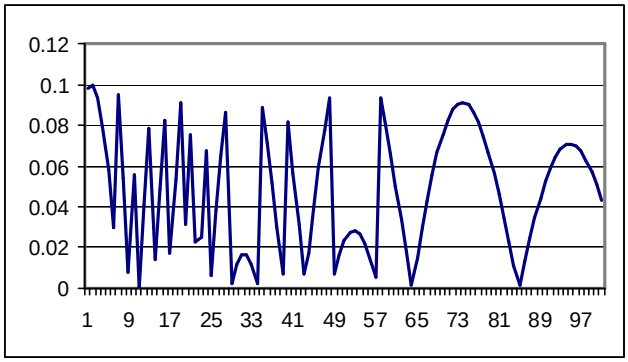
As we can see, the number of messages involved reduces substantially when the quantum size increases, confirming the theoretical results presented in [4], and the empirical results presented in [11].

A second example we will present here shows the execution of a model generated using an attenuated sine signal (sin(x)/x). The model generated is the same than the one presented in Figure 9, but the function chosen has been changed to:

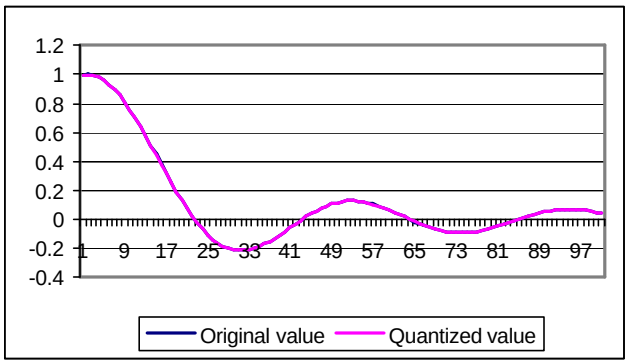
```
[function]
initialValue  : -0.1
lastValue     : 15.0
qValues       : 100
function      : sinat
evalFrequency : 0.01
```

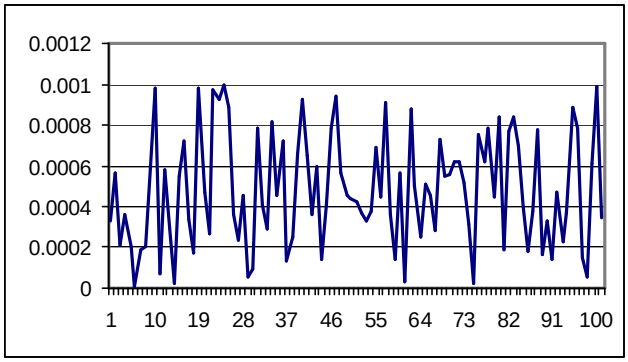
Figure 12. Q-DEVS model generated: attenuated sine





(a)





(b)

Figure 13. Q-DEVS model execution: Attenuated sine vs. Quantized version; simulation Error. (a) q=0.1; (b) q=0.001.

In this particular case, there is 55% increase in number of messages (1911 vs. 1228) with 100 times error improved (decreased). The number of messages increased because, in this case, there are more crossings of the boundaries defined by the quantum size (q=0.001). The error doesn't exceed the quantum size.

5. CONCLUSIONS

The DEVS formalism is a method defined for modeling and simulation of discrete event systems. During the last years the DEVS theory has evolved, and it was recently upgraded in order to permit simulation of continuous and hybrid systems. In this work, we present a tool for modeling and simulation of continuous systems based on the DEVS formalism. Models are described using Modelica, a modular and acausal standard specification language for physical systems modeling.

A hierarchy of models was developed within CD++ to give support in the simulation of continuous and hybrid systems. A Q-DEVS 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. In some cases, this reduction can be very significant.

It is important to have in mind that Modelica/CD++ approximates the system solution based on the Q-DEVS method, which uses a simple first order integration approach. Most of the results produced by Modelica/CD++ were contrasted with results generated using a higher order and variable step-size integration method, DASSL. It was shown that, in general, choosing adequate quantization parameters produce accurate solutions and decrease error.

Different open topics must be considered for future research in this area. 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).

REFERENCES

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

[2] Åström, K. J; Elmqvist, H.; Mattsson, S. E. "Evolution of continuous-time modeling and simulation". The 12th European Simulation Multiconference, ESM'98, Manchester, UK, June 1998.

[3] B. Zeigler, T. Kim, and H. Praehofer. *Theory of Modeling and Simulation*. Academic Press. 2000.

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

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

[6] M. D'Abreu, G. Wainer. "Modelica/CD++: a compiler for continuous systems on DEVS". Technical Report SCE-05-07. Carleton University. Submitted for publication. 2005.

[7] M. D'Abreu. "M/CD++: a tool for hybrid modeling and simulation". M. Sc. Thesis. Computer Science Dept. Universidad de Buenos Aires, Argentina. 2004.

[8] G. Wainer. "CD++: a toolkit to develop DEVS models". *Software-Practice and Exp*. 32, 1261-1306. 2002.

[9] Cellier, F.E.; Elmqvist, H. "Automated formula manipulation supports object-oriented continuous-system modeling" IEEE Control Systems, 13(2), pp. 28-38, April 1993.

[10]D'Abreu, M.; Wainer G. "Defining hybrid system models using DEVS quantization techniques". In Proceedings of the Winter Simulation Conference. New Orleans, LA. U.S.A., 2003.

[11]Wainer, G.; Zeigler, B. "Experimental results of Timed Cell-DEVS quantization". In *Proceedings of AIS'2000*. Tucson, Arizona. U.S.A. 2000.