# Experimental results on the implementation of Modelica using DEVS modeling and simulation

Mariana C. D'Abreu

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

Gabriel A. Wainer

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

***Abstract**: We present experimental results on the use of M/CD++, a Modelica compiler built on CD++. CD++ is an implementation of the DEVS formalism, a method originally created for Modeling and Simulation of discrete-event systems. We combined DEVS with quantization techniques, and created a toolkit that can be used for analysis of electrical systems using Modelica notations. This approach makes easier the integration of continuous and discrete event systems within the same environment. In order to prove the feasibility of the approach, we compare the results obtained by M/CD++ with those of a commercial tool (Dymola), showing promising results.*

## 1. INTRODUCTION

In [1], we presented the design and implementation of M/CD++, a tool based on *Modelica* [2] and *CD++* [3], a modeling and simulation tool implementing DEVS theory. Modelica is an object-oriented language 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 includes different libraries of standard components providing ODEs, block diagrams, and electrical and mechanical models. M/CD++ allows the creation of dynamic systems belonging to the electrical domain.

Although most existing simulation tools find solutions for continuous systems by finding approximate solutions to the equations representing the systems [4] (which are based on discretization of time), in the last few years, different efforts tried to simulate continuous systems under the discrete event paradigm. This presents some advantages over discrete time simulation, including reduction of the number of calculations for a given accuracy [5] and seamless integration of complex systems composed by both continuous time and discrete events. These solutions are based on the DEVS (Discrete Event Specification) formalism [6]. The idea of this method, called Quantized Systems theory (Q-DEVS), is to provide quantization of the state variables obtaining a discrete event approximation of the continuous system [7]. The state variables of the system are thus converted into a piecewise constant function via a quantization function [7]. The Quantized State System (QSS) method [8] is an extension to Q-DEVS with hysteresis, a general method for integration of ODEs using discrete event theory.

M/CD++ permits creating electrical circuits specified using Modelica standard notation, which are subsequently translated into Bond Graphs [9], which are used to check for algebraic loops and singularities (elements that have discontinuities; e.g. diodes). Then, we generate an optimized BG corresponding to the electrical circuit, which, in turn, is used to generate a DEVS model specification according to the rules used by CD++. In order to evaluate the simulation results obtained with M/CD++, test cases were created, and their results were evaluated using *Dymola* [10], a commercial simulation toolkit with full support of the Modelica language. The idea about using Dymola was to compare the results given by M/CD++ with those calculated by an existing and proven physical systems simulation tool with Modelica support. We will present an introduction to the facilities available in the tool (further details can be found in [1]), and will discuss the results obtained when creating different electrical circuits, studying the error obtained when compared to Dymola.

## 2. BACKGROUND

The continuous behavior of dynamic systems is usually described in terms of Differential Algebraic Equations (DAEs), Ordinary Differential Equations (ODEs) and Partial Differential Equations (PDEs). Dynamic systems simulation based on these formalisms is mainly accomplished numerically solving the set of differential equations describing the system and finding consistent initial conditions [1][2]. In recent years, there have been numerous efforts focusing on how to model complex physical system via system decomposition. The idea is to divide the system into a number of smaller subsystems interfaced by distinct connections, and some of the techniques use Object-oriented modeling to promote models specification in a more natural way. Object orientation permits decreasing the abstraction gap between the real system and the representation model, and allows the development and reusability of models within a hierarchical con-

struction process [9][11]. Numerous of these concepts were adopted and applied to the design of a new family of modeling and simulation tools for continuous systems modeling. Modelica [2] is one of such languages.

Figure 1 presents an example of an electrical circuit specified using Modelica's electrical library:

```
model circuit
  Mode-
lica.Electrical.Analog.Sources.PulseVoltage
V(V=200, period=1, width=10);
  Modelica.Electrical.Analog.Basic.Capacitor
C(C=200);
  Modelica.Electrical.Analog.Basic.Resistor
R(R=1.5);
  Modelica.Electrical.Analog.Basic.Inductor
I(L=40);
  Modelica.Electrical.Analog.Basic.Ground Gnd;
 equation
  connect(V.p, R.p);
  connect(R.n, I.p);
  connect(R.n, C.p);
  connect(I.n, V.n);
  connect(C.n, V.n);
  connect(C.n, Gnd.p);
 end circuit;
```

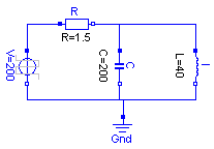



*Figure 1. Modelica specification of a sample circuit.*

In [1] we presented the design and implementation of M/CD++, a tool to construct continuous systems based on Modelica, using DEVS as the underlying formalism. M/CD++ permits simulating electrical circuit models like the one in Figure 1 by implementing a subset of Modelica's language specification. M/CD++ models are converted into Bond Graphs (BG), which are subsequently translated into DEVS [6]. DEVS has been used recently for continuous systems simulation by different research teams [5][7][8][12][13][14][15][16] [17]. In these articles, it has been shown that discrete event methods in general and DEVS in particular, present several advantages (which are the ones we considered when we created M/CD++):

- Computational time reduction: for a given accuracy, the number of calculations can decrease
- Hierarchical modular modeling
- Seamless integration with models defined with other modeling techniques mapped to DEVS
- Simulation of discrete time models: can be seen as particular cases of discrete event methods
- Hybrid systems modeling: the discrete event paradigm provides the theory to develop a uniform approach to model and simulate systems with continuous and discrete components.

Most of these techniques are based on Q-DEVS [7], whose main idea is to represent continuous signals by the crossing of an equal spaced set of boundaries. 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 the variable will enter a given state. QSS (Quantized State Systems) [8][12] is an extension to Q-DEVS, which introduces the concept of hysteresis for Q-DEVS. In [8] it was proved that differential equation systems can be approximated by legitimate DEVS models with QSS; the addition of hysteresis in the quantizer removes the problem of the possible infinite number of transitions performed by a model in a finite time interval greater than zero [12], and the existence of a minimum time interval between events constitutes a sufficient condition to obtain legitimate models [8].

CD++ is a toolkit that implements DEVS theories [3]. Atomic models are implemented using a built-in specification language [18] or in C++, adding flexibility and construction power to the developer. New atomic models extend the behavior of the basic atomic model and they must inherit from the Atomic class, provided by the tool. Coupled models are described in a configuration file using a specification language provided by the tool. The file includes information about the components, the coupling and the input and output ports associated to the model. CD++ also enables the user to define coupled models by using a built-in specification language that follows DEVS formal specifications.

M/CD++ allows simulating dynamic systems in the electrical domain using CD++. M/ CD++ contains language support for a subset of *Modelica v2.1* [2], including the components needed to allow electrical circuits construction provided by the Modelica Electrical library. Electrical circuit simulation capabilities are based on the implementation of QSS concepts: the resulting CD++ model represents the equations system associated to the electrical circuit that has to be solved. Based on QSS and Quantized Bond Graph theories, we constructed Atomic models that were added to CD++, approximating numerically the solution using a discrete event approach. The objects defined in the Modelica Electrical library and supported on M/CD++ are: *Modelica.Electrical.Analog.Basic: Ground, .Resistor, .Conductor, .Capacitor, .Inductor; Modelica.Electrical.Analog.Ideal: .IdealTransformer, .IdealGyrator; Modelica.Electrical.Analog. Sources:ConstantVoltage, .StepVoltage, .SineVoltage, .PulseVoltage, .ConstantCurrent, .StepCurrent, .SineCurrent and.PulseCurrent.*

## 3. *M/CD++* SIMULATION ERROR ANALYSIS

In this section it will be presented several test cases executed using M/CD++, providing some examples to assess the precision of the toolkit. It will also be shown the error analysis performed over the generated results. As mentioned in the Introduction, we created the different circuits using Dymola, and compared the results to compare the error obtained with M/CD++. We used Dymola version 5.1b, whose simulator, *Dymosim*, provides a number of different integration methods for the simulation of dy-

namic systems. It includes fixed and variable step-size algorithms, many of them of fixed order, and others with order variation during the simulation. Most integration methods available have a variable step-size algorithm. The integration step-size is chosen in such a way that the local estimated error is smaller than the desired maximum local error, defined via the relative and absolute tolerances. Then, integration method and relative tolerance are parameters that can be specified on the simulation setup.

The objective of this work was not to carry out a detailed performance study, moreover considering CD++ is an open-source project developed by graduate students, and Dymola is a very advanced commercial tool. Likewise, M/CD++ is the first implementation of our tool (developed by one person only) and it does not contain any optimizations. The goal of our work was to achieve the objectives analyzed in section 1, which are mostly related to provide a mechanism for interfacing discrete-event and continuous models in a seamless way, permitting executing the solution in different environment (including real-time and parallel simulators that CD++ is provided with). Nevertheless, in many cases the performance we achieved is equivalent or better than the one by Dymola.

The integration methods used on *Dymosim*, to solve the systems described by the test cases on this chapter, were *DASSL* and *Euler*. *DASSL* is a variable step-size and variable order (1-5) algorithm [19][20]. It is used to integrate DAE and ODE systems. *Euler* is a fixed step-size first order algorithm intended for real time simulation.

In order to compare the trajectories obtained with M/CD++ and those calculated by Dymola, the interpolation of the related curves has to be performed. Then, M/CD++ states trajectories are linearly interpolated with the Dymola states trajectories through

$$\tilde{f}(x_j) = \frac{f_d(x_{i+1}) - f_d(x_i)}{|x_{i+1} - x_i|}.(x_j - x_i) + f_d(x_i) \quad x_i <= x_j <= x_{i+1} \quad \wedge \quad |x_{i+1} - x_i| > 0$$

In our case, $\tilde{f}$ corresponds to the interpolated M/CD++ state trajectory and $f_d$ to the Dymola state trajectory. Then, the relative error between trajectories can be calculated as:

$$e = \sum_j \frac{|\tilde{f}(x_j) - f_{mcd}(x_j)|}{|\tilde{f}(x_j)|}$$

where $f_{mcd}$ corresponds to the state trajectory calculated by M/CD++.

### 3.1 Example 3.1

On this example, the electrical circuit shown on Figure 5 is simulated. Two test cases were executed for this model. Test cases have a variation of the integration method used on Dymola, in order to compare the results with M/CD++ calculated trajectories.

```
model example3.1
  Modelica.Electrical.Analog.Basic.Capacitor Ca-
pacitor1(C=30);
  Modelica.Electrical.Analog.Basic.Resistor  Re-
sistor1;
  Modelica.Electrical.Analog.Basic.Inductor  In-
ductor1(L=40);
  Modelica.Electrical.Analog.Basic.Inductor  In-
ductor2;
  Modelica.Electrical.Analog.Basic.Capacitor Ca-
pacitor2(C=30);
  Modelica.Electrical.Analog.Basic.Inductor  In-
ductor3;
  Modelica.Electrical.Analog.Basic.Inductor  In-
ductor4(L=80);
  Modelica.Electrical.Analog.Basic.Capacitor Ca-
pacitor3(C=40);
  Modelica.Electrical.Analog.Basic.Ground
Ground1;
  Mode-
lica.Electrical.Analog.Sources.ConstantCurrent
ConstantCurrent1;
equation
  connect(Capacitor1.p, Inductor2.p);
  connect(Resistor1.p, Inductor2.p);
  connect(Inductor1.p, Inductor2.p);
  connect(Capacitor2.p, Inductor2.n);
  connect(Inductor3.n, Capacitor3.p);
  connect(Inductor3.n, Inductor4.p);
  connect(Capacitor1.n, Resistor1.n);
  connect(Resistor1.n, Inductor1.n);
  connect(Capacitor3.n, Inductor4.n);
  connect(Inductor2.n, Inductor3.p);
  connect(Capacitor2.n, Capacitor3.n);
  connect(ConstantCurrent1.p, Inductor2.p);
  connect(Inductor1.n, ConstantCurrent1.n);
  connect(ConstantCurrent1.n, Capacitor2.n);
  connect(Ground1.p, Capacitor3.n);
end example3;
```

*Figure 2 – Modelica code for circuit on example 3.1*

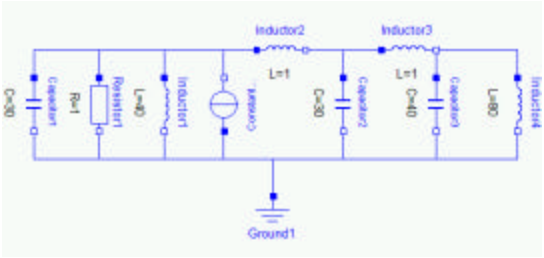



*Figure 3– Electrical circuit Example 3.1*

We simulated this model circuit using the DASSL integration method on Dymola, for 60 seconds of simulated time, with 500 intervals and a tolerance of 0.0001. We compared the results obtained with a QSS DEVS model implemented by M/CD++, using the following parameters for each of the components:

| *Component* | *Quantum* | *Hysteresis window* |
|---|---|---|
| Capacitor1 | 0.0005 | 0.00025 |
| Capacitor2 | 0.0001 | 0.00005 |
| Capacitor3 | 0.0001 | 0.00005 |
| Inductor1 | 0.0001 | 0.00005 |
| Inductor2 | 0.0005 | 0.00025 |
| Inductor3 | 0.001 | 0.0005 |
| Inductor4 | 0.0001 | 0.00005 |

The following figures show the error for the capacitor (*C1*) and the state trajectories for the inductor (*I1*) on M/CD++ and Dymola for the given parameters:

| | ***Relative error average*** | |
|---|---|---|
| ***Curve*** | ***Case 3.1*** | ***Case 3.2*** |
| Capacitor1.v (C1.v) | 0.28 % | 0.288 % |
| Inductor2.i (I2.i) | 3.33 % | 1.28 % |

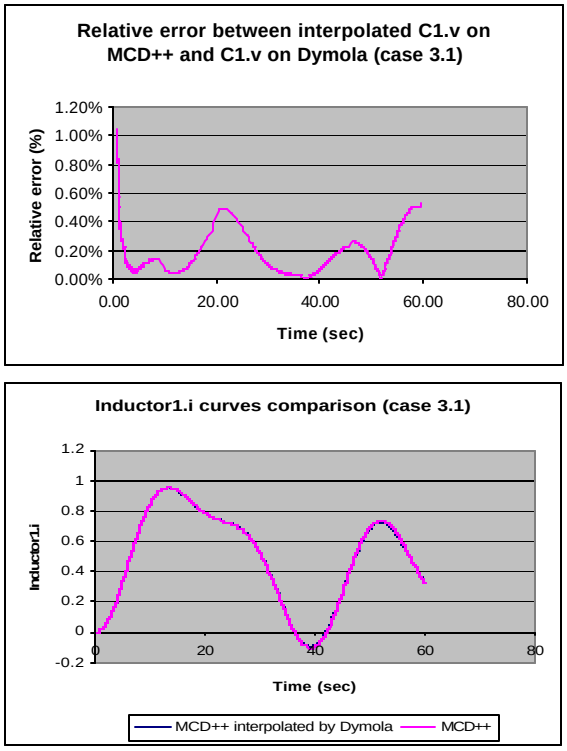



*Figure 4. (a) Relative error M/CD++ vs. Dymola for v. on C1 (b) M/CD++ and Dymola for current on I1*

*Figure 7 (a)* shows the relative error curve for voltage on capacitor *C1*. *Figure 7 (b)* shows the trajectories for the current on inductor *I1*. It can be seen that M/CD++ approximates the model trajectories well, and that the relative error is constrained (below 0.5%). Larger relative errors are obtained for points near to zero, given the fixed quantum size used through the entire simulation (if we see the formula we used to compute the relative error in section 3, we are dividing the difference between the execution of both models by $\tilde{f}(x_j)$; when this value tends to zero, the relative error grows, as we see in the crossing zero areas).

### 3.2 Example 3.2

This example just changed the simulation of the *Example 3.1* circuit using the Euler integration method on Dymola. The simulation time is 60 seconds, and we used an integration step of 0.01, with a tolerance of 0.0001. We compare the results of the model with the same quantum size/hysteresis window presented for Example 3.1, permitting us to compare the results with a more traditional integration algorithm.

The following figure show the error between the capacitor (*C1*) and inductor (*I1*) state trajectories on M/CD++ and Dymola for the given simulation parameters:

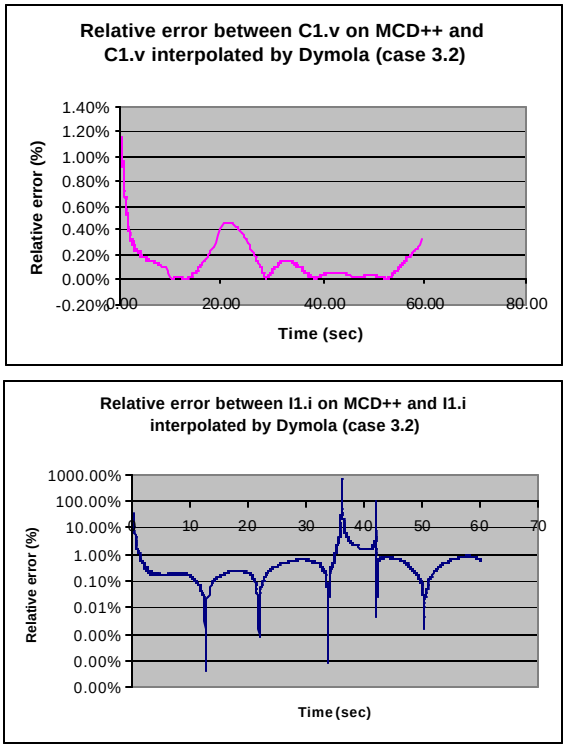



*Figure 5 – Simulation results for test case 3.2. Relative error for current on I1*

*Figure 8* shows how the relative error is still small, and the larger relative error is for values near to zero, although this value was reduced when using a fixed step-size integration method on Dymola, as the Euler algorithm. The following table shows the Relative error averages for test cases on *Example 3.1* circuit.

As we can see, although we are using a fundamentally different approach (a discrete-event simulator for continuous systems models), the amount of the error, while compared with well-established numerical methods is reduced. As showed in [13] (and as we will discuss in section 3.5), these values can be even improved by choosing a smaller quantum size for the Q-DEVS version.

### 3.3 Example 3.3

In this case, we implemented the electrical circuit on *Figure 9*. Two test cases were executed for this model varying the integration method used on the simulation with Dymola.

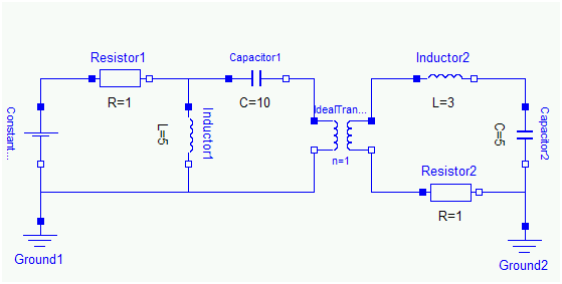



*Figure 6 – Electrical circuit of example 3.3*

Initially, we simulated *Example 3.3* circuit using the DASSL integration method on Dymola. The simulation time was 30 seconds, and we used 500 intervals at a precision of 0.0001. The following table presents the parameters used for the DEVS models running in M/CD++.

| *Component* | *Quantum* | *Hysteresis window* |
|---|---|---|
| Capacitor1 | 0.0001 | 0.00005 |
| Capacitor2 | 0.001 | 0.0005 |
| Inductor1 | 0.0005 | 0.00025 |
| Inductor2 | 0.0008 | 0.0002 |

The following figures show the error, for the current trajectory on the ideal transformer, between M/CD++ and Dymola for the given simulation parameters.

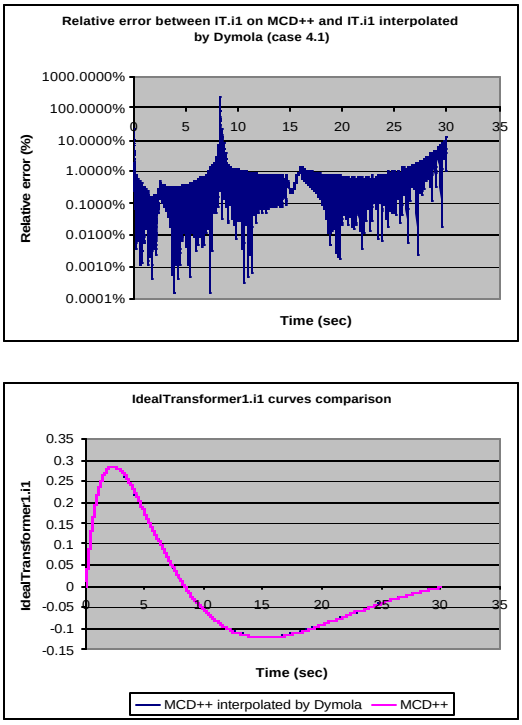
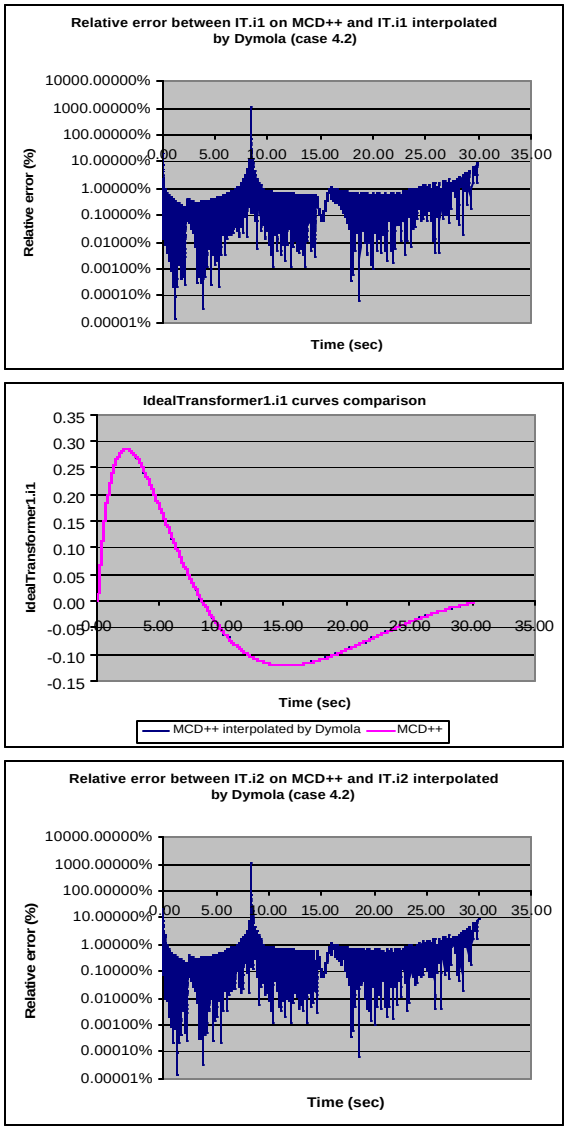



*Figure 7 – (a) Relative error: between current on the ideal transformer (b) Current curves comparison.*

*Figure 11 (a)* shows the relative error, between M/CD++ and Dymola, on the ideal transformer input current. State trajectories on the system where calculated on Dymola using the DASSL integration method. Again, we can see that the approximation is very accurate (the difference between the curves obtained is indistinguishable).

### 3.4 Example 3.4

In this case, we repeated the studies carried out in Example 3.3, but using the Euler integration method on Dymola, using an integration step of 0.005, and a tolerance of 0.0001. The following figures show the error, for the current trajectories on the ideal transformer (input and output flow), between M/CD++ and Dymola for the given simulation parameters. A similar error to the one produced on the previous case is given using the Euler integration method on Dymola, with a step size equal to 0.005. Results can be seen on *Figure 12 (a)* and *(c)*.



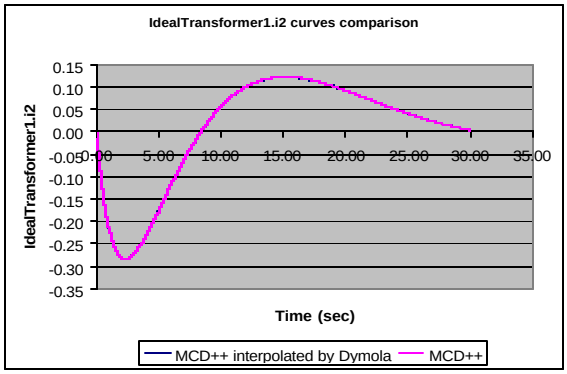



*Figure 8– (a) (b) Error comparison for current i1 (flow out) on the ideal transformer (c) (d) Error for current i2 (flow in) on the ideal transformer*

The following table shows the relative error averages we found for test cases on *Example 3.4* circuit.

| | ***Relative error average*** | |
|---|---|---|
| ***Curve*** | ***Case 4.1*** | ***Case 4.2*** |
| IdealTransformer1.i1 | 0.61 % | 0.62 % |

### 3.5 Example 3.5

The electrical circuit on *Figure 13* is simulated on this example. Two test cases were executed for this model varying the quantization parameters used for state trajectories on M/CD++.

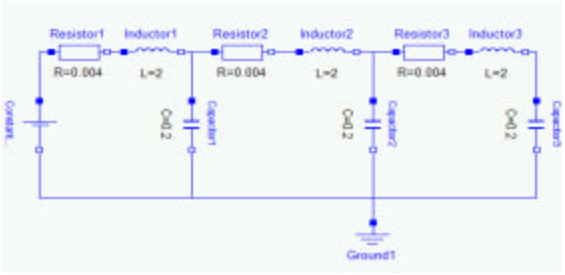



*Figure 9– Electrical circuit of example 3.5*

Simulation of the *Example 5* circuit using the DASSL integration method on Dymola, for 15 seconds. Again, we used 500 intervals with a precision of 0.0001. The following figure shows the parameters for M/CD++ simulations.

| ***Component*** | ***Quantum*** | ***Hysteresis window*** |
|---|---|---|
| Capacitor1 | 1.60 | 0.5 |
| Capacitor2 | 1.60 | 0.5 |
| Capacitor3 | 1.60 | 0.5 |
| Inductor1 | 0.50 | 0.10 |
| Inductor2 | 0.50 | 0.10 |
| Inductor3 | 0.50 | 0.10 |

The following figures show the error, for the state trajectories on capacitor (C1) and inductor (I1), between M/CD++ and Dymola for the given simulation parameters:

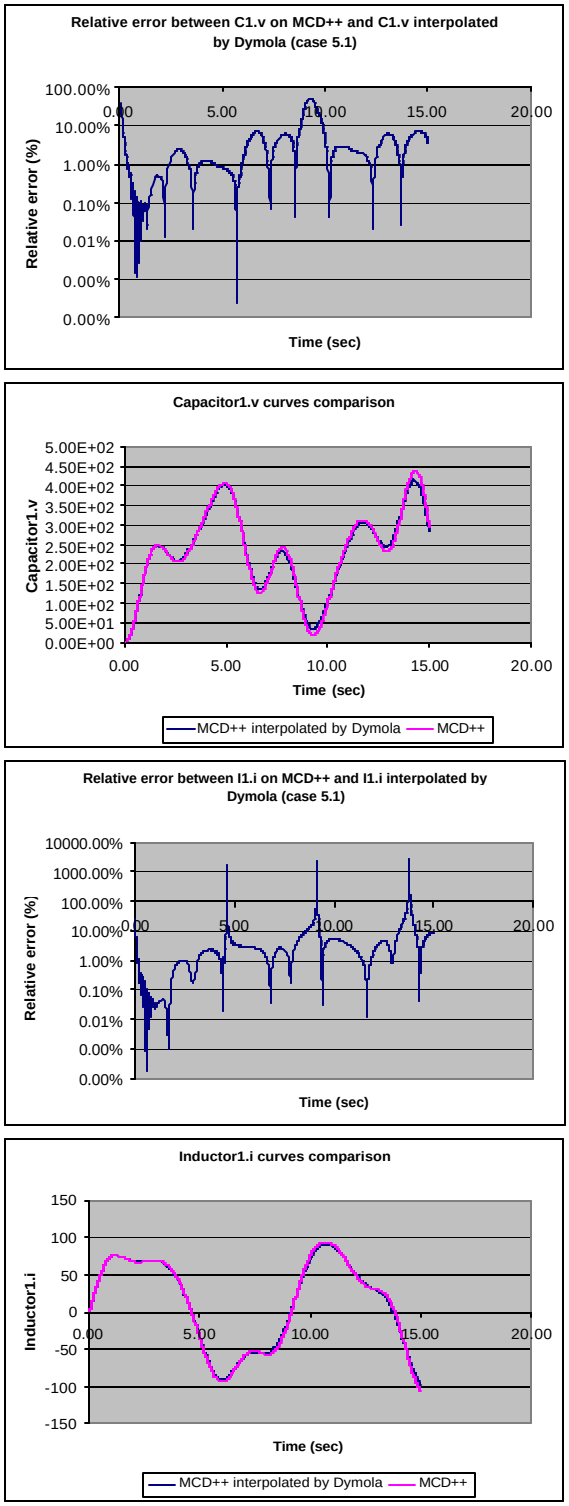



*Figure 10– (a), (b) Error for voltage curve on C1 (c) (d) Error for current curve on I1*

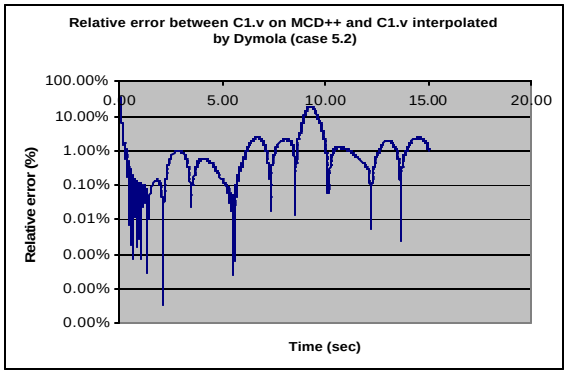
This test case was simulated using the DASSL method on Dymola. *Figure 15 (a)* and *(b)* show the higher relative errors on the zero crosses for state trajectories on *capacitor1* and *inductor1*. Next test case will show how de-

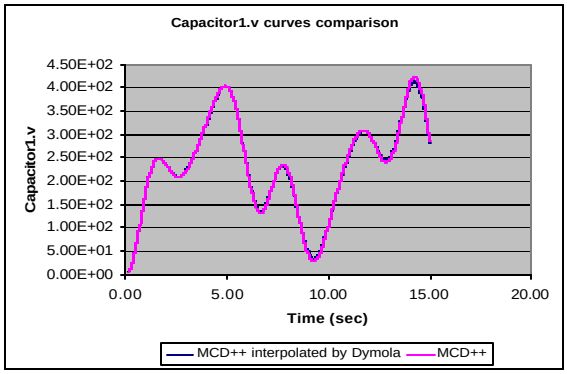
creasing the quantum size will produce more accurate results.

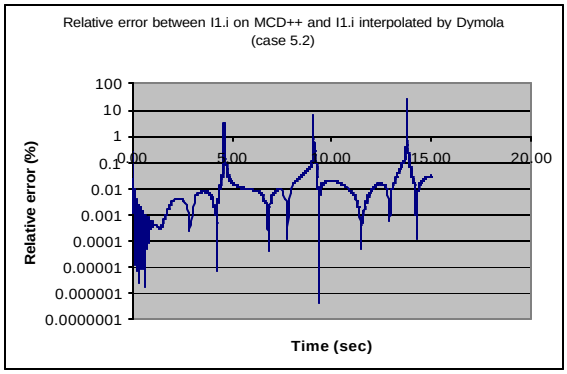
### 3.6 Example 3.6

We simulated the circuit of *Example 3.5* circuit using the DASSL integration method on Dymola and decreasing the quantum and hysteresis window size on M/CD++ simulation. The following table shows the new simulation parameters for M/CD++ simulations.

| *Component* | *Quantum* | *Hysteresis window* |
|---|---|---|
| Capacitor1 | 0.65 | 0.01 |
| Capacitor2 | 0.60 | 0.01 |
| Capacitor3 | 0.70 | 0.01 |
| Inductor1 | 0.25 | 0.01 |
| Inductor2 | 0.20 | 0.01 |
| Inductor3 | 0.20 | 0.01 |



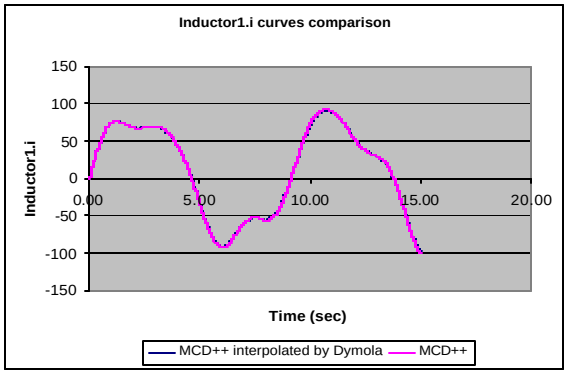




*Figure 11– (a) (b) Error for voltage curve on C1 (c) (d) Error for current curve on I1*

The following table shows the relative error averages for all the test cases executed on *Example 3.5* circuit.

| | *Relative error average* | |
|---|---|---|
| *Curve* | *Case 5.1* | *Case 5.2* |
| Capacitor1.v | 3.84 % | 1.37 % |
| Inductor1.i | 17.13 % | 6.04 % |

## 4. CONCLUSION

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. We introduced a tool for modeling and simulation of continuous systems based on DEVS. Models are described using Modelica, a modular and acausal standard specification language for physical systems modeling. Examples of model simulation with their execution results are included. The simulation results generated by M/CD++ were compared with those produced by a complex physical system simulation environment with Modelica support called *Dymola*. Several test cases were executed using both toolkits, varying the quantization parameters used on M/CD++ and the integration methods utilized by Dymola.

It was shown that a higher relative error is obtained for values near to zero on a trajectory. This is related with the fixed quantum size used by the quantization function over a state trajectory, and the formulas used for computing the relative error. Then, for smaller values, greater differences are given. An approach to improve the simulation results could be developed using an adaptive quantization function, making the quantum vary according to the trajectory evolution. Promising results on dynamic quantization can be found in [21] and [16].

On the examples, it was also shown how accurate results could be obtained reducing the quantum and hysteresis window size. It is important to have in mind that M/CD++ approximates the system solution based on the

QSS method, which uses a simple first order integration approach. Most of the results produced by M/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.

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. 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 (continuous time, discrete time, and discrete event).

**REFERENCES**


[1] D'Abreu, M.; Wainer, G. "M/CD++: modeling continuous systems using Modelica and DEVS". Proceedings of MASCOTS 2005. Atlanta, GA. 2005.

[2] Modelica Language Specification, version 2.1, http://www.modelica.org March 2004.

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

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

[5] Zeigler, B.P., "Continuity and Change (Activity) are Fundamentally Related in DEVS Simulation of Continuous Systems", LNCS, Vol. 3397/2005, Springer-Verlag, NY, pp. 1-17.

[6] Zeigler, B; Kim, T; Praehofer, H. "Theory of Modeling and Simulation". Academic Press. New York, 2000.

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

[8] Kofman, E. "Discrete Event Based Simulation and Control of Continuous Systems". PhD's thesis. Universidad Nacional de Rosario, Argentina. August 2003.

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

[10] Dynasim Laboratories. "Dymola". [online]. Available online via: http://www.dynasim.com/dymola.htm [Accessed June 13 2004]

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

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

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

[14] Wainer, G., B.P. Zeigler, "Experimental Results of Timed Cell-DEVS Quantization, AI and Simulation," AIS 2000, pp. 203-208, Tucson, AZ, March 2000.

[15] James J. Nutaro, Bernard P. Zeigler, R. Jammalamadaka, S. Akerkar. "Discrete Event Solution of Gas Dynamics within the DEVS Framework". International Conference on Computational Science , pp. 319-328, 2003.

[16] Jean-Sébastien Bolduc and Hans Vangheluwe. Mapping ODEs to DEVS: Adaptive Quantization. *Summer Computer Simulation Conference*, pp. 401-407. Montréal, Canada. 2003.

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

[18] G. Christen, A. Dobniewski, G. Wainer. "Modeling State-Based DEVS Models CD++". *MGA, Advanced Simulation Technologies Conference 2004.* Arlington, VA. U.S.A. 2004.

[19] L. R. Petzold. *A description of DASSL: A differential/algebraic system solver*, in IMACS Trans. Scientific Computing Vol. 1, R. S. Stepleman et al., eds., North-Holland, Amsterdam, 1993, pp. 65-68.

[20] K. E. Brenan, S. L. Campbell and L. R. Petzold. *Numerical Solution of Initial-Value Problems in Differential Algebraic Equations*, Elsevier, North Holland, New York, N.Y., 1989.

[21] G. Wainer, B. Zeigler. "Experimental results of Timed Cell-DEVS quantization". *Proceedings of AIS'2000 Artificial Intelligence, Simulation and Planning*. Tucson, Arizona. U.S.A. 2000.