Proceedings of the 2018 Winter Simulation Conference
M. Rabe, A.A. Juan, N. Mustafee, A. Skoogh, S. Jain, and B. Johansson, eds.

BUILDING PARTIAL DIFFERENTIAL EQUATIONS MODELS USING CELL-DEVS

Gabriel Wainer
Cristina Ruiz-Martín

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

Rodrigo Castro

Departamento de Computación
FCEyN, UBA, and ICC, CONICET,
Ciudad Universitaria, Pabellón 1,
C1428EGA, Buenos Aires, ARGENTINA

ABSTRACT

The study of complex systems usually requires hybrid simulations because they have components that are continuous in nature and other that are discrete. It has been proved that DEVS is s a common denominator to combine different Modeling and Simulation methodologies (such as Petri Nets, Cellular Automata, Modelica etc.). We present a cellular model solution to solve PDEs as an extension of classical numerical methods combined with the Cell-DEVS formalism. We explain how to use Cell-DEVS to solve PDEs, focusing on two examples: the PDEs of a Heat Diffusion Process solved with the Method of Lines, and the Shallow Water Equations solved using the Lax-Wendroff method. We will discuss the advantages of solving PDEs with Cell-DEVS, in particular its integration with other hybrid models.

1 INTRODUCTION

Researchers and engineers use different methods to model systems, usually based on the nature of the problem to solve. Traditionally, discrete systems are modeled and studied with discrete methods such as DEVS, Petri Nets, Markov Chains, etc. To model continuous systems, Partial Differential Equations (PDEs), Systems Dynamics (which many times also involves PDEs), and many other methods are used. Nevertheless, most complex Systems have some components that are better represented as discrete models, and other that are better modeled as continuous models. Hybrid modeling and simulation can help in dealing with these issues.

Now, how do we solve continuous models using PDEs? This is not trivial, and a wide variety of numerical methods has been proposed. These methods are used to discretize the space or time (which are continuous in nature) and to produce a new system of Ordinary Differential Equations (ODEs) equivalent to the original PDEs. This new system of ODEs is, in principle, easier to solve. These equations are usually solved using classical methods and simulation software. Using Cell-DEVS offers different advantages, which have been discussed in different research articles (Wainer 2002; Wainer 2009; Wainer and Castro 2010). First, we have asynchronous model execution, which, as showed in, results in improved simulation execution times. Timing constructions permit defining complex conditions for the cells in a simple fashion. As DEVS models are closed under coupling, seamless integration with other types of models in different formalisms is possible. The independent simulation engines permit these models to be executed interchangeably in single-processor, parallel or real-time simulators.

Our goal is to show the applicability of Cell-DEVS (an extension to DEVS that allows implementing cellular models with timing delays) to solve such equations. We leverage the space-explicit semantics of Cell-DEVS to encode the space discretization required in all PDE problems. This approximation uses a hybrid method to model and simulate the system. We define the behavior of the system using PDEs, we transform it into a discrete system using numerical approximations such as such as the Method of Lines

(MOL) or Lax-Wendroff, and we simulate it using a discrete event specification (in our case, the Cell-DEVS formalism) and a discrete event simulator (in our case, the CD++ simulator, Wainer (2002)).

To exemplify how Cell-DEVS and CD++ can be used to simulate systems modeled using PDEs we show two case studies. First, we model and simulate a Heat Diffusion Process using MOL. Then, we model and simulate the Shallow Water equations using Lax-Wendroff.

The structure of the paper is as follows. In section 2, we present the background related to our research. We summarize DEVS, Cell-DEVS, MOL, and Lax-Wendroff. In section 3, we present the first case study: simulating a Heat Diffusion Process using MOL and Cell-DEVS. In section 4, we present the second case study: simulating the Shallow Water equations using Lax-Wendroff and Cell-DEVS. Finally, in section 5, we summarize the conclusions of this work.

2 BACKGROUND

Modeling Complex System using hybrid simulation allows combining components that are discrete in nature and others that are continuous. However, it is not trivial how to integrate these models. In (Mustafee et al. 2015), the authors distinguished between hybrid simulation and hybrid M&S studies based on the techniques applied and the stage of the modeling and simulation process where these different techniques were applied. According to Powell and Mustafee (2014), *hybrid simulation* refers to the use of multiple M&S techniques in the model implementation stage and *hybrid M&S study* refers to the application of methods and techniques from different disciplines (other than M&S) such as operations research, systems engineering and computer science to one or more stages of a simulation study. In a more recent study (Mustafee et al. 2017), the authors discuss that there is not a convergence on definitions, although they highlight that most of the studies refer to the use of at least two commonly used simulation techniques: Discrete-Event Simulation, Dynamic Systems, and Agent-Based Simulation. Here, when we refer to hybrid simulation, we refer to the last definition. We combine Partial Differential Equations (usually used in Dynamic Systems) with Discrete-Event Simulation.

Many works have demonstrated that DEVS is an appropriate formalism to be used as a common denominator when using different modeling and simulation methodologies formalism, and Zeigler, Praehofer, and Kim (2000) demonstrated that DEVS is a common denominator for Discrete-Event systems Specifications. Wainer and Giambiasi (1996) introduced a transformation graph from Cellular Models to DEVS. Jacques and Wainer (2002) showed how to model Petri Nets using DEVS. Urquia et al. (2012) developed libraries to integrate the object-oriented modeling language Modelica and DEVS Graphs (a specific notation to define DEVS models) and in (D'Abreu and Wainer 2005; D'Abreu and Wainer 2006; Wainer and D'Abreu 2015) we introduced an architecture to model continuous and hybrid systems with Modelica and translate them into DEVS for simulation. We also introduce the algorithms for a Modelica compiler with DEVS and CD++ toolkit. Petriu and Wainer (2004) developed a Layered Queuing Network (LQN) (used for performance analysis of software systems) library and we mapped it into DEVS using the CD++ toolkit. Zheng and Wainer (2003) built a library of finite state machine (Moore machines) for DEVS. Mehta and Wainer (2005) explained how to simulate mixed signal Hardware Description Language models using DEVS. Saadawi and Wainer (2004) analyzed complex continuous physical systems using two-dimensional Cell-DEVS models instead of using traditional methods such as Finite Elements and Finite Differences.

Because PDEs are traditionally solved using Classic numerical methods, we propose to integrate these methods within a DEVS simulation to solve the equations. It is worth noting that several particular difficulties can arise when numerically solving the sets of resulting ODEs depending on the class of PDE at hand (parabolic, hyperbolic, and elliptic). Each class exhibits its own particular challenge, including stiff behavior, marginal stability, presence of travelling shock waves, etc. There exist a myriad of numerical methods available, and a wrong choice of method for a given type of problem can result in unaffordable simulation times. General–purpose PDE software is still a promise, and it is still required to invest time to understand the intricacies of each problem (Cellier and Kofman 2006).

Being DEVS a common denominator that allows us to integrate different M&S methodologies, we find it useful to solve PDEs using DEVS. Here, we explain how to solve PDEs using an extension of DEVS (the Cell-DEVS formalism) and two different numerical methods: MOL and Lax-Wendroff.

2.1 DEVS and CELL-DEVS

The DEVS formalism (Zeigler, Praehofer, and Kim 2000) provides a framework to develop hierarchical models in a modular way, allowing model reuse and thus, reducing development time and testing. A DEVS model is defined as a black box with a state and a duration for that state. When state duration time elapses, an output event is sent, and an internal transition takes place to change the model state. A state can also change when an external event is received. Then, a DEVS model is defined by describing the set of states the model goes through, the internal and external transition functions, the output function and the state duration function. These single models are called atomic. DEVS atomic models can be put together by linking the outputs of a model to inputs of other models to form coupled models.

Atomic models define the behavior of the system. The formal definition of an atomic model is as follows:

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

X: is the set of input events.
Y: is the set of output events.
S: is the set of sequential states.
$\mathrm{ta}: \mathrm{S} \rightarrow \mathbb{R}_0^+ \cup \infty$ is the time advance function that determines the time until the next internal transition.
$\delta_{\mathrm{ext}}: \mathrm{QxX} \rightarrow \mathrm{S}$ is the external transition function that determines the next state when external events arrive, where $\mathrm{Q} = \{(\mathrm{s}, \mathrm{e}) | \mathrm{s} \in \mathrm{S}, 0 \leq \mathrm{e} \leq \mathrm{ta}(\mathrm{s})\}$ } and e is the elapsed time since the last state transition.
$\delta_{\mathrm{int}}: \mathrm{S} \rightarrow \mathrm{S}$ is the internal transition function that determines the state transition of the model when the time advance is consumed.
$\lambda: \mathrm{S} \rightarrow \mathrm{Y} \cup \emptyset$ is the output function that determines the output of the model based on its current state.

Coupled models are defined connecting multiple DEVS models (either coupled or atomic) linking the models inputs and outputs. A coupled model is defined as:

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

X: Is the set of input events.
Y: Is the set of output events.
D: Is the set of the names of the sub-components
$\{\mathrm{M_i}\}$: Is the set of sub-components where $\mathrm{i} \in \mathrm{D}$. Each $\mathrm{M_i}$ is a DEVS model (either atomic or coupled)
$\mathrm{I_i}$: is the set of influences of the model i, for each $\mathrm{i} \in \mathrm{D}$
$\mathrm{Z_{i,j}}$: is a function that translates i-to-j output-input, for each $\mathrm{j} \in \mathrm{I_i}$

Select is a tiebreaker function that decides whether the internal or external function has priority when a conflict exists.

Based on the DEVS formalism, Cell-DEVS defines a cell space as DEVS atomic models. A Cell-DEVS atomic model is defined as follow (Wainer and Giambiasi 2002; Wainer 2009):

$$\text{Cell\_DEVS AM} = < \mathrm{X}, \mathrm{Y}, \mathrm{I}, \mathrm{S}, \theta, \mathrm{E}, \mathrm{delay}, \mathrm{d}, \delta_{\mathrm{ext}}, \delta_{\mathrm{int}}, \tau, \lambda, \mathrm{D} >$$

X: Is the set of input events.
Y: Is the set of output events.
I: Represents the model's modular interface.
S: Is the set of sequential states for the cell.
θ: Is the set of the cell's state variables.
E: Is the set of states for the input events to compute the future state.
Delay: is the type of delay in the cell to send its value to the neighbors. The delay can be transport or inertial. In transport delays, the future value will be added to a queue sorted by output time. In inertial delays, any previously scheduled output values will be preempted and only the new one will be scheduled.
d: Is the delay for the cell.
δ_{int}: Is the internal transition function.
δ_{ext}: Is the external transition function. This function activates the local computation function (τ)
τ: Is the local computation function (i.e. the rules to calculate the next state)
λ: Is the output function.
D: Is the state's duration function.
Once the behavior of each cell is defined, the cell space is built as a Cell-DEVS coupled model:

$$\text{Cell\_DEVS CM} = < \text{Xlist}, \text{Ylist}, \text{I}, \text{X}, \text{Y}, \text{n}, \{t_1, \dots, t_n\}, \text{N}, \text{C}, \text{B}, \text{Z}, \text{select} >$$

Xlist: Is the input coupling list.
Ylist: Is the output coupling list.
I: represents the definition of the interface for the modular model.
X: Is the set of external input events.
Y: Is the set of external output events.
n: Is the dimension of the cell space.
$\{t_1,...,t_n\}$: Is the number of cells in each of the dimensions.
N: Is the neighborhood set. It defines the connections between cells.
C: Is the cell space. The cell space is finite.
B: Is the set of border cells. The border cells may have a different neighborhood or the space can be "wrapped" (i.e. the cells in one border are connected with the ones in the opposite border)
Z: Is the translation function to define the external and internal coupling of cells.
select: Is the tie-breaking function for simultaneous events.

To simulate Cell-DEVS models, we use the CD++ simulator (Wainer 2002; Wainer 2009). This tool simulates both DEVS and Cell-DEVS models. It is an open-source environment that runs either in standalone (single CPU) or in parallel mode (over a network of machines). In CD++, the local transition function is defined as a set of rules. They are implemented following the CD++ high-level language with the form:

```
rule: {POSTCONDITION} DELAY {PRECONDITION}
```

This language indicates that when the `PRECONDITION` is satisfied, the state of the cell will change to the designated `POSTCONDITION`, whose computed value will be transmitted to other components after consuming the `DELAY`. If the precondition is false, the next rule in the list is evaluated.

2.2 The Method of Lines (MOL) to Solve Partial Differential Equations (PDE)

The Method of Lines (MOL) is a technique used to transform a partial differential equation (PDE) into an equivalent set of ordinary differential equations (ODEs) (Carver and Hinds 1978; Schiesser 1991).

If we have an unknown function u(x(1), … , x(k), t), and a PDE over this function, we can get an approximation of the solution of the equation (for specific range intervals) by applying the MOL method.

The MOL consists of a three-step process: (1) Discretize the variables, (2) Make an approximation of the differential equations and (3) Rewrite the equations.

2.2.1 Discretize the Variables

Considering the ranges where we want a solution for the PDE, the first step is to divide them into regular intervals. For each variable x(i), we define the regular intervals with the objective of transforming the a PDE in a set of ODEs over t.

We divide each range $\left[x_0^{(i)}, x_f^{(i)}\right]$ into n-1 intervals of size $\Delta x^{(i)}$, where: $\Delta x^{(i)} = \frac{x_f^{(i)} - x_0^{(i)}}{n-1}$

Thus n equidistant points $\{x_0^{(i)}, \ldots, x_{n-1}^{(i)}\}$ are obtained, separated by $\Delta x^{(i)}$ (where $x_{n-1}^{(i)}$ is equal to $x_f^{(i)}$).

Using this step, we discretize each x(i) variable and obtain a regular grid.

In the following steps, we assume that we only have a single x variable (the unidimensional case) to simplify the notation. The process can be generalized to more variables.

2.2.2 Make an Approximation of the Differential Equations

Using the definition of a first-order PDE and a constant small step h > 0 a PDE is equivalent to the *forward difference* that holds when h→0:

$$\frac{\delta u}{\delta x}(x,t) \approx \frac{u(x+h,t) - u(x,t)}{h}$$

Similarly, we can also obtain the *backward difference* and the *central difference* as follows:

$$\frac{\delta u}{\delta x}(x,t) \approx \frac{u(x,t) - u(x-h,t)}{h}$$
$$\frac{\delta u}{\delta x}(x,t) \approx \frac{u(x+h,t) - u(x-h,t)}{2h}$$

In general, using approximations based on Taylor polynomial series, we can rewrite exactly each of the approximations above. For example, the forward difference can be expresses as:

$$\frac{\delta u}{\delta x}(x,t) = \frac{u(x+h,t) - u(x,t)}{h} + R_1(x,t)$$

Because $R_1(x,t)$ is never calculated, it must be analyzed asymptotically, and is usually represented with the O notation. When $R_1(x,t)$ is a linear function of h, $R_1(x,t)$ is represented with O(h) meaning it is a first order approximation method.

The central difference is in fact a second order method, and the approximation error is represented as $O(h^2)$. As we have h<1, a second order method dominated by h^2 provides a better approximation than a first order method with error dominated by h. Using the Taylor polynomial approximates of degree n, we can get approximations of higher orders (although it could bring up undesired side effects).

2.2.3 Rewrite the Equations

The final step is to replace x by one of the approximations calculated in the previous step to transform the PDE into a set of ODEs. We obtain one equation for each x_i of an interval.

For example, using the forward difference, we obtain the following ODE for each *i*-th interval: $u_i'(t) = f(t, u_i(t), u_{i+1}(t))$. Because the approximation itself is not a function of t, we can rewrite the equation as follows: $u_i'(t) = f(u_i(t), u_{i+1}(t))$. These equations still require defining border conditions to deal with values at the extremes of the discretization range.

2.3 Lax-Wendroff Methods to Solve Partial Differential Equations (PDE)

Lax-Wendroff (Lax and Wendroff 1960) is a numerical method based on finite differences particularly suited for the solution of hyperbolic PDEs. The method defines a square grid with a vector solution in the middle of each cell as shown in Figure 1.

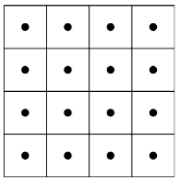


Figure 1: Solution of the method in the center of each cell.

In the middle of each cell (i,j) and at the time step n, we have vector $U^n_{i,j}$ with 3 dimensions. Each time step has two phases. In the first phase, we define the values for the midpoints of cells' borders (see Figure 2) at time *n +1/2* using the following equations:

$$U^{n+1/2}_{i+1/2,j} = \frac{1}{2}\left(U^n_{i+1,j} + U^n_{i,j}\right) - \frac{\Delta t}{2\Delta x}\left(F^n_{i+1,j} - F^n_{i,j}\right)$$
$$U^{n+1/2}_{i,j+1/2} = \frac{1}{2}\left(U^n_{i,j+1} + U^n_{i,j}\right) - \frac{\Delta t}{2\Delta y}\left(G^n_{i,j+1} - G^n_{i,j}\right)$$

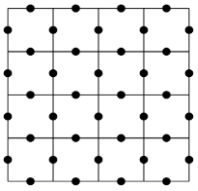


Figure 2: Values in the middle of each cell border.

In the second phase, we calculate the values for the points in the middle of the cell (see Figure 1) for the next time step as follows: $U^{n+1}_{i,j} = U^n_{i,j} - \frac{\Delta t}{\Delta x}\left(F^{n+1/2}_{i+1/2,j} - F^{n+1/2}_{i-1/2,j}\right) - \frac{\Delta t}{\Delta y}\left(G^{n+1/2}_{i,j+1/2} - G^{n+1/2}_{i,j-1/2}\right)$

3 MOL WITH CELL-DEVS: APPLICATION TO A HEAT DIFFUSION PROCESS

To exemplify how to solve a PDE by applying MOL with Cell-DEVS and the CD++ simulator, we will use a case study of a unidimensional heat diffusion process. The results obtained with the model will represent the values of the unknown function u(x(1), . . . , x(k), t) over the discretized grid.

For the unknown function u(x(1), . . . , x(k), t), we will define an n-dimensional Cell-DEVS model. The number of cells in each dimension i will be the number of discrete values for the variable x(i) once the MOL has been applied. We define the rules that govern the Cell-DEVS models in a way that the values on the Cell-DEVS grid are the solutions for the equation in the points defined using the MOL. For example, for the function u(x,t), the cell space will be bi-dimensional as shown in Figure 3. Note that even when the original problem has only one dimension (along the x-axis), we designed a two-dimensional grid for convenience in order to explicitly visualize the time evolution of the system along the second time axis. Yet, the physical system modeled exists only along the x-axis (e.g. heat diffusing along a metal rod).

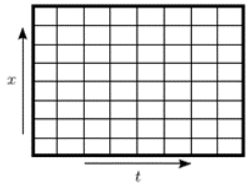


Figure 3: Cell-DEVS grid for the unknown function u(x,t).

To solve the set of ODEs, a 4th order Runge-Kutta method (RK4) is used as the integration method.

3.1 Model Definition

A heat diffusion process can be defined by the equation $\frac{\delta u}{\delta x}(x,t) = \sigma \frac{\delta^2 u}{\delta t^2}(x,t)$ and the border conditions $u(x,0) = 100\sin(\frac{\pi x}{c})$, $u(0,t) = 0$, $u(c,t) = 0$.

To study the heat diffusion process on a bar of length 1, we apply heat in the middle, which is expected to diffuse to the borders where the temperature is maintained at 0 degrees. We fix c=1 and $\sigma = 1$. The dynamics of this system represents the heat dissipation over time.

This problem can be studied in the interval ranges: $x \in [0,1]$, $t \in [0,1]$

We then apply the MOL approach, using the Taylor polynomial approximation of order 2, to transform the PDE into a set or ODEs following the steps described in section 2.2. If we set the border conditions, we obtain the following set of ODEs:

$$\frac{\partial u}{\partial t}(x_i,t) \cong \frac{u(x_{i+1},t)-2u(x_i,t)+u(x_{i-1},t)}{\Delta x^2},\ 0<i<n\text{-}1$$
$$u(x_0,t) = 0\,, u(x_{n-1},t) = 0$$

With this approximation, the only variable remaining unknown is t. In this specific case, the cell space is bi-dimensional as shown in Figure 3. The size of the cell space is n x m, being n the number of discrete values for x and m the number of discrete values for t. For readability purposes, we rewrite the ODEs using the following notation: $u(x_i,t) \triangleq u_i(t)$ and $\Delta x \triangleq h$.

$$u_i'(t) = \frac{u_{i+1}(t) - 2u_i(t) + u_{i-1}(t)}{h^2}, 0 < i < n-1$$
$$u_0(t) = 0\,, u_{n-1}(t) = 0$$

Taking into account that $\forall i \in (0, n-1)$ the following holds: $u_i'(t) = g\big(t, u_{i-1}(t), u_i(t), u_{i+1}(t)\big)$. After algebraic manipulations the Runge-Kutta factors turn out to become all equal:

$$k_{1,i} = k_{2,i} = k_{3,i} = k_{4,i} = \frac{u_{i+1}(t) - 2u_i(t) + u_{i-1}(t)}{h^2}$$

and the equation $u_i(t_{j+1}) = u_i(t_j) + \frac{1}{6}\Delta t(\, k_{1,i} + 2k_{2,i} + 2k_{3,i} + k_{4,i})$ is simplified to $u_i(t_{j+1}) = u_i(t_j) + \Delta t\; k_{1,i}$.

Finally, the cells in the Cell-DEVS model are defined as: $c(i,j) = u_i(t_j)$

In this very particular case study, the MOL combined with RK4 assumes the form of a first order Euler Method. Now, using CD++ notation, the rules for the Cell-DEVS model can be defined as in Figure 4. Note that Macro(N), Macro(M), Macro(DX) and Macro(DT) represent calculations for n, m, Δx and Δt respectively. The first two rules in Figure 4 represent the border conditions of the system. The third rule implements the MOL method to solve the heat equation.

```
...
rule: {0} 1 {cellpos(0) = (#Macro(N)-1) and cellpos(1)=0}
rule: {100*sin(PI*cellpos(0)*#Macro(DX))} 1 {cellpos(1)=0}
rule: {(0,-1)+(#Macro(DT)/(#Macro(DX)*#Macro(DX)))*((-1,1)-2*(0,-1)+(1,-1))} 1 {t}
...
```

Figure 4: Cell-DEVS rules to solve heat equation: MOL combined with fourth order Runge-Kutta.

The neighborhood cells will depend on the type of finite difference used to make the approximation. In this example, we used the second-order central difference. It uses the values $u_{i-1}(t_{j-1}), u_i(t_{j-1}), u_{i+1}(t_{j-1})$. Therefore, considering the cell space defined in Figure 3, the required neighborhood is N={(-1,-1),(0,-1),(1,-1),(0,0)}. It is graphically represented in Figure 5.

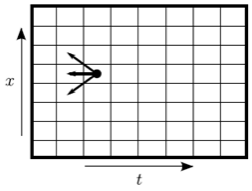


Figure 5: Neighborhood definition for Heat equation using MOL with the central difference.

As of the numerical stability of the method, and for the resulting Euler method, the solution will be stable only if the following inequality holds $r = \frac{\Delta t}{\Delta x^2} < 0.5$.

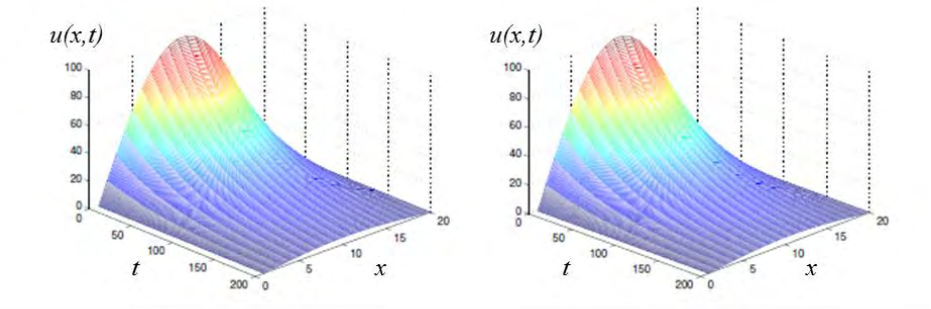
Thus, the cell space size (n and m dimensions) must satisfy this condition.

3.2 Simulation Results

We have simulated the model using the following parameters to solve the Heat equation in the whole bar and to satisfy the stability condition: $(n, m) = (20{,}200)$; $(t_0, t_f) = (0{,}0.3)$; $(x_0, x_f) = (0{,}1)$

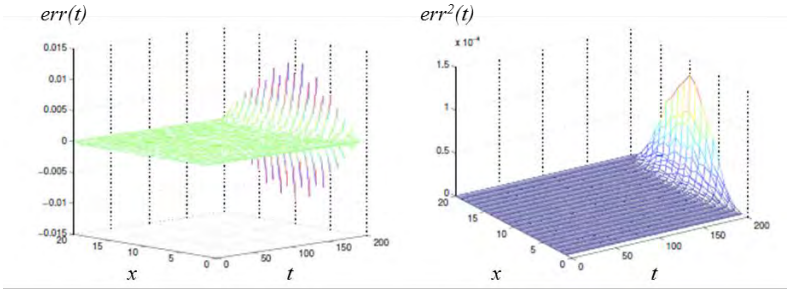
Therefore, the parameters are $\Delta t \approx 0.00125$; $\Delta x \approx 0.0526$, yielding $r \approx 0.4535 < 0.5$. To verify the results obtained with Cell-DEVS and the CD++ toolkit, we provide a comparison against an implementation of the same system in the MATLAB toolkit. In Figure 6, we show results obtained with both tools. Figure 7 shows the error and the quadratic error between CD++ and MATLAB results.

With this first case study we have exemplified how Cell-DEVS and CD++ can be used to model and simulate a distributed continuous system formulated using partial differential equations, obtaining solutions comparable to traditional toolkits.



a) b)

Figure 6: (a) Results obtained using MATLAB (b) Results obtained using Cell-DEVS and CD++.



a) b)

Figure 7: (a) Error and (b) Quadratic error of the results obtained using Cell-DEVS and CD++ compared to the solution provided by MATLAB.

4 LAX-WENDROFF WITH CELL-DEVS: THE SHALLOW WATER EQUATIONS

To exemplify how to solve a PDE applying Lax-Wendroff using Cell-DEVS and CD++ simulator, we will use as a case study the shallow water equations.

Shallow water models are based on the equations of conservation of mass and conservation of linear momentum. The independent variables are the time (t) and the space (x,y). The dependent variables are the height or depth of the fluid (h) and the fluid velocity in the space (u,v). Considering that the force affecting the fluid is the gravity results in the following system of PDEs:

$$\frac{\partial h}{\partial t} + \frac{\partial (uh)}{\partial x} + \frac{\partial (vh)}{\partial y} = 0$$

$$\frac{\partial (uh)}{\partial t} + \frac{\partial (u^2 h + \frac{1}{2} g h^2)}{\partial x} + \frac{\partial (uvh)}{\partial y} = 0$$

$$\frac{\partial (vh)}{\partial t} + \frac{\partial (uvh)}{\partial x} + \frac{\partial (v^2 h + \frac{1}{2} g h^2)}{\partial y} = 0$$

We can rewrite the equation compactly introducing the following vectors:

$$U = \begin{pmatrix} h \\ uh \\ vh \end{pmatrix}$$

$$F(U) = \begin{pmatrix} uh \\ u^2h + \frac{1}{2}gh^2 \\ uvh \end{pmatrix}$$

$$G(U) = \begin{pmatrix} vh \\ uvh \\ v^2h + \frac{1}{2}gh^2 \end{pmatrix}$$

The compact PDE for shallow water results in $\frac{\partial U}{\partial t} + \frac{\partial F(U)}{\partial x} + \frac{\partial G(U)}{\partial y} = 0$

The Lax-Wendroff method can be used to solve this PDE. We set the border conditions as h=1 and u=v=0. In order to simulate a disturbance we add a bidimensional perturbation with the shape of a Gauss bell. Using CD++, we define the rules that implement the Lax-Wendroff method to calculate the dependent variables: h, v and u. A sketch of these rules is presented in Figure 8.

```
...
[zone_H] % H=0 Ux=5 Vy=1 HalfStep=7
rule: { (0,0,0) - (#macro(dt) / #macro(dx)) * ((0,-1,5) - (-1,-1,5)) - (#macro(dt) /
#macro(dy)) * ((-1,0,1) - (-1,-1,1)) } #macro(delay) { (0,0,7) = 1 }
rule: { (0,0,0) } #macro(delay) { t }
[zone_U] % U=0 Hx=7 Ux=6 Hy=4 Uy=3 Vy=2 HalfStep=8
rule: { (0,0,0) - (#macro(dt) / #macro(dx)) * ((((0,-1,6) * (0,-1,6)) / (0,-1,7) +
#macro(gravity) / 2 * ((0,-1,7) * (0,-1,7))) - (((-1,-1,6) * (-1,-1,6)) / (-1,-1,7) +
#macro(gravity) / 2 * ((-1,-1,7) * (-1,-1,7)))) - (#macro(dt) / #macro(dy)) * (((-1,0,2) *
(-1,0,3) / (-1,0,4)) - ((-1,-1,2) * (-1,-1,3) / (-1,-1,4))) } #macro(delay) { (0,0,8) = 1 }
rule: { (0,0,0) } #macro(delay) { t }
[zone_V] % V=0 Hx=8 Ux=7 Vx=6 Hy=5 Vy=3 HalfStep=9
rule: { (0,0,0) - (#macro(dt) / #macro(dx)) * (((0,-1,7) * (0,-1,6) / (0,-1,8)) - ((-1,-
1,7) * (-1,-1,6) / (-1,-1,8))) - (#macro(dt) / #macro(dy)) * ((((-1,0,3) * (-1,0,3)) / (-
1,0,5) + #macro(gravity) / 2 * ((-1,0,5) * (-1,0,5))) - (((-1,-1,3) * (-1,-1,3)) / (-1,-
1,5) + #macro(gravity) / 2 * ((-1,-1,5) * (-1,-1,5)))) } #macro(delay) { (0,0,9) = 1 }
rule: { (0,0,0) } #macro(delay) { t }
...
```

Figure 8: Sketch of rules for the Cell-DEVS model to solve shallow water equations.

In Figure 9, we present the simulation results at three consecutive time steps. We compare the results obtained using the CD++ simulator against results obtained with MATLAB. In the figure, we can appreciate a qualitative correspondence between the experiments. This shows the feasibility of using CD++ to solve this class of PDEs.

5 CONCLUSIONS

We have proposed Cell-DEVS as an alternative to solve PDEs using classic numerical methods. The aim is being able to combine continuous models with discrete models. We have exemplified how to use Cell-DEVS and the CD++ simulator using two continuous systems and two different numerical methods as an example. We first studied a heat diffusion process modeled with PDEs. To solve this system, we applied the MOL. The second example was a model of the behavior of water surface after the shallow water equations solved using Lax-Wendroff.

Solving PDEs with a discrete M&S formalism, particularly with DEVS, has the advantage that it can be easily integrated with other models developed using different techniques such as Petri Nets, Cellular automata, etc. As mentioned in the background section, DEVS has been proved as a common denominator for different M&S approaches.

Solving PDEs with DEVS is particularly useful when studying complex systems. Complex systems usually require hybrid modeling and simulation, understood as a combination of different techniques to

study the problems at hand. In these cases, it is important to resort to formalisms (such as DEVS) that allow integrating hybrid components of the model and simulate them together.

Future research along these lines include a detailed performance comparison of CD++ simulations for PDEs against well-established simulation toolkits in the domain.

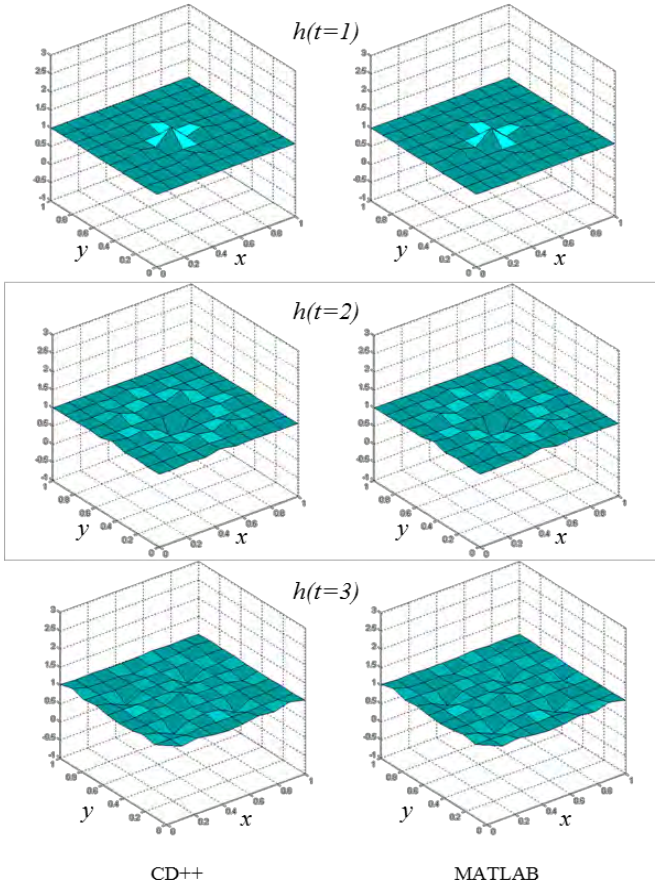


Figure 9: Qualitative comparison between CD++ (Left) and MATLAB (Right).

ACKNOWLEDGMENTS

This work has been developed by numerous authors, in particular Matías Nitsche and Ignacio Vivona (at the Computer Science Department, FCEyN, UBA) developed many of the models introduced.

REFERENCES

Carver, M B., and H.W. Hinds. 1978. "The Method of Lines and the Advective Equation." *SIMULATION* 31 (2). Sage PublicationsSage CA: Thousand Oaks, CA: 59–69. doi:10.1177/003754977803100205.

Cellier, F E., and E Kofman. 2006. *Continuous System Simulation*. Boston: Springer Science & Business Media. doi:10.1007/0-387-30260-3.

D'Abreu, M C., and G Wainer. 2005. "Experimental Results on the Implementation of Modelica Using DEVS Modeling and Simulation." In *Procedings of IEEE/ACM MASCOTS*. Atlanta, GA.

D'Abreu, M C., and G Wainer. 2006. "A Bond-Graph Mapping Mechanism for M/CD++." In *Proceedings of the SCS Summer Computer Simulation Conference*. Calgary, AB, Canada.

Jacques, C, and G A Wainer. 2002. "Using the Cd++ DEVS Toolkit to Develop Petri Nets." In *Proceedings of the SCS Conference*. San Diego, California.

Lax, P, and B Wendroff. 1960. "Systems of Conservation Laws." *Communications on Pure and Applied Mathematics* 13 (2). Wiley-Blackwell: 217–37. doi:10.1002/cpa.3160130205.
Mehta, S, and G Wainer. 2005. "DEVS for Mixed-Signal Modeling Based on VHDL." In *Proceedings of the 2005 DEVS Integrative M&S Symposium, Spring Simulation Conference*. San Diego, California.
Mustafee, N, S Brailsford, A Djanatliev, T Eldabi, M Kunc, and A Tolk. 2017. "Purpose and Benefits of Hybrid Simulation: Contributing to the Convergence of Its Definition." In *2017 Winter Simulation Conference (WSC)*, 1631–45, edited by W.K.V. Chan et al., Piscataway, New Jersey:IEEE.
Mustafee, N, M. Sahnoun, A. Smart, P. Godsiff, D.Baudry, and A. Louis. 2015. "Investigating Execution Strategies for Hybrid Models Developed Using Multiple M&S Methodologies." In *Proceedings of the 48th Annual Simulation Symposium*, 78–85.
Petriu, D.B., and G. Wainer. 2004. "A DEVS Library for Layered Queuing Networks." In *Proceedings of the SCS 1st Mediterranean Multiconference on Modeling and Simulation*. Genoa, Italy.
Powell, J, and N Mustafee. 2014. "Soft or Approaches in Problem Formulation Stage of a Hybrid M&S Study." In *Proceedings of the 2014 Winter Simulation Conference*, 1664–75, edited by A.Tolk et al., Piscataway, New Jersey:IEEE.
Saadawi, H, and G Wainer. 2004. "Modeling Complex Physical Systems Using 2D Finite Element Cell-DEVS." In *Advanced Simulation Technologies Conference 2004*.
Schiesser, W. E. 1991. *The Numerical Method of Lines : Integration of Partial Differential Equations*. San Diego, California: Academic Press.
Urquia, A, C Martin-Villalba, M Moallemi, and G. Wainer. 2012. "DEVS Graph In Modelica For Real-Time Simulation." *Proceedings 26th European Conference on Modelling and Simulation*, 1: 157–63
Wainer, G. 2002. "CD++: A Toolkit to Develop DEVS Models." *Software: Practice and Experience* 32 (13): 1261–1306. doi:10.1002/spe.482.
Wainer, G. 2009. *Discrete-Event Modeling and A Practitioner's Approach*. Boca Raton: CRC Press.
Wainer, G, and R Castro. 2010. "A Survey on the Application of the Cell-DEVS Formalism." *J. Cellular Automata* 5: 509-524.
Wainer, G, and M C. D'Abreu. 2015. "Using a Discrete-Event System Specifications (DEVS) for Designing a Modelica Compiler." *Advances in Engineering Software* 79. Elsevier Ltd: 111–26.
Wainer, G, and N Giambiasi. 1996. "CELL-DEVS Models with Transport and Inertial Delays." In *Proceedings of the SCS European Simulation Symposium (ESS)*. Passau, Germany.
Wainer, G, and N Giambiasi. 2002. "N-Dimensional Cell-DEVS Models." *Discrete Event Dynamic Systems* 12 (2): 135–57. doi:10.1023/A:1014536803451.
Zeigler, B P, H Praehofer, and T G Kim. 2000. *Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems*. San Diego: Academic press.
Zheng, T, and G Wainer. 2003. "Implementing Finite State Machines Using the CD++ Toolkit." In *Proceedings of the 2003 SCS Summer Computer Simulation Conference (Student Workshop)*. Montreal, QC. Canada.

AUTHOR BIOGRAPHIES

GABRIEL WAINER is Professor at the Department of Systems and Computer Engineering at Carleton University. He is a Fellow of the Society for Modeling and Simulation International (SCS). His email address is gwainer@sce.carleton.ca.

CRISTINA RUIZ-MARTIN has obtained a Ph.D. in Industrial Engineering (University of Valladolid, UVa) and Systems and Computer Engineering (Carleton University). She is a Postdoctoral Fellow at SCE, Carleton University. Her email address is cristinaruizmartin@sce.carleton.ca.

RODRIGO CASTRO is Professor at the Departamento de Computación, Facultad de Ciencias Exactas y Naturales, Universidad de Buenos Aires, head of the Simulation Lab, and a researcher at ICC-CONICET. His research interests include modeling, simulation and control of hybrid systems. His email address is rcastro@dc.uba.ar.