1D/2D Cellular Automata Modeling with Modelica

Victorino Sanz† Alfonso Urquia† Alberto Leva‡
† Dpto. Informática y Automática, ETSI Informática, UNED
Juan del Rosal, 16, 28040, Madrid, Spain
{vsanz,aurquia}@dia.uned.es
‡ Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano
Piazza Leonardo da Vinci 32, 20133 Milano, Italy
leva@elet.polimi.it

Abstract

Cellular Automata (CA) can be used to describe dynamic phenomena dependent of the spatial coordinates. This approach exhibits two main advantages: CA models are conceptually simple and can be simulated very efficiently. A new Modelica library named *CellularAutomataLib* is presented. It facilitates describing one- and two-dimensional CA in Modelica, and interfacing these CA models with other Modelica models. Simulation performance and large model support have been highest priority in the design of the library. To achieve these goals, the CA internal description is programmed in C and it is consequently hidden to the modeling environment, which is released from the burden of causalizing and manipulating the millions of equations that typically compose CA models. The library architecture and use are discussed in this manuscript. Two examples illustrate the library use: heat diffusion on a chip and spread of an epidemic disease. *CellularAutomataLib* is freely available at `http://www.euclides.dia.uned.es`.

Keywords: Cellular Automata, Hybrid Models, Modelica

1 Introduction

Cellular Automata (CA) are discrete and dynamic models initially proposed by John Von Neumann for the study of self-reproducing automata [1]. These models are represented as a grid of identical volumes, named cells, that can be in any finite number of dimensions [2]. The state of each cell in the automata is discrete, and it is updated at discrete time steps during the simulation following a transition function or rule. This rule constitutes a function of the current state of the cell and the state of its neighbors, and defines the state of the cell for the next time step [3]. The neighborhood of a cell is usually composed of a selection of its surrounding cells, but not necessarily. It can be defined in different ways, such as the Moore's neighborhood that includes all the surrounding cells; the von Neumann's neighborhood that includes the cells adjoining the four faces of one cell; or the extended von Neumann's that also includes each cell just beyond one of the four adjoining cells [4].

Formally, CA can be defined as a tuple [5]:

$$CA : < T, X, \Omega, S, \delta, Y, \lambda >$$

where T is the time base (isomorphic with $\mathbb{N}$); X is the input set; Ω is the set of all input segments ω (an input segment may be restricted to a domain T, $\omega : T \rightarrow X$); S is the state that is the same for all cells because the cellular space is homogeneous; $\delta : \Omega \times S \rightarrow S$ is the global transition function used to update the state of each cell ($\delta(\omega, s_i) \rightarrow \delta_l(N_i)$, δ_l is the uniform local transition function, s_i is the state of the i-th cell of the grid and N_i is the set of states that correspond to the neighborhood of the i-th cell, usually defined as a set of offsets from i); Y is the output set; and $\lambda : S \rightarrow Y$ is the output function used to observe the state of the automata.

The application of CA in the study of systems is broad and diverse, mainly due to the simplicity of describing these kind of models (i.e., by describing the state of the cell, the initial state of the space and the transition rule) and the computational efficiency of their simulation. They have been used to model systems in medicine [6], architecture [7], chemistry [8], economics [9], biology [10], among many others [11].

The feasibility for describing CA models using the Modelica language was demonstrated by Fritzson [12]. He described the Conway's Game of Life model

in Modelica by representing the cellular space using a matrix of integer numbers. The initial conditions are set using a vector that contains the coordinates of the initially active cells. At discrete times, generated using a *sample* operator, the state of the automata is updated by iterating the whole matrix using two *for* loops, and using the transition function to update each individual cell. The model uses the Moore's neighborhood. In this model, the description of the cellular space and the evaluations of the transition function in each cell are coupled, difficulting its reutilization to describe other automata.

Another approach to describe CA models in Modelica was performed by the authors [13]. The *CellularPDEVS* package, distributed with the DESLib library [14, 15], supports the description of CA using the Parallel DEVS formalism [16, 17]. DESLib is freely distributed under the Modelica License 2, and can be downloaded from the Modelica Association website. The cellular space is represented as coupled Parallel DEVS models, and each cell is described as an atomic Parallel DEVS model. *CellularPDEVS* allows the user to focus on describing the behavior of the cell and the characteristics of the cellular space. The state of each cell corresponds to the state of the atomic Parallel DEVS model and can be represented using an arbitrarily complex Modelica data structure. The transition rule corresponds to the internal transition function of each cell, which can contain any Modelica algorithm. This approach facilitates the description of the model by making the simulation algorithm of the automata transparent to the user.

CellularPDEVS also facilitates the combination of CA models with other Modelica models. Inputs and outputs to the cellular space can be described using the external transition and output functions, respectively. These functions (i.e., internal transition, external transition and output) are used in the DEVS formalism to define the behavior of models. However, the performance and the scalability of this library are not satisfactory. The reasons are twofold. First, the size of CA models is typically in the order of hundreds of thousands and millions of equations. The translation of models with so large number of equations (when even possible) is time consuming and huge executable files are generated (see also discussion in [18]). Second, long event chains are executed to update the CA state. The complete model is reevaluated after executing each event in the event chain, which in most cases is unnecessary, degrading the simulation performance significantly.

A new library, named *CellularAutomataLib*, for describing CA models is presented in this manuscript. The objective of this new library is to preserve the characteristics of *CellularPDEVS*, in terms of facility to describe the behavior of the model and the characteristics of the space, and provide a solution for its main drawbacks (i.e., simulation performance and scalability). *CellularAutomataLib* is not based in the Parallel DEVS formalism. The CA model (i.e., the state of the cell and the transition function) is described as a C data structure (i.e., a C *struct*) and a function. Its simulation algorithm is also directly implemented into several C functions that are called from Modelica by using the external function interface provided by the language [19]. CA models defined in this way are not manipulated by the Modelica tool (Dymola in our case), which produces an smaller simulation code and avoids the reevaluation of the whole model after the treatment of an event in the CA. The user can focus on describing the behavior of the model and not the simulation algorithm. Also, the simulation of CA models automatically displays a graphical animation that is generated using Gnuplot [20]. *CellularAutomataLib* provides interface models that facilitate the connection of CA with other Modelica models. These interface models use user-defined external functions to translate the state of the cell into a standard Modelica data type that can be used in other models, and vice-versa. The library has been developed using Dymola FD1 2013 on an Intel Core i5 2.3GHz machine with 16GB of RAM and running Linux 3.11 x86_64.

The structure of the manuscript is as follows. The architecture of the library and its design principles are detailed in Section 2. The procedure to develop new CA models using the library is described in Section 3. The description of the interface models used to combined CA with other Modelica models is given in Section 4. Two case studies of 2D CA models are presented in Section 5. Finally, some conclusions and future work ideas are given in Section 6.

2 Architecture of the Library

The architecture of *CellularAutomataLib* is shown in Fig. 1. The library is composed of the following models and packages: `License` (description of the license); `User Guide` (library documentation); `Path_gnuplot` (path to Gnuplot binary, used to generate the graphical animation); `Input_Region` model (used as interface between CA models);

DOI 10.3384/ECP14096489

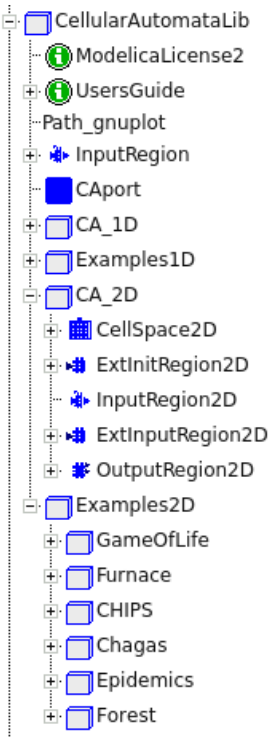


Figure 1: Architecture of *CellularAutomataLib*.

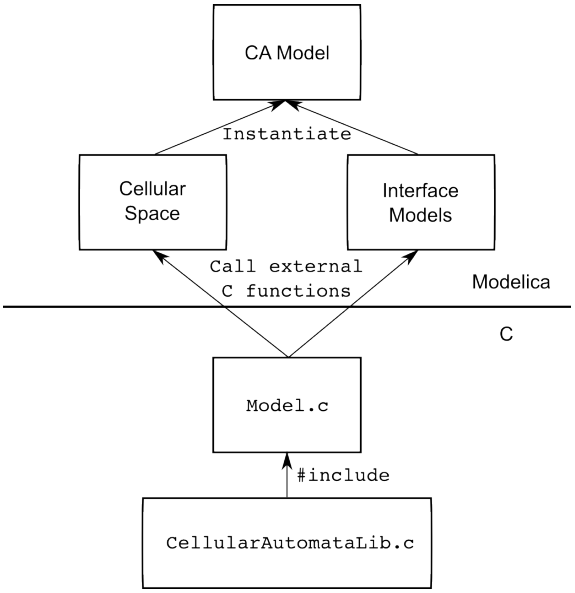


Figure 2: Relationship between Modelica and external C code in *CellularAutomataLib*.

`CAport` connector (used to connect CA models with other models); `CA_1D` package (models to describe 1D CA); `Examples1D` package (examples of 1D CA); `CA_2D` package (models to describe 2D CA); `Examples2D` package (examples of 2D CA).

The library has been implemented using C functions that are called from Modelica functions using the external function interface. The basic functions that perform the simulation of the automata are included in the file `CellularAutomataLib.c`, which should not be modified by the user. These functions include the creation and initialization of the space, the simulation of a step or the reception of an external input, among others. The behavior of the model has to be described as external C code (i.e., `model.c`).

3 Development of New Models

A CA model in *CellularAutomataLib* is composed of one or several *cellular spaces*, that represent the 1D or 2D grid of cells, and some models, named *interface models*, used as interface between cellular spaces or between cellular spaces and other models. Cellular spaces and interface models include functions that call external C functions. These external C functions are used to describe the behavior of the models. The relationship between the external code and the Modelica code is summarized in Fig. 2. In this section the development of new cellular spaces is described. The use of the interface models is described in Section 4.

3.1 Description of New Cellular Spaces

A cellular space in *CellularAutomataLib* is composed of the cellular space model and some functions. The cellular space model is a partial model that describes the one- or two-dimensional space represented by the automata. The parameters of the model are shown in Table 1. The cellular space model includes three replaceable functions: `Create`, that is used to create the cellular space, allocate memory for the cells and set them to the default state; `Initial`, that is used to initialize the cells indicated using the `init_cells` parameter and; `Rule`, that represents the transition function and is used to update the state of the cells at each simulation step.

The behavior of the cellular space is described by redeclaring these functions with functions that call external C functions (cf. Fig. 2). The description of these external functions is detailed below.

At the beginning of the simulation, the cellular space model creates the space using the `Create` function, and initializes the cells included in the `init_cells` parameter. After that, it performs periodic simulation steps every `Tstep`

Table 1: Parameters of cellular space models.

Name	*Description*
`space_nrows`	defines the number of rows of the space (only used in 2D spaces).
`space_ncols`	defines the number of columns of the space.
`neighborhood`	defines the topology of the neighborhood. It contains a list of the relative positions of the neighbors from the center cell.
`n_inputs`	defines the number of inputs received from interface models connected to the automata.
`wrapped_borders`	defines the boundary conditions. In 1D spaces it is either 1 for wrapped or 0 for non-wrapped. In 2D spaces it can be 0 for non-wrapped, 1 for wrapped only north-to-south, 2 for wrapped only east-to-west and 3 for wrapped in all directions.
`Tstep`	defines the interval between two steps in the simulation of the automata.
`Initial_step`	defines the time for performing the first simulation step.
`plot_animation`	defines if the graphical animation is generated (value 1) or not (value 0).
`plot_range`	defines the maximum value of the variable displayed in the animation. Thus, the displayed variable can be in the $[0, plot\_range]$ interval.
`display_delay`	defines a delay in the graphical animation that can be used to improve its visualization, which otherwise could be too fast to be observed.
`init_cells`	defines a list of coordinates of the cells that will be initialized at the beginning of the simulation.
`name`	defines a name for the automata that will be displayed in the graphical animation.

time, starting at `time = Initial_step` (i.e., `sample(Initial_step,Tstep)`).

3.2 Description of External C Functions

In order to facilitate the description of the behavior of new cellular spaces, the library includes a template file (named `draft.c`) that can be used to describe the required external C functions.

Following the formal specification of the automata, the user has to define the state variables that represent the state of the cells (S) and the model behavior (i.e., the transition function δ) by reimplementing the functions included in the `draft.c` file, into a new file (e.g., `model.c`). The time base T is set using the parameters `TStep` and `Initial_step` of the cellular space model. The rest of the elements of the tuple (X, Ω, Y, λ) are defined using the interface models.

As an example, the development of the Rule 30 model described by Wolfram [4] is presented. The transition function for this model is shown in Fig. 3.

The `draft.c` file can be used as a template to describe the behavior of the model. It has been renamed as `wolfram.c` for this example. The state of each cell is defined as an `int` value by modifying the `State` data type in the template. The default value for the cell state will be set using the `DefaultState` function, and so it has to be modified to set the default state to 0. The state of initialized cells will be set using the `InitialState` function, and so it has to be modified to set the initial state to 1. The transition function shown in Fig. 3 has to be implemented by modifying the `transition` function in the template. In order to automatically generate the graphical animation, the `Display` function in the template has to be modified to convert the state of the cell (i.e., the `State` data type) into a `double` value.

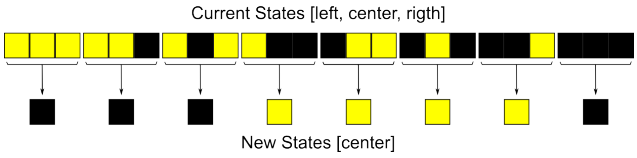


Figure 3: State transitions for the Rule 30 model.

The cellular space for the Rule30 model is described in Modelica by extending the *CellSpace1D* model from *CellularAutomataLib*. The `Create`, `Initial` and `Rule` functions are redeclared using functions that call the external C functions defined in `wolfram.c`. The parameters for the Rule30 model are: `Space_ncols = 20`, `neighborhood = {-1,1}`, `wrapped_borders = 1`, `Tstep = 1`, `Initial_step = 0`, `plot_animation = 1`, `plot_history = 1`, `init_cells = 10`, `name = "Rule 30"`. The graphical

DOI 10.3384/ECP14096489

animation will be automatically generated using Gnuplot if `plot_animation` is set to 1. In this model, state 0 is displayed in black, and state 1 is displayed in yellow. The first 10 steps of simulation for the Rule 30 model are shown in Fig. 4 (the number of step is represented in the vertical axis).

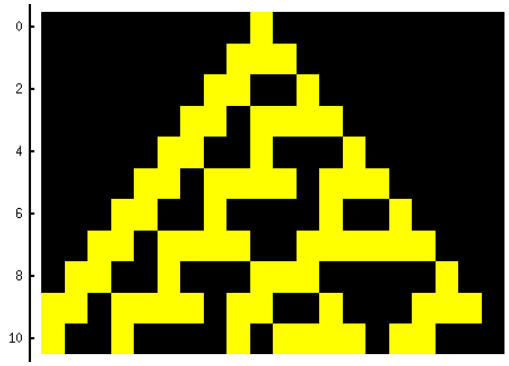


Figure 4: Simulation of the first 10 steps for the Rule 30 model.

4 Interfacing with Other Models

CellularAutomataLib includes several interface models that facilitate the combination of CA with other Modelica models. The inputs of the CA model (X,Ω) are described using *input* and *external input* region models. The outputs of the CA model (Y,λ) are described using the *output* region model. Their behavior and use are detailed below.

4.1 Input Region

Cellular spaces can be combined to increase the modeling functionality of the library. This communication can be performed using the `Input_Region` model. The same model can be used between 1D and 2D spaces.

The combination is performed by translating the state of some cells from one space as inputs for the other. The prototype of the transition function in C includes a vector of the received inputs, in order to allow the user to manage them. Each `Input_Region` has associated an input identifier, set using the parameter `input_id`, that can be used as index for the vector of inputs of the transition function.

The `Input_Region` model has two interface ports: `FROM` and `TO`. These interface ports are used to connect to the involved cellular spaces. The state of the cell $[i,j]|i \in [\texttt{RstartFrom}:\texttt{RendFrom}], j \in [\texttt{CstartFrom}:\texttt{CendFrom}]$, in the `FROM` space, is translated using the `SetInput` function into an input for the cell $[m,n]|m \in [\texttt{RstartTo},\texttt{RstartTo} + (\texttt{RendFrom} - \texttt{RstartFrom})], n \in [\texttt{CstartTo},\texttt{CstartTo} + (\texttt{CendFrom} - \texttt{CstartFrom})]$ in the `TO` space. `RstartFrom`, `RendFrom`, `CstartFrom`, `CendFrom`, `RstartTo` and `CstartTo` are parameters of the model. In 1D spaces, only the column parameters are used (i.e., `CstartFrom`, `CendFrom` and `CstartTo`). An additional parameter, named `column_1D_2D`, allows to use a 1D region as a column, instead of a row, of inputs for a 2D space. The communication is started at `time = comm_start` and is performed every `comm_rate` time.

The function `void SetInput(int Fspace, int Frow, int Fcol, int Tspace, int Trow, int Tcol, int input_id)` from the `draft.c` file can be used to redeclare the `SetInput` of this model.

4.2 External Input Region

Similarly to the `Input_Region` model, the model `ExtInputRegion` can be used to set an input to a region of cells in the automata. In this case, the input is generated using an external signal instead of the state of the cells of other automata.

The model receives an external Real input signal through port `u`, which is used as input for a region of cells in the automata connected to port `TO`. As in the previous interface model, a 2D region is defined by the positions between `Rstart` and `Rend` (for the rows) and `Cstart` and `Cend` (for the columns). In 1D regions only the parameters referring to columns are considered. The input is assigned to the position `input_id` of the vector of inputs which is available for the user in the transition function. The external signal can be observed in the following ways (defined by the parameter `Input_type`), in order to be converted into an input:

- *Quantizer*: the input is set every time the value of the signal changes in a defined value or quantum.
- *Cross_UP*: the input is set every time the value of the signal crosses a defined threshold in the upwards direction.
- *Cross_DOWN*: the input is set every time the value of the signal crosses a defined threshold in the downwards direction.
- *Cross_ANY*: the input is set every time the value of the signal crosses a defined threshold in any direction.

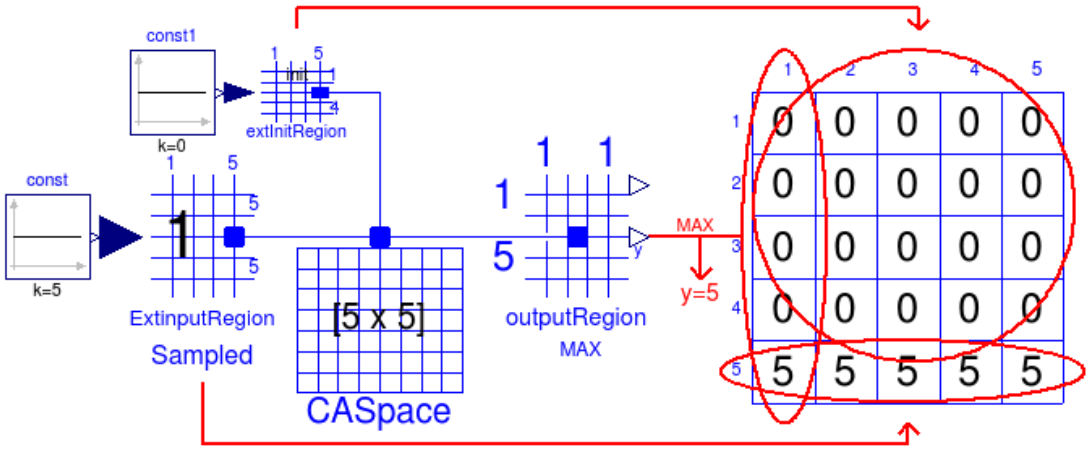


Figure 5: Example of behavior of external input region, external init region and output region models.

- *Sample*: the input is set periodically using the `sample` operator.

The signal is translated into an input using the function `ExtInput`, that can be redeclared using the `void ExtInput(int space,int row, int col, double value, int input_id)` function included in `draft.c`.

An example of external input is shown in Fig. 5. The *ExtInputRegion* model, connected to the CA *CASpace*, is used to set the state of the cells in the fifth row of the cellular space to the value of its input *const* (i.e., 5). The values of the states of the cells are graphically represented at the right of the figure.

4.3 External Init Region

This model can be used to set the initial state of a region of cells in the space using the value of an external signal. The model has an input port, named `u`, where a Real signal is received, and a port named `TO` that connects to the CA. This signal is translated, using the `ExtInit` function, into a cell state that will be used to initialize the cells in the region of the automata connected to port `TO`.

The 2D region is defined by the positions between `Rstart` and `Rend` (for the rows) and `Cstart` and `Cend` (for the columns). Only the column parameters are considered for 1D regions. The `ExtInit` function can be redeclared using the `void ExtInit(int space,int row, int col, double value)` function included in `draft.c`.

An example of external init is shown in Fig. 5. The *extInitRegion* model is used to initialize the state of the cells in the rows 1 to 4 (and columns 1 to 5) with the value of its input *const1* (e.g., 0).

4.4 Output Region

The `OutputRegion` model can be used to observe the state of the cells in a region of the automata connected to port `FROM`. The state is translated into an output Real signal that can be used by other Modelica models. As in the previous interface models, a 2D region is defined by the positions between `Rstart` and `Rend` (for the rows) and `Cstart` and `Cend` (for the columns). In 1D regions only the parameters referring to columns are considered.

The model contains two output Real ports, `y` and `yM[Rend-Rstart+1,Cend-Cstart+1]` (being `yM[Cend-Cstart+1]` for the 1D case). Depending on the value of the parameter `Output_type`, the state is observed in different ways:

1. (AVERAGE): the value of `y` is calculated as the average value of the states of the cells in the `[Rstart:Rend,Cstart:Cend]` interval, or `[Cstart:Cend]` for 1D.

2. (MAX): the value of `y` is calculated as the maximum value of the states of the cells in the `[Rstart:Rend,Cstart:Cend]` interval, or `[Cstart:Cend]` for 1D.

3. (MIN): the value of `y` is calculated as the minimum value of the states of the cells in the `[Rstart:Rend,Cstart:Cend]` interval, or `[Cstart:Cend]` for 1D.

DOI 10.3384/ECP14096489

4. (MATRIX): the value of the state of the i,j-th cell in the space is assigned to `yM`[m,n] (where $m = 1 : (\texttt{Rend} - \texttt{Rstart} + 1)$ and $n = 1 : (\texttt{Cend} - \texttt{Cstart} + 1)$). In 1D, the i-th cell is assigned to `yM`[n] (where $n = 1 : (\texttt{Cend} - \texttt{Cstart} + 1)$)

The value of the state is translated into a Real value using the `ExtOutput` function, that can be redeclared using the `double Output(int space, int row, int col)` function included in `draft.c`.

An example of output region is shown in Fig. 5. The *outputRegion* model is used to calculate the maximum value (i.e., `Output_type` = MAX) among the states of the cells in the first column of the space. In the case shown in the figure, the output port y of *outputRegion* is set to 5.

5 Case Studies

CellularAutomataLib includes several examples of 1D and 2D models, whose purpose is to demonstrate the functionality of the library and to facilitate the development of new models. The modeler can use these examples as a base for constructing new CA. Two of these examples are described in this section: a model of heat transfer on a chip and a SIR (Susceptible Infected Removed) epidemic spread model.

5.1 Heat Transfer on a Chip

The model describes the flow of the heat generated in a chip by the execution of software instructions. Two heat transfer mechanisms are considered: heat diffusion in the chip surface and convective heat flow from the chip surface to the air. The model contains two bi-dimensional CA: one describes the chip and the other describes the air. The software execution is modeled using power sources located at certain points of the chip surface. These points correspond to the position of the circuit components (ALU, memory, etc.) that dissipate more heat.

The structure of the CA model is shown in Fig. 6. This model combines two cellular spaces, one for the chip (named *Chip*) and another for the air (named *Air*), with other Modelica models used to represent the sources of power (named *T+3S+N*, *Pg1* and *Pg2*). Two external input region models (named *Pext1* and *Pext2*) are used to combine the external sources of power with the *Chip* cellular space. Two input regions are used to represent the transfer of heat between chip and air (named *Chip2Air*), and vice-versa (named *Air2Chip*). An external init region (named *InitTemp*) is

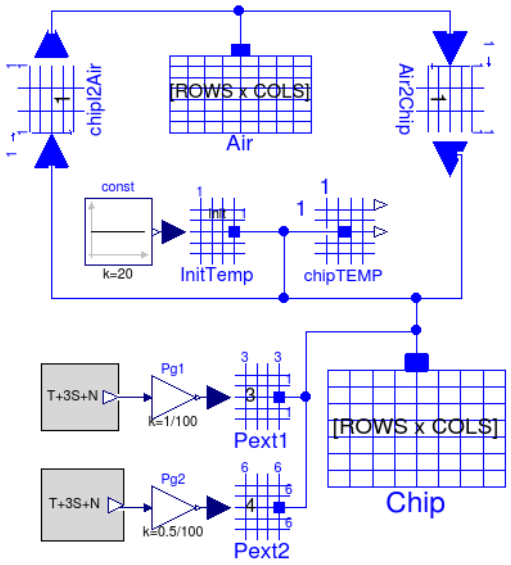


Figure 6: CA model of heat transfer on a chip.

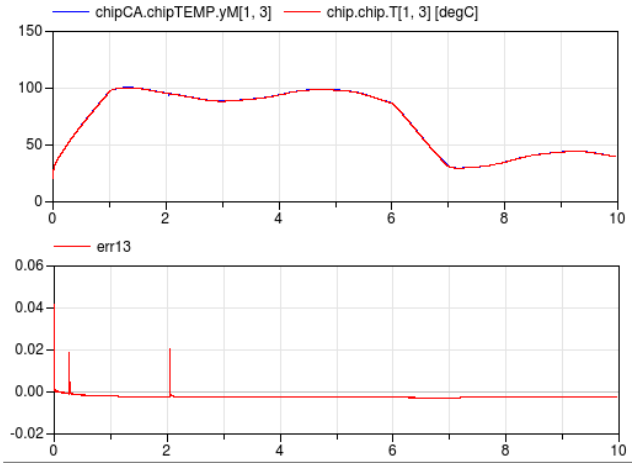
Table 2: Parameters of the chip model.

Name	*Value*	*Unit*	*Description*
gamma	100	$\frac{W}{m^2.K}$	Heat transfer coefficient
cp	710	$\frac{J}{kg.K}$	Specific heat capacity
ro	2330	$\frac{kg}{m^3}$	Average density
rows	10	-	Number of rows
cols	10	-	Number of columns
length	0.005	m	Layer length
width	0.005	m	Layer width
thickness	0.0001	m	Layer thickness
k	149	$\frac{W}{m.K}$	Thermal conductivity

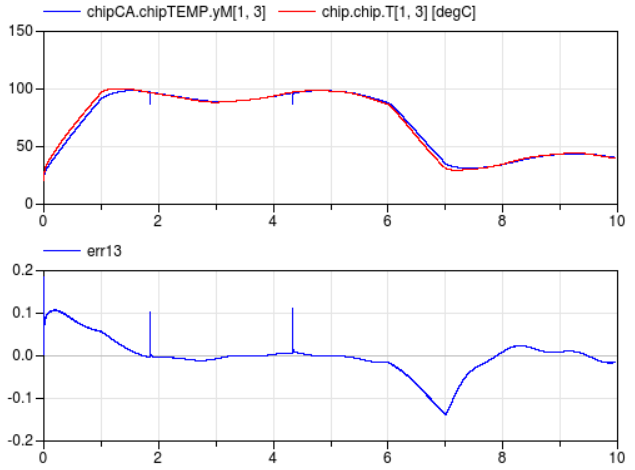
used to initialize the cells of *Chip* at 20ºC, given by a constant source from the Modelica Standard Library. Finally, an output region model (named *chipTEMP*) is used to observe the evolution of temperatures in the chip.

The model has been implemented into a C file, named `chip.c`. The equations that describe the heat transfer have been implemented in the transition function of Chip. The *Air2Chip* model sets the temperature of the Air as an input for *Chip*, which is used to calculate the convection of heat from the chip to the air. The *Pext1* and *Pext2* set two inputs for *Chip* that are used as input heat flows in the equations.

The transition function calculates the evolution of temperatures in the chip using two alternatives: a forward Euler and a leap-frog integration algorithms. These are explicit integration algorithms that are easily



(a) Euler integrator



(b) Leap-frog integrator

Figure 7: Simulation results for cell $[1,3]$ and error between Modelica and CA models: a) using Euler integration; and b) using Leap-frog integration.

included in the transition function of the CA. At each step of the simulation the transition function calculates an integration step and updates the values of the temperatures. The interval between steps using the forward Euler has to be $0.0001s$ in order to ensure stability. That interval can be increased to $0.001s$ using the leap-frog algorithm.

An analogous model has been developed using Modelica. In this model, the space has been discretized using finite volumes. In order to perform a comparison between the Modelica and the CA models, each volume will be represented by a cell in the CA.

Both models, Modelica and CA, have been simulated for $10s$ using the parameters shown in Table 2. The simulation results at the cell $[1,3]$ and the error between the Modelica and CA approaches

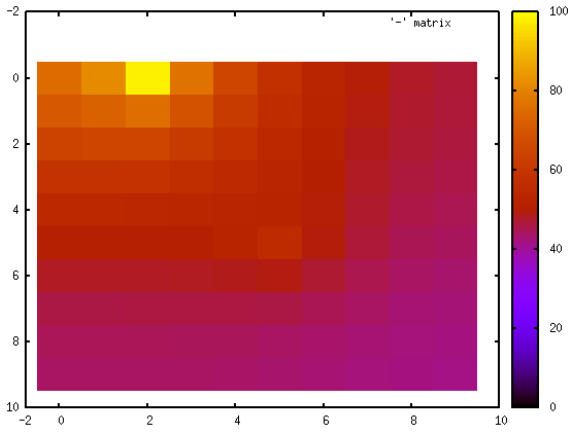


Figure 8: Capture of the graphical animation for the chip CA model.

Table 3: CPU time (in seconds) for integration of models during $10s$ of simulated time.

Model	*Grid Side Size*			
	10	*50*	*100*	*200*
Modelica (DASSL)	0.02	15.3	506	error
Modelica (EULER)	1.26	1225	5046	-
CA (EULER)	22	1670	6720	-
CA (LEAP-FROG)	2	173	711	2790

Table 4: Number of equations for the *Chip* model.

Model	*Grid Side Size*			
	10	*50*	*100*	*200*
Modelica	$1.2e^3$	$2.7e^4$	$1.1e^5$	$4.4e^5$
CA	$1.7e^3$	$2.5e^3$	$1e^4$	$4e^4$
CA without *chipTEMP*	70	70	70	70

are shown in Fig. 7. A capture of the graphical animation is shown in Fig. 8. The evolution of the simulation time with respect to the size of the grid is shown in Table 3. The simulation using EULER integration for the 200x200 grid was not performed and thus it does not appear in the table. The number of equations in the Modelica and CA models are shown in Table 4. If the *chipTEMP* output region model is removed from the CA model the number of equations is 70, independently of the grid size. The Modelica model rapidly reaches the maximum number of equations that can be efficiently handled by Modelica/Dymola, while the number of equations in the CA model remains lower. Note that Dymola

DOI 10.3384/ECP14096489

fails, due to an unknown internal error, to compile the Modelica model in a grid of 200x200 cells.

5.2 Epidemic Spread

The dynamics of epidemic spread are modeled in this example. This model was proposed in [21]. It is a SIR model where susceptible (S), infected (I) and recovered (R) individuals are considered. The evolution of the number of these individuals is defined by Eqs. (1), (2) and (3).

$$I_{ij}^{t} = (1-\varepsilon)\cdot I_{ij}^{t-1} + v\cdot S_{ij}^{t-1}\cdot I_{ij}^{t-1} + S_{ij}^{t-1}\cdot \sum_{(\alpha,\beta)\in V} \frac{N_{i+\alpha,j+\beta}}{N_{ij}}\cdot \mu_{\alpha\beta}^{i,j}\cdot I_{i+\alpha,j+\beta}^{t-1} \quad (1)$$

$$S_{ij}^{t} = S_{ij}^{t-1} - v\cdot S_{ij}^{t-1}\cdot I_{ij}^{t-1} - S_{ij}^{t-1}\cdot \sum_{(\alpha,\beta)\in V} \frac{N_{i+\alpha,j+\beta}}{N_{ij}}\cdot \mu_{\alpha\beta}^{i,j}\cdot I_{i+\alpha,j+\beta}^{t-1} \quad (2)$$

$$R_{ij}^{t} = R_{ij}^{t-1} + \varepsilon\cdot I_{ij}^{t-1} \quad (3)$$

where V is the neighborhood of the (i,j) cell, and $\mu_{\alpha\beta}^{i,j} = c_{\alpha\beta}^{(i,j)}\cdot m_{\alpha\beta}^{(i,j)}\cdot v$, where $c_{\alpha\beta}^{(i,j)}$ and $m_{\alpha\beta}^{(i,j)}$ are the connection factor and the movement factor between the (i,j) cell and its neighbor cell $(i+\alpha, j+\beta)$, and $v \in [0,1)$ is the virulence of the epidemic. The parameter ε defines the portion of infected individuals that recover from the disease at each step.

This model has been programmed in a C file, named `epidemics.c`. The size of the cellular space has been set to 50x50 in order to validate its results with the ones presented in [21]. Only the cell in the center of the space (i.e., position $[25,25]$) is initialized at the beginning. The parameters are set using the values: $\varepsilon = 0.4$, $v = 0.6$, $c = 1$ and $m = 0.5$. The Moore's neighborhood is used. The CA model includes the cellular space model and three output region models (see Fig. 9), used to sum the values of the state variables (S, I and R) of the whole CA. Each output region model redefines the `ExtOutput` function using a different function from `epidemics.c`, in order to observe the desired variable. The simulation results after 50 steps are shown in Fig. 10.

In order to demonstrate the simulation of a larger space, this model has been simulated with an space size of 500x500. This generates around 750,000 equations, due to the matrices defined in the output region models. These matrices are managed by the Modelica simulation algorithm. Dymola generates a C file of 140MB which is difficult to compile and simulate. Because of this the output region models

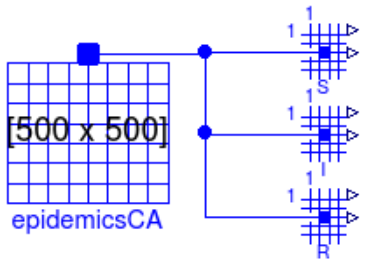


Figure 9: CA model of SIR epidemics spread.

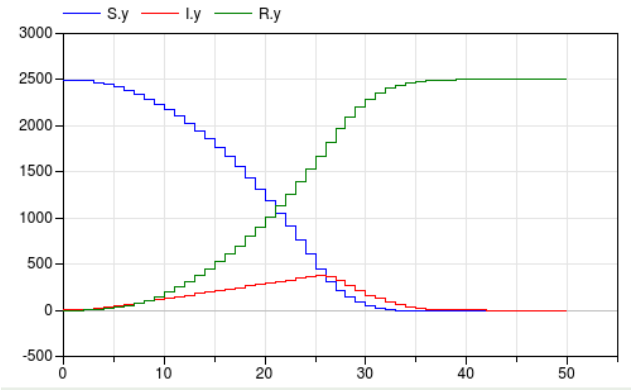


Figure 10: Evolution of the sum of the state variables (S,I and R) of the whole CA after 50 steps.

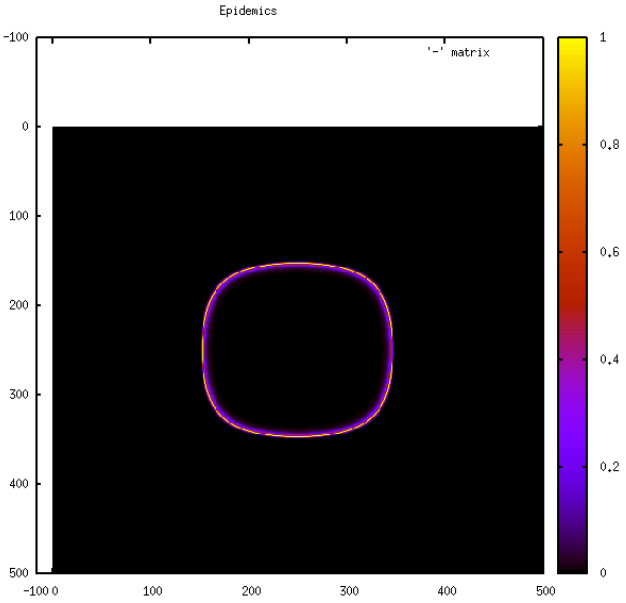


Figure 11: Capture of graphical animation at t = 100 for the epidemic spread model using a 500x500 grid.

are removed from the CA model before performing the simulation. The capture of the graphical animation at the final step (t = 100) is shown in Fig. 11.

6 Conclusions

A new Modelica library has been developed to facilitate the description of CA models. The simulation algorithms are transparent to the user, who has to focus on the description of the behavior of the model. The behavior of the models is described using external C functions. The use of external functions improves the performance and scalability of the simulations. The functionality of the library has been demonstrated by means of two case studies.

Some future work ideas are: to support the description of 3D models; to improve the generation of the graphical animation using graphical libraries instead of Gnuplot; to develop a graphical interface to define the initial conditions of the CA model; and to automatically parallelize the simulation of the CA in order to improve the performance.

7 Acknowledgments

This work has been supported by UNED, under 2013-026-UNED-PROY grant.

References

[1] von Neumann J. Theory of self-reproducing automata. Univ. of Illinois Press, Urbana and London, 1966.

[2] Ilachinski A. Cellular Automata: A Discrete Universe. World Scientific, Singapore, 2001.

[3] Schiff J.L. Cellular Automata: A Discrete View of the World. Wiley-Interscience, New York, USA, 2008.

[4] Wolfram S. A New Kind of Science. Wolfram Media Inc., Champain, IL, USA, 2002.

[5] Vangheluwe H.L.M., Vansteenkiste G.C. The cellular automata formalism and its relationship to DEVS. In: In 14th European Simulation Multi-conference (ESM.

[6] Hötzendorfer H., Estelberger W., Breitenecker F., Wassertheurer S. Three-dimensional cellular automaton simulation of tumour growth in inhomogeneus oxygen environment. Mathematical and Computer Modelling of Dynamical Systems, 15:pp. 177–189, 2009.

[7] O'Sullivan D., Torrens P.M. Cellular Models of Urban Systems. In: S. Bandini, T. Worsch, eds., Theoretical and Practical Issues on Cellular Automata. Springer-Verlag, London, 2000.

[8] Kier L.B., Seybold P.G., Cheng C.K. Modeling Chemical Systems using Cellular Automata. Springer, Dordrecht, The Netherlands, 2005.

[9] Rouhaud J.F. Cellular automata and consumer behaviour. European Journal of Economic and Social Systems, 14:pp. 37–52, 2000.

[10] Kroc J., Sloot P.M., Hoekstra A.G., eds. Simulating Complex Systems by Cellular Automata. Springer-Verlag, Berlin, 2010.

[11] Ganguly N., Sikdar B.K., Deutsch A., Canright G., Chaudhuri P.P. A Survey on Cellular Automata. Tech. rep., 2003.

[12] Fritzson P. Principles of Object-Oriented Modeling and Simulation with Modelica 2.1. Wiley-IEEE Computer Society Pr, 2003.

[13] Sanz V., Urquia A. An Approach to Cellular Automata Modeling in Modelica. In: Proceedings of the 5th International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools, pp. 121–130. Nottingham, UK, 2013.

[14] Sanz V., Urquia A., Cellier F.E., Dormido S. System Modeling Using the Parallel DEVS Formalism and the Modelica language. Simulation Modeling Practice and Theory, 18(7):pp. 998–1018, 2010.

[15] Sanz V., Urquia A., Dormido S. Parallel DEVS and Process-Oriented Modeling in Modelica. In: Proceedings of the 7^{th} International Modelica Conference, pp. 96–107. Como, Italy, 2009.

[16] Zeigler B.P., Kim T.G., Prähofer H. Theory of Modeling and Simulation. Academic Press, Inc., Orlando, FL, USA, 2000.

[17] Sanz V. Hybrid System Modeling Using the Parallel DEVS Formalism and the Modelica Language. Ph.D. thesis, ETSI Informática, UNED, Madrid, Spain, 2010.

[18] Zimmer D. Module-Preserving Compilation of Modelica Models. In: Proceedings of the 7^{th} International Modelica Conference, pp. 880–889. Como, Italy, 2009.

[19] Modelica Association. Modelica - An Unified Object-Oriented Language for Physical Systems Modeling. Language Specification version 3.3, 2013. URL http://www.modelica.org/documents.

[20] Williams T., Kelley C. Gnuplot 4.6: An Interactive Plotting Program, 2012. URL http://www.gnuplot.info.

[21] White S.H., del Rey A.M., Sanchez G.R. Modeling epidemics using cellular automata. Applied Mathematics and Computation, 186(2007):pp. 193–202, 2007.

 DOI 10.3384/ECP14096489