An Approach to Cellular Automata Modeling in Modelica

Victorino Sanz   Alfonso Urquia
Dpto. de Informática y Automática, ETSI Informática, UNED, Spain
{vsanz,aurquia}@dia.uned.es

Abstract

A new Modelica library, named CellularPDEVS, is introduced in this manuscript. This new library facilitates the description of one- and two-dimensional Cellular Automata (CA) models in Modelica. CellularPDEVS models have been specified using Parallel DEVS. The library has been implemented using the functionality of the DEVSLib library which supports the Parallel DEVS formalism in Modelica. CellularPDEVS allows the user to focus on describing the behavior of the cell and the characteristics of the cellular space. CellularPDEVS models are compatible with other DEVSLib models, facilitating the combination of CA, Parallel DEVS and other Modelica models. Three examples are presented: Wolfram's rule 30 and 110, and the Conway's Game of Life.

Keywords Modelica, Cellular Automata, Parallel DEVS, CellularPDEVS, DEVSLib

1. Introduction

Cellular automata (CA) are a class of models initially proposed in the 1940s by John von Neumann and Stanislaw Ulam [30, 29, 28]. CA are dynamic, discrete-time and discrete-space models. They are represented as a grid of identical discrete volumes, named cells [11]. The grid can be in any finite number of dimensions. The state of each single cell is finite and it is usually represented using integer numbers. The operational dynamics of the automata is described by a rule or transition function that is used to update the state of each cell at discrete time steps. 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 [27]. Examples of different neighborhoods are shown in Figure 1: 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 [34].

5th International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools. 19 April, 2013, University of Nottingham, UK.

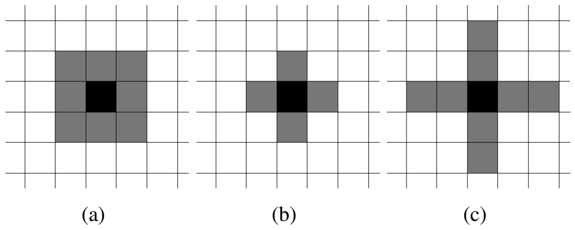


Figure 1. Examples of CA neighborhoods: a) Moore's; b) von Neumann's and; c) extended von Neumann's.

As it can be observed, the definition and behavior of the CA are simple. CA can provide an intuitive way of describing complex behavior using simple rules. CA may be considered as discrete idealizations in time and space of physical systems [35]. Due to its simplicity, CA have been used to describe models of complex systems in multiple domains. CA models have been developed in areas like chemistry [12], economics [22], medicine [10], biology and environment [13], and urban architecture [18], among many others [8]. An extension of CA models, named Lattice Gas Cellular Automata (LGCA), has been applied to the study of fluid flows. LGCA models have been also extended into Lattice Boltzmann Models (LBM) that are used as a microscopic approach for the study of fluid dynamics [33].

CA models can be combined with models described using different formalisms. For example, macroscopic quantities are usually calculated from LGCA and LBM models in order to combine them with other continuous-time models. One of the motivations of the work presented in this manuscript is to facilitate the combination of CA and continuous-time models using Modelica.

Discrete-event modeling specifications have been used to formally describe the behavior of CA, facilitating the development of models and their understanding. For instance, DEVS has been used by Zeigler [37] and Wainer [32] to describe CA models. The former provides a description of CA using Classic DEVS and Multicomponent DEVS. Models following these specifications can be implemented using tools such as DEVSJAVA [38], CoSMoS and MS4 Me™[36]. The latter introduces the Cell-DEVS formalism that is supported by CD++ [31].

On the other hand, the general-purpose, object-oriented modeling languages support the physical modeling paradigm [2]. In particular, the Modelica language [14] facilitates the object-oriented description of DAE-hybrid models, i.e., models composed of differential and algebraic equations, and discrete-time events. Modelica supports a declarative description of the continuous-time part of the model (i.e., equation-oriented modeling) and provides language expressions for describing discrete-time events. A detailed description of the characteristics of the language can be found in the specification of the language [14].

Modelica features have facilitated the development of libraries supporting several modeling formalisms and describing phenomena in different physical domains [15]. The main Modelica library is the Modelica Standard Library (MSL) [17] which is developed and supported by the Modelica Association. Modelica facilitates the reuse of models and model components which contribute to reduce the cost of new model development [21].

A number of Modelica libraries have been implemented for supporting discrete-event modeling formalisms, including StateCharts [6], state graphs [19], hybrid automata [20], Petri Nets [16] and extended Petri Nets [5]. The DEVSLib library was developed by the authors to facilitate the description of Parallel DEVS models in Modelica, and their combination with other Modelica models [24, 25].

Modelica has been used to describe CA models. The Game of Life is a particular example of two dimensional CA. The description in Modelica of the Conway's Game of Life is discussed by Fritzson [7]. In this implementation, the cellular space is represented using a matrix of integer numbers. The initial condition is set using a vector that contains the coordinates of the initially active cells. The behavior of the model is implemented using a function that is evaluated at discrete intervals using the Modelica *sample* operator. Two *for* loops are used in this function to iterate over all the components of the matrix, calculating the state of the neighbors (following Moore's neighborhood) and updating the state of the current component.

A new Modelica library, named CellularPDEVS, has been developed by the authors in order to facilitate the description of CA. CA in CellularPDEVS are described using the Parallel DEVS formalism, as coupled models of interconnected atomic cells. CellularPDEVS has been programmed using the DEVSLib Modelica library [25]. In this way, CA constructed using CellularPDEVS are compatible with the models described using DEVSLib, facilitating the connection between CA, Parallel DEVS and other Modelica models. The CellularPDEVS library can be freely downloaded as a part of the DESLib library [26, 4].

The structure of the manuscript is as follows. A short introduction to Parallel DEVS is presented in Section 2. The use of the Parallel DEVS as a base for CA implementation in Modelica is discussed in Section 3. The architecture and functionality of the CellularPDEVS library are described in Section 4. The construction of new CA using CellularPDEVS, as well as examples of one and two dimensional CA, are presented in Section 5. Finally, some future work ideas and conclusions are given in Sections 6 and 7, respectively.

2. Parallel DEVS

The Parallel DEVS formalism is briefly introduced in this section. Models in Parallel DEVS can be described behaviorally (named *atomic*) or structurally (named *coupled*).

2.1 Atomic Parallel DEVS Models

According to the Parallel DEVS formalism, an atomic model is the smallest component that can be used to describe the behavior of a system. It is defined by a tuple of eight elements [3, 37]:

$$Atomic =< X_M, S, Y_M, \delta_{int}, \delta_{ext}, \delta_{con}, \lambda, ta >$$

where:

$X_M = \{(p,v)|p \in IPorts, v \in X_p\}$ is the set of *input ports and values*.

S is the set of *sequential states*.

$Y_M = \{(p,v)|p \in OPorts, v \in Y_p\}$ is the set of *output ports and values*.

$\delta_{int} : S \longrightarrow S$ is the *internal transition* function.

$\delta_{ext} : Q \times X_M^b \longrightarrow S$ is the *external transition* function, where $Q = \{(s,e)|s \in S, 0 \leq e \leq ta(s)\}$ is the *total state* set and e is the *time elapsed* since the last transition.

$\delta_{con} : Q \times X_M^b \longrightarrow S$ is the *confluent transition* function.

$\lambda : S \longrightarrow Y_M^b$ is the *output* function.

$ta : S \longrightarrow \Re^+_{0,\infty}$ is the *time advance* function.

An atomic model remains in the state $s \in S$, for a time interval $t_s = ta(s)$. After t_s is elapsed, an *internal event* is triggered and the state is changed to $s_{new} = \delta_{int}(s)$. Before that, an output can be generated using the output function and the state prior to the event ($output = \lambda(s)$).

A new internal event is scheduled to occur at time instant $t_{new} = ta(s_{new}) + time$, where $time$ is the current time, i.e., the time instant of the current event, and $ta(s_{new})$ is the duration until the next internal event scheduled as a consequence of the current event. The duration $ta(s_{new})$ is a function of the new state s_{new}.

Multiple inputs can be received simultaneously through one or several ports:

- If any input is received at time t_{ext} and $t_{ext} < t_s$ (so the inputs are received before the next internal event), an *external event* is triggered. As a consequence of the external event, the state is changed to $s_{new2} = \delta_{ext}(s, e, bag)$, where s is the current state, e is the elapsed time since the last transition ($t_{ext} - t_{last}$) and $bag \subseteq X_M$ is the set of received input messages.
- If the external input is received at time t_{ext} and $t_{ext} = t_s$, the external and the internal events are triggered simultaneously. This situation triggers a *confluent event* (that substitutes the external and internal events), and the state is changed to $s_{new3} = \delta_{con}(s, e, bag)$, being s the current state, e the elapsed time, and $bag \subseteq X_M$ the set of received inputs (similarly to the δ_{ext} function).

Also, similarly to the internal events, an output can be generated as $output = \lambda(s)$ before executing the confluent transition function.

New internal events are also scheduled after the external and confluent transitions using $ta()$. Note that the time advance function can return a zero value, generating an immediate internal event.

2.2 Coupled Parallel DEVS Models

The Parallel DEVS formalism supports the hierarchical and modular description of the model. Every model has an interface to communicate with other models.

A coupled Parallel DEVS model is a model composed of several interconnected atomic or coupled models that communicate externally using the input and output ports of the coupled model interface. It is described by the following tuple [37]:

$$Coupled =< X, Y, D, \{M_d | d \in D\}, EIC, EOC, IC >$$

where:

$X = \{(p, v) | p \in IPorts, v \in X_p\}$ is the set of *input ports and values*.

$Y = \{(p, v) | p \in OPorts, v \in Y_p\}$ is the set of *output ports and values*.

D is the set of the *component names*.

M_d is a *DEVS model*, for each $d \in D$.

EIC is the *External Input Coupling*: connections between the inputs of the coupled model and its internal components.

EOC is the *External Output Coupling*: connections between the internal components and the outputs of the coupled model.

IC is the *Internal Coupling*: connections between the internal components.

The connection of Parallel DEVS models implies the establishment of an information transmission mechanism between the connected models. Parallel DEVS models follow a message passing communication mechanism. A model generates messages as outputs using its output function which are received by other models as external inputs. Messages can be received simultaneously through one or multiple ports. Connections between models can be in the form of 1-to-1, 1-to-many and many-to-1. Each message can transport an arbitrarily complex amount of information, depending on the particular application or experiment being studied.

3. Specification of CellularPDEVS Models

Parallel DEVS has been used to describe the CA components (i.e., the cell and the cellular space) implemented in CellularPDEVS. The formal specification of these components is presented in this section.

3.1 Specification of the Cell Models

CellularPDEVS includes two cell models: $Cell1D$ and $Cell2D$. The formal specification of the one dimensional cell, $Cell1D$, following Parallel DEVS is shown in Table 1.

The interface of $Cell1D$ is composed of:

- The "in_E" input port, used to connect with its eastern neighbor.
- The "in_W" input port, used to connect with its western neighbor.
- The "in_{ext}" input port, used to receive external inputs from outside the cellular space.
- The "out" output port, used to communicate the state of the cell.

A graphical representation of the interface of the $Cell1D$ model is shown in Figure 2.

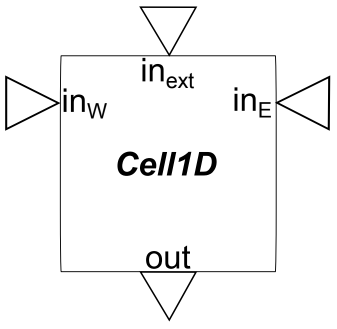


Figure 2. Interface of the $Cell1D$ model.

The state variables of $Cell1D$ are:

- $phase$ that represents if the cell is "$active$" or "$passive$". An "$active$" phase means that the state variable CS of the cell has changed in the current time step. The cell remains "$active$" until the change of the state is communicated to the neighbors. Otherwise, the phase is "$passive$".
- $sigma$ that represents the time delay until the execution of the next internal transition of the cell.
- CS that represents the current state of the cell represented by the $Cell1D$ model.
- N_E and N_W that are used to locally store the state of the neighbors, also represented by $Cell1D$ models.

The behavior of the $Cell1D$ model is as follows. An example of cell simulation is shown in Figure 3, where the evolution of the inputs, outputs and the state of the cell are shown. Initially, cells have $sigma = \infty$ and $phase =$ "$passive$" meaning that without an external input no internal transitions will be executed in the cell. Input events sent to the "in_{ext}" port are intended for initializing the cell, and have to be received at discrete time steps. The default duration of the time step is 1 second, however it can be adjusted as desired. Input events received at port "in_{ext}" update the state variable CS of the cell, set $phase =$ "$active$" and schedule an internal event at $time + 0.5$ (i.e., the middle of the current time step). In the example shown in Figure 3, the initial input event is received

Table 1. Parallel DEVS specification of the Cell1D model included in CellularPDEVS.

$$Cell1D =< X, S, Y, \delta_{int}, \delta_{ext}, \delta_{con}, \lambda, ta >$$

where:
$X_M = \{(ps, v) | p \in \{\text{“}in_E\text{”}, \text{“}in_W\text{”}, \text{“}in_{ext}\text{”}\}, v \in \mathbb{Z}\}$
$S = \{\text{“}active\text{”}, \text{“}passive\text{”}\} \times \mathbb{R}^+_{0,\infty} \times \mathbb{Z} \times \mathbb{Z} \times \mathbb{Z}$
$Y_M = \{(\text{‘}out\text{”}, \mathbb{Z})\}$
$\delta_{int}(phase, sigma, CS, N_E, N_W) =$
$$\begin{cases} (\text{“}passive\text{”}, 0.5, CS, N_E, N_W) & \text{if } phase == \text{“}active\text{”} \\ (\text{“}passive\text{”}, \infty, CS, N_E, N_W) & \text{if } phase == \text{“}passive\text{” and } CS == Rule(CS, N_E, N_W) \\ (\text{“}active\text{”}, 0.5, Rule(CS, N_E, N_W), N_E, N_W) & \text{if } phase == \text{“}passive\text{” and } CS \neq Rule(CS, N_E, N_W) \end{cases}$$
$\delta_{ext}(phase, sigma, CS, N_E, N_W, e, X^b_M) =$
$$\begin{cases} (phase, 0.5, CS, V_E, N_W) & \text{if event received in port “}in_E\text{” , whose value } V_E \in \mathbb{Z} \\ (phase, 0.5, CS, N_E, V_W) & \text{if event received in port “}in_W\text{” , whose value } V_W \in \mathbb{Z} \\ (phase, 0.5, CS, V_E, V_W) & \text{if events received in ports “}in_E\text{” and “}in_W\text{” , whose values } V_E, V_W \in \mathbb{Z} \\ (\text{“}active\text{”}, 0.5, V_{ext}, N_E, N_W) & \text{if event received in port “}in_{ext}\text{” , whose value } V_{ext} \in \mathbb{Z} \end{cases}$$
$\delta_{con}(S, e, X^b) = \delta_{int}(\delta_{ext}(S, e, X^b))$
$\lambda(phase, sigma, CS, N_E, N_W) =$
$$\begin{cases} (\text{“}out\text{”}, CS) & \text{if } phase == \text{“}active\text{”} \\ \varnothing & \text{if } phase == \text{“}passive\text{”} \end{cases}$$
$ta(phase, sigma, CS, N_E, N_W) = max(sigma, 0)$

at $time = 2s$. The scheduled internal event generates an output to send the new state to the neighbors (e.g., output at $time = 2.5s$ in the figure), fires an internal transition that sets $phase = \text{“}passive\text{”}$ and schedules a new internal event at $time + 0.5$ that corresponds to the next time step. The new scheduled internal event will fire a new internal transition to update the state CS of the cell using the $Rule$ function. If $CS \neq Rule(CS, N_E, N_W)$ (i.e., CS changes) then $phase = \text{“}active\text{”}$, $sigma = 0.5$ (i.e., the middle of the time step) and $CS = Rule(CS, N_E, N_W)$ (e.g., in the example, $phase$ always changes to $\text{“}active''$ when CS changes). Otherwise, $phase = \text{“}passive\text{”}$ and $sigma = \infty$ (e.g., in the example CS remains constant at $time = 3s$ and $sigma = \infty$). Each neighbor receives the update of the state in the middle of the time step as an external event. This fires an external transition that updates the locally stored neighbor state with the received value and sets $sigma = 0.5$ to schedule an internal event at the next time step (e.g., in the example, changes in the states of the neighbors are received as external inputs, and the local state variables N_E and N_W are updated with the received values).

The formal specification of the $Cell2D$ is analogous to the $Cell1D$, provided the following modifications. Additional input ports have to be included to connect with the additional neighbors (e.g., eight neighbors in the case of the Moore's neighborhood). The state has also to be extended to locally store the states of the additional neighbors. The δ_{ext} is used to update the locally stored states with the values received with the events. This function has

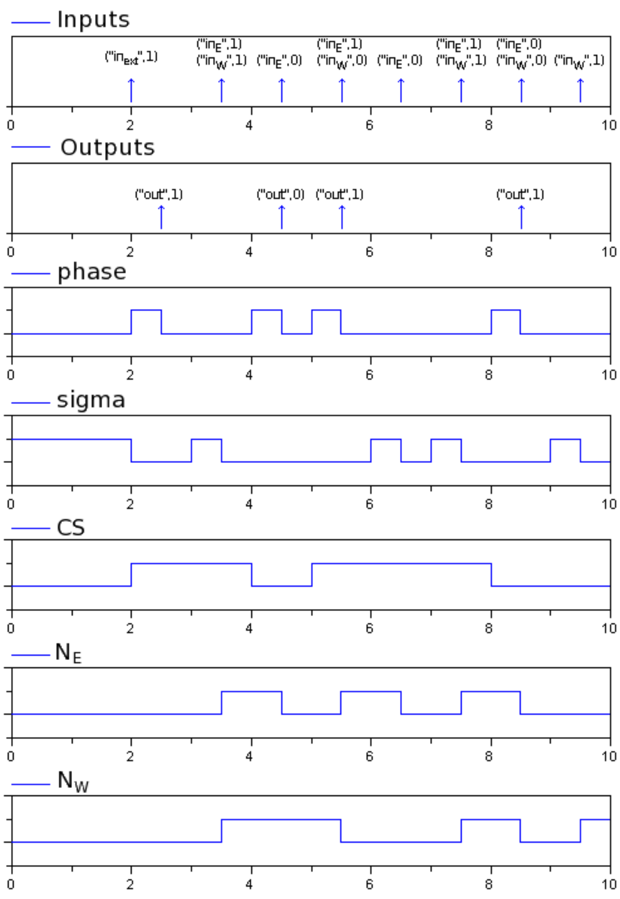


Figure 3. $Cell1D$ model execution example.

Table 2. Parallel DEVS specification of the $CellSpace1D$ model.

$$CellSpace1D = < X, Y, \{M_d | d \in D\}, EIC, EOC, IC >$$

where:
$X = \{(p, v) | p \in \{\text{“}in_1\text{”}, \ldots, \text{“}in_N\text{”}\}, v \in \mathbb{Z}\}$
$Y = \{(p, v) | p \in \{\text{“}out_1\text{”}, \ldots, \text{“}out_N\text{”}\}, v \in \mathbb{Z}\}$
$M_d = Cell1D$ for all $d \in D$, where $D = \{cell_1, \ldots, cell_N\}$
$EIC = \{(CellSpace1D, \text{“}in_i\text{”}) - (cell_i, \text{“}in_{ext}\text{”}) | i = 1, \ldots, N\}$
$EOC = \{(CellSpace1D, \text{“}out_i\text{”}) - (cell_i, \text{“}out\text{”}) | i = 1, \ldots, N\}$
$IC = \{(cell_{i-1}, \text{“}out\text{”}) - (cell_i, \text{“}in_W\text{”}) | i = 2, \ldots, N\} \cup$
$\{(cell_{i+1}, \text{“}out\text{”}) - (cell_i, \text{“}in_E\text{”}) | i = 1, \ldots, N-1\} \cup$
$\{(cell_N, \text{“}out\text{”}) - (cell_1, \text{“}in_W\text{”}), (cell_1, \text{“}out\text{”}) - (cell_N, \text{“}in_E\text{”})\}$

to be extended to allow all the possible combinations of input events from the ports of the model.

3.2 Specification of the Cellular Space Model

CellularPDEVS includes two models that represent cellular spaces: $CellSpace1D$ and $CellSpace2D$. Each cellular space is defined as a coupled Parallel DEVS model. Cellular spaces are composed of individual cells and their interconnections. The formal specification of the one dimensional cellular space, $CellSpace1D$, using Parallel DEVS is shown in Table 2.

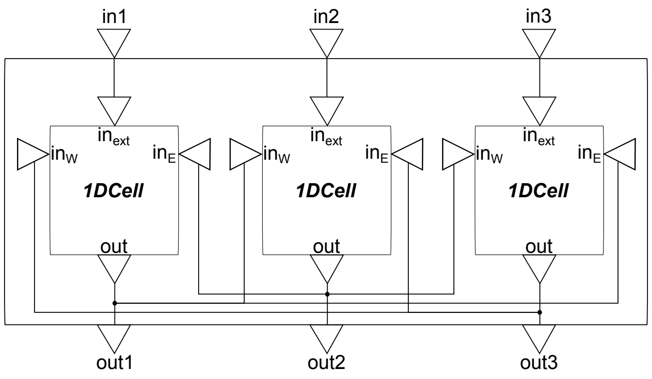


Figure 4. $CellSpace1D$ model of size 3.

The $CellSpace1D$ model is defined as an array of $Cell1D$ atomic models. The size of the array is N. An example of one dimensional CA with three cells is shown in Figure 4. Each cell in the array receives connections from its eastern neighbor (to the “in_E” port) and its western neighbor (to the “in_W” port) following the one dimensional Moore's neighborhood. The boundaries of the space are considered wrapped, so the western neighbor of the first cell of the array is the last cell of the array and vice-versa for the eastern neighbor of the last cell (cf. connections shown in Figure 4). The interface of the cellular space is composed of one input port (“in_i”) and one output port (“out_i”) for each cell in the space. These ports are connected to the “in_{ext}” and “out” ports of each cell, respectively.

The specification of the $CellSpace2D$ model is analogous to the $CellSpace1D$ provided the following modifications. The cellular space has to be defined as a two dimensional matrix of $Cell1D$ atomic models. The size of the space is $N \times N$. Each cell receives connections from its eight neighbors to its input ports. The boundaries are also wrapped considering the two dimensions of the space.

4. Architecture of CellularPDEVS

The architecture of the CellularPDEVS library is shown in Figure 5.

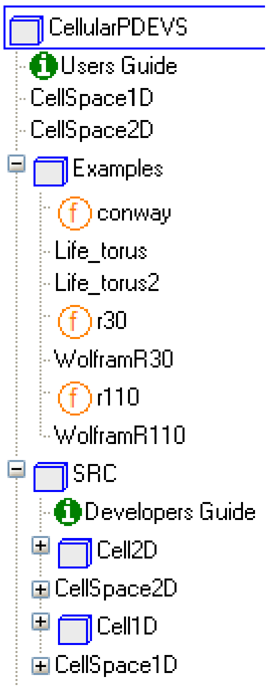

CellularPDEVS
Users Guide
CellSpace1D
CellSpace2D
Examples
conway
Life_torus
Life_torus2
r30
WolframR30
r110
WolframR110
SRC
Developers Guide
Cell2D
CellSpace2D
Cell1D
CellSpace1D

Figure 5. Architecture of the CellularPDEVS library.

The library is structured in two areas: 1) user's area and; 2) developer's area. The user's area is composed of:

- The *Users Guide* that contains the user oriented documentation.
- The *CellSpace1D* model that is used to construct new one dimensional CA.
- The *CellSpace2D* model that is used to construct new two dimensional CA.

- The *Examples* package that contains several examples of use.

The developer's area is encapsulated into the *SRC* package and contains the internal implementation of the models and the developer oriented documentation. CellularPDEVS includes two atomic DEVSLib models to represent one and two dimensional cells, named *Cell1D* and *Cell2D* respectively. Cellular spaces, named *SRC.CellSpace1D* and *SRC.CellSpace2D*, are constructed as an array or a matrix of interconnected *Cell1D* or *Cell2D* models depending on the dimension of the space.

The connections between cells are predefined into the cellular space describing the Moore's neighborhood. Since these connections between individual cells generate algebraic loops in the cellular space, a BreakLoop model from the DEVSLib library is inserted between cell connections. The BreakLoop model uses the Modelica *pre* operator to break the loop. The boundaries of the cellular space are wrapped automatically, by using the *mod* operator in the calculations of the indecees for the connections of cells.

Also, the cellular space includes the models required to generate the graphical animation of the simulation. The World, Fixed and fixedShape models, from the Multibody package of the Modelica Standard Library, have been used to generate the 1D and 2D visualizations.

Each cellular space model in CellularPDEVS includes a replaceable function, named *Rule* that needs to be redeclared in order to define the transition function for new models. The inputs of this function are the state of the current cell and its neighbors, and the output is the future state of the current cell. The prototype of this function is shown in Listing 1.

```
function Rule
  input Integer s;
  input Integer[N] neighbors;
  output Integer sout;
algorithm
end Rule;
```

Listing 1. Prototype of the CellularPDEVS transition function (`N` is the number of neighbors).

Finally, the cellular space also includes a Generator and a DUP_N models, from the DEVSLib library that are used to initialize the required cells at the beginning of the simulation. A message with $Type == 1$ is sent by the Generator model to each cell to be initialized. This message is received and managed by the external transition function of the cell which initializes the state and schedules an internal transition.

5. Modeling using CellularPDEVS

The construction of new CA using CellularPDEVS requires the description of the parameters of the cellular space (i.e., size and initial conditions of the cells) and the rule or transition function that describes the behavior of each cell. The formalism and the internal implementation of the cellular space and the cells is transparent to the user. As mentioned before, the graphical animation of the simulation is automatically generated, and it can be deactivated using a parameter of the model.

The transition function can be any Modelica function with the state of the cell and its neighbors as inputs, and the updated cell state as output (cf. the prototype of this function shown in Listing 1). All these values are represented using integer numbers.

CellularPDEVS models can be combined with any other Modelica model. The state of the cells in the automata can be observed using a variable, named *state*, included in the *CellSpace1D* and *CellSpace2D* models. Also, the state of the cells can be modified during the simulation by sending a message to the in_1 port of the desired cell. The message has to have $Type == 1$ and transport the new value for the cell. This message can be sent from any model constructed using DEVSLib. The *DUP_N* model can be used to duplicate the message if multiple cells have to be changed simultaneously.

Additionally, DEVSLib includes interfaces between continuous-time models and Parallel DEVS models which translate continuous-time signals into event trajectories (i.e., series of messages), and viceversa. These interface models allow combining the use of Parallel DEVS models developed with DEVSLib and hybrid models developed using other Modelica libraries. In this way, the behavior of a continuous Modelica model can be used to affect the state of the CA model, and viceversa.

The continuous-time to discrete-event interfaces translate continuous-time signals into event trajectories, where each event corresponds with the send of a message. Two different implementations of this interface are included in DEVSLib: quantization (*Quantizer* model) and value-crossing interfaces (*CrossUP* and *CrossDOWN* models). The quantization interface generates an event (i.e., a message) for every change in the continuous-time signal bigger than a given quantum value. The value-crossing interface generates an event every time the continuous signal crosses a given value in one direction, upwards or downwards.

The discrete-event to continuous-time interface translates the received message values into a piecewise-constant real signal. A boolean output is also included, together with the output real signal, in order to notify the reception instant of the messages. This boolean output may be useful when the received messages have the same value and consequently the reception instants cannot be inferred from the output real signal. This interface is implemented by the *DICO* model.

CellularPDEVS includes a package that contains several example model that are used to demonstrate its functionality and facilitate the construction of new models. These models have been also used to validate the library by comparison with equivalent models constructed using Golly which is an open source application for exploring CA models [9].

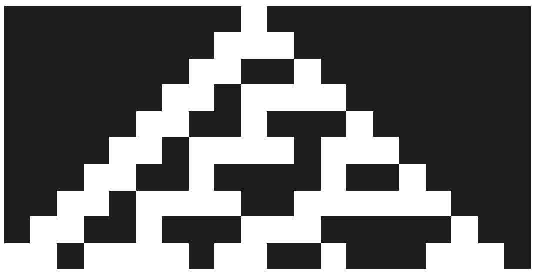
The examples included are the Wolfram's rule 30 and rule 110 [35] and two different initial conditions for the Conway's Game of Life. These models are detailed next.

5.1 Examples of One Dimensional CA

The Wolfram's rule 30 and rule 110 represent two different transition functions for one dimensional CA. These functions evaluate the state of a cell and its two adjacent neighbors and return the future state for the cell. The state of each cell is binary. The combination of possible input values and their outputs are shown in Table 3. The number of the rule, 30 and 110, defines the decimal value of the binary outputs of each function (e.g., looking at the *future state* row of the table, the binary values of the output can be interpreted as a decimal number: for the rule 30, $00011110_2 = 30_{10}$).

Table 3. Wolfram's rule 30 and rule 110.

	Rule 30							
current pattern	111	110	101	100	011	010	001	000
future state	0	0	0	1	1	1	1	0
	Rule 110							
current pattern	111	110	101	100	011	010	001	000
future state	0	1	1	0	1	1	1	0


(a)

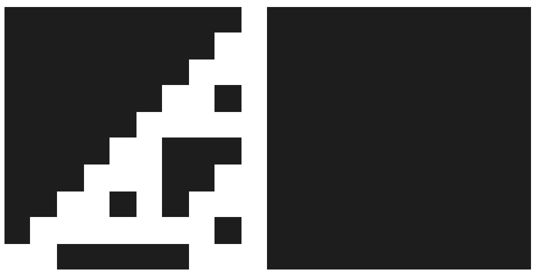

(b)

Figure 6. Simulation of CellularPDEVS 1D models: a) rule 30 and; b) rule 110.

The implementation of these rules in Modelica is straight forward. The code included in CellularPDEVS for the rule 30 is shown in Listing 2. An analogous code is included in the library for the rule 110.

The models of the rule 30 and rule 110 are constructed in CellularPDEVS by extending the *CellSpace1D* model, and redeclaring the *Rule* function with the corresponding functions. The parameters of the models are the size of the cellular space size (named *Ssize*) and the initial cell (named *init_cell*), since all Wolfram rules have only one active cell at the beginning of the simulation. The simulation results of the rule 30 and rule 110 models with a space size of 20, the cell in the middle of the space as initial cell (i.e., *init_cell* = 10) and a simulation time of 10 time steps are shown in Figure 6.

```
function r30
  input Integer s;
  input Integer[2] neighbors;
  output Integer sout;
protected
  Integer[2] n = neighbors;
algorithm
  if     n[2]==1 and s==1 and n[1]==1 then
    sout := 0;
  elseif n[2]==1 and s==1 and n[1]==0 then
    sout := 0;
  elseif n[2]==1 and s==0 and n[1]==1 then
    sout := 0;
  elseif n[2]==1 and s==0 and n[1]==0 then
    sout := 1;
  elseif n[2]==0 and s==1 and n[1]==1 then
    sout := 1;
  elseif n[2]==0 and s==1 and n[1]==0 then
    sout := 1;
  elseif n[2]==0 and s==0 and n[1]==1 then
    sout := 1;
  elseif n[2]==0 and s==0 and n[1]==0 then
    sout := 0;
  end if;
end r30;
```

Listing 2. Rule 30 Modelica code.

5.2 Examples of Two Dimensional CA

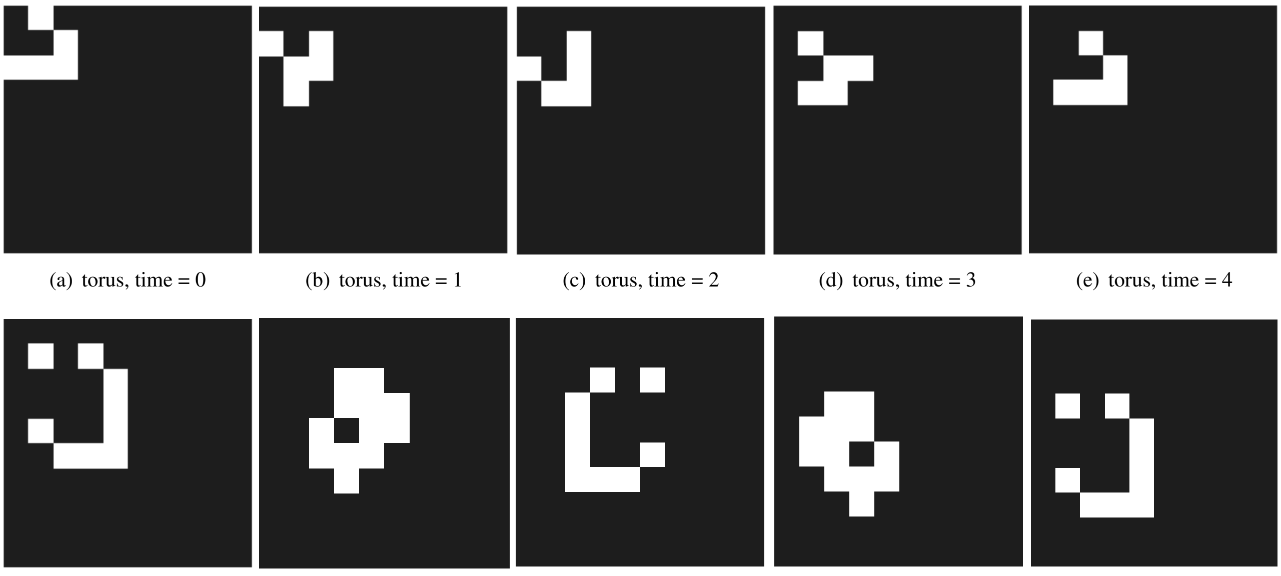
CellularPDEVS includes an implementation of the Game of Life model described by Conway. This model represents a two dimensional cell space where each cell may be alive or dead. The transition function of the model is defined by the following rules:

- A dead cell becomes alive when it has a number of alive neighbors equal to 3.
- A living cell dies when it has less than 2 or more than 3 alive neighbors.
- Otherwise, the cell remains in its current state.

```
function conway
  input Integer s;
  input Integer[8] neighbors;
  output Integer sout;
protected
  Integer[8] n = neighbors;
algorithm
  sout := s;
  if s==0 then // dead, maybe borns
    if sum(n)==3 then
      sout := 1;
    end if;
  else // alive, maybe dies
    if (sum(n)<2 or sum(n)>3) then
      sout := 0;
    end if;
  end if;
end conway;
```

Listing 3. Modelica code of the Game of Life's transition function.

The description of 2D models in CellularPDEVS is analogous to the 1D ones. The Game of Life model is con-


(a) torus, time = 0 (b) torus, time = 1 (c) torus, time = 2 (d) torus, time = 3 (e) torus, time = 4

(f) torus2, time = 0 (g) torus2, time = 1 (h) torus2, time = 2 (i) torus2, time = 3 (j) torus2, time = 4

Figure 7. Simulation of CellularPDEVS 2D models. The Conway's Game of Life.

structed by programming its transition function, as shown in Listing 3. The model extends the *CellSpace2D* model and redeclares the *Rule* function using the *conway* function shown. The initial state of the model is described as a matrix where each row represents the coordinates that correspond to the cells in the space that will be initially active (e.g., [1,2;2,3;4,4]). The first cell, (1,1), corresponds to the top-left cell of the matrix. The first index represents rows and the second represents columns (e.g., (3,5) represents the cell in the third row and the fifth column).

The first five steps of the simulation of two initial states, named *torus* and *torus2*, for the game of life model in CellularPDEVS are shown in Figure 7. The *torus* model corresponds to the initial cells: [1,2; 2,3; 3,1; 3,2; 3,3]. This model evolves in a periodical diagonal movement from the top-left area of the cellular space to the bottom-right. The *torus2* model corresponds to the initial cells: [2,2; 2,4; 3,5; 4,5; 5,5; 6,3; 6,4; 6,5; 5,2]. This model evolves in a periodical vertical movement from the top area of the cellular space to the bottom area.

6. Future Work

The models presented in this manuscript have been included in CellularPDEVS in order to validate the library and demonstrate its functionality. CellularPDEVS will be used to model more complex systems using CA such as a cement clinker cooler [1] or a PEM fuel cell [23]. The development of these models will show the applicability of library. Also, these new CA models will be used to evaluate the simulation performance of the library in comparison with the already developed Modelica models.

7. Conclusions

A new Modelica library has been developed to facilitate the description of Cellular Automata. These are discrete-time and -space models represented using a grid of individual cells, whose state is updated at discrete time steps using a predefined transition function. The library supports the description of one and two dimensional automata. The main components of the library are the cell and the cellular space. The behavior of these components has been specified using the Parallel DEVS formalism.

The behavior of each cell is specified as an atomic Parallel DEVS model and implemented using the DEVSLib library. The interface of the cell allows it to receive messages from its neighbors and from outside the cellular space. The state of the cell represented by the model is described using an integer number. Different behaviors can be described by redeclaring the rule, or transition function that defines the dynamics of the cell. The duration of the time step can be adjusted to the requirements of the simulation.

The cellular space is specified as a coupled Parallel DEVS model. It is composed of a grid of interconnected cell models. The interface of the cellular space allows to receive external messages, via its input ports, and to observe the state of the automata, via its output ports. This interface facilitates the combination of cellular automata models with other Modelica models. The boundaries of the cellular space are wrapped. It uses the Moore's neighborhood by default. However, future versions will allow the user to define the neighborhood as desired. A graphical animation of the simulation is automatically generated.

Three examples have been presented in order to demonstrate the functionality of the library and validate its models. The Wolfram's rule 30 and rule 110 elementary cellular automata as examples of one dimensional automata. The Conway's Game of Life, including two initial conditions, as an example of two dimensional automata.

References

[1] Oscar Acuña, Carla Martin-Villalba, and Alfonso Urquia. Virtual-lab of a cement clinker cooler for operator training.

In *Proceedings of the 7^{th} Vienna International Conference on Mathematical Modeling (MATHMOD)*, pages 331–336, Vienna, Austria, 2012.

[2] Karl J. Åström, Hilding Elmqvist, and Sven Erik Mattsson. Evolution of continuous-time modeling and simulation. In *Proceedings of the 12^{th} European Simulation Multiconference (ESM'98)*, pages 9–18, Manchester, UK, 1998.

[3] Alex Chung Hen Chow. Parallel DEVS: A parallel, hierarchical, modular modeling formalism and its distributed simulator. *Transactions of the Society for Computer Simulation International*, 13(2):55–67, 1996.

[4] Some free modeling and simulation resources. Dpto. Informática y Automática, UNED. `http://www.euclides.dia.uned.es/`, 2013.

[5] Stefan M.O. Fabricius and E. Badreddin. Hybrid dynamic plant performance analysis supported by extensions to the Petri Net library in Modelica. In *Proceedings of the 4^{th} Asian control Conference (ASCC)*, pages 41–50, Singapore, 2002.

[6] J.A. Ferreira and J.P. Estima de Oliveira. Modelling hybrid systems using StateCharts and Modelica. In *Proceedings of the 7^{th} IEEE International Conference on Emerging Technologies and Factory Automation*, pages 1063–1069, 1999.

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

[8] Niloy Ganguly, Biplab K Sikdar, Andreas Deutsch, Geoffrey Canright, and P Pal Chaudhuri. A survey on cellular automata. Technical report, 2003.

[9] Golly. `http://golly.sourceforge.net`, 2013.

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

[11] Andrew Ilachinski. *Cellular Automata: A Discrete Universe*. World Scientific, Singapore, 2001.

[12] Lemont B. Kier, Paul G. Seybold, and Chao-Kun Cheng. *Modeling Chemical Systems using Cellular Automata*. Springer, Dordrecht, The Netherlands, 2005.

[13] Jiri Kroc, Peter M.A. Sloot, and Alfons G. Hoekstra, editors. *Simulating Complex Systems by Cellular Automata*. Springer-Verlag, Berlin, 2010.

[14] Modelica Association. Modelica - An unified object-oriented language for physical systems modeling. Language specification version 3.1, 2012.

[15] Modelica Libraries. Modelica free and comercial libraries. `http://www.modelica.org/libraries`, 2013.

[16] Pieter J. Mosterman, Martin Otter, and Hilding Elmqvist. Modelling Petri Nets as local constraint equations for hybrid systems using Modelica. In *Proceedings of the Summer Computer Simulation Conference*, pages 314–319, 1998.

[17] Modelica standard library. `http://www.modelica.org/libraries/Modelica`, February 2010.

[18] David O'Sullivan and Paul M. Torrens. Cellular models of urban systems. In S. Bandini and T. Worsch, editors, *Theoretical and Practical Issues on Cellular Automata*. Springer-Verlag, London, 2000.

[19] Martin Otter, Karl-Erik Årzén, and Isolde Dressler. State-Graph - a Modelica library for hierarchical state machines. In *Proceedings of the 4^{th} International Modelica Conference*, pages 569–578, Hamburg, Germany, 2005.

[20] Tiziano Pulecchi and Francesco Casella. HyAuLib: modelling hybrid automata in Modelica. In *Proceedings of the 6^{th} International Modelica Conference*, pages 239–246, Bielefeld, Germany, 2008.

[21] Stewart Robinson, Richard E. Nance, Ray J. Paul, Michael Pidd, and Simon J.E. Taylor. Simulation model reuse: definitions, benefits and obstacles. *Simulation Modelling Practice and Theory*, 12(7-8):479 – 494, 2004. Simulation in Operational Research.

[22] Jean-François Rouhaud. Cellular automata and consumer behaviour. *European Journal of Economic and Social Systems*, 14:37–52, 2000.

[23] Miguel A. Rubio, Alfonso Urquia, and Sebastian Dormido. Dynamic modelling of PEM fuel cells using the *fuelcell*lib Modelica library. *Mathematical and Computer Modelling of Dynamical Systems*, 16(3):165–194, 2010.

[24] Victorino Sanz. *Hybrid System Modeling Using the Parallel DEVS Formalism and the Modelica Language*. PhD thesis, E.T.S.I. Informática, UNED, Madrid, Spain, 2010.

[25] Victorino Sanz, Alfonso Urquia, François E. Cellier, and Sebastian Dormido. System modeling using the Parallel DEVS formalism and the Modelica language. *Simulation Modeling Practice and Theory*, 18(7):998–1018, 2010.

[26] Victorino Sanz, Alfonso Urquia, and Sebastian Dormido. Parallel DEVS and process-oriented modeling in Modelica. In *Proceedings of the 7^{th} International Modelica Conference*, pages 96–107, Como, Italy, 2009.

[27] Joel L. Schiff. *Cellular Automata: A Discrete View of the World*. Wiley-Interscience, New York, NY, USA, 2008.

[28] Stanislaw Ulam. Random processes and transformations. In *Sets, Numbers and Universes*. MIT Press, Cambridge, 1974.

[29] John von Neumann. The general and logical theory of automata. In *Collectd Works*, volume 5. Macmillan, New York, 1963.

[30] John von Neumann. *Theory of self-reproducing automata*. University of Illinois Press, Urbana and London, 1966.

[31] Gabriel Wainer. CD++: A toolkit to develop DEVS models. *Software: Practice and Experience*, 32(13):1261–1306, 2002.

[32] Gabriel A. Wainer. *Discrete-Event Modeling and Simulation - A Practitioner's Approach*. CRC Press, Boca Raton, FL, USA, 2009.

[33] Dieter A. Wolf-Gladrow, editor. *Lattice Gas Cellular Automata and Lattice Boltzmann Models: An Introducction*. Springer-Verlag, Berlin, 2000.

[34] Stephen Wolfram. *Cellular Automata and Complexity: Collected Papers*. Addison-Wesley, 1994.

[35] Stephen Wolfram. *A New Kind of Science*. Wolfram Media Inc., Champain, IL, USA, 2002.

[36] Benard P. Zeigler and Hessam S. Sarjoughian, editors.

Guide to Modeling and Simulation of Systems of Systems. Springer-Verlag, London, 2013.

[37] Bernard P. Zeigler, Tag Gon Kim, and Herbert Prähofer. *Theory of Modeling and Simulation*. Academic Press, Inc., Orlando, FL, USA, 2000.

[38] Bernard P. Zeigler and Hessam S. Sarjoughian. Introduction to DEVS modeling & simulation with JAVA: Developing component-based simulation models, 2003.