To appear in *Mathematical and Computer Modelling of Dynamical Systems*
Vol. 00, No. 00, Month 20XX, 1–23

CellularAutomataLib2: Improving the Support for Cellular Automata Modeling in Modelica

Victorino Sanz[a*], Alfonso Urquia[a] and Alberto Leva[b]

[a]*Dpto. Informática y Automática, Universidad Nacional de Educación a Distancia (UNED), Juan del Rosal 16, 28040, Madrid, Spain;* [b]*Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano, Piazza Leonardo da Vinci 32, 20133, Milano, Italy*

(*v5.0 released February 2015*)

CellularAutomataLib is a library developed by the authors to facilitate the description of Cellular Automata (CA) models in Modelica. It supports the description of 1D and 2D CA and their combination with other Modelica models. Modeling versatility and scalability are the main focus in the design. The internal behavior of CA models is programmed in C, that is consequently hidden to the modeling tool and not considered in the causalization and manipulation of the model. A new version, named *CellularAutomataLib2*, is presented in this manuscript. The library has been extended to improve the simulation performance, by only evaluating active cells, and to support Lattice Gas Cellular Automata (LGCA) models. The library design and use are discussed. Two models, forest fire spread and the ARGESIM C17 "SIR-type Epidemic Spread", are used to illustrate the functionality of the library. *CellularAutomataLib2* is freely available at http://www.euclides.dia.uned.es (under the Modelica License 2).

Keywords: Cellular Automata; Modelica; Hybrid systems

AMS Subject Classification: 68Q80; 34K34; 68U20; 93A30

1. Introduction

Modelica [1] is a general-purpose, object-oriented modeling language designed to facilitate the application of the physical modeling paradigm, allowing significant reduction in the time and effort required for model development and maintenance, promoting model reuse, and reducing consequently the modeling costs [2]. Modelica models can be described using a combination of two approaches: behaviorally, by declaratively describing their mathematical behavior using equations, and; structurally, as a combination of multiple interconnected components. This functionality facilitates the development of modular and hierarchical models to describe complex systems and subsystems. Models are usually arranged into libraries of components, each library focused on a specific domain (e.g., electrical, multibody, thermodynamical, etc.) or formalism (e.g., Statecharts, PetriNets, System Dynamics, DEVS, Bond Graphs, etc.), thus supporting the development of multidomain and multiformalism hybrid models.

*Corresponding author. Email: vsanz@dia.uned.es

Models described in Modelica are composed of differential and algebraic equations, algorithms and events. This type of models, named hybrid-DAE models, describe dynamic systems with *time* as the only independent variable. However, the study of diverse phenomena, such as flexible bodies, sound, heat, fluid flows, etc., usually involves dependencies on spatial dimensions that are often modeled in terms of partial differential equations (PDE). The dependence on the spatial coordinates has to be considered when modeling these phenomena. As Modelica does not support the description of PDEs, Modelica use in describing models with dependence on the spatial coordinates is an open research topic. The benefit of using Modelica to describe spatially dependent models is the use of the object-oriented modeling functionality, already supported by the language, to describe different parts of the model in combination with the spatially dependent parts. Different approaches to describe such models in Modelica, such as PDE discretization, Modelica extensions, co-simulation and Cellular Automata, are discussed next.

PDEs can be discretized with respect to the spatial coordinates in order to represent them as equations in a hybrid DAE model. Modelica provides several features that can be used to describe the sets of equations with regular structure obtained from this spatial discretization: 1) describe the discretized equations by means of matrix equations and *for* clauses combined with the use of vector and matrix data types and; 2) the modular and hierarchical modeling using arrays of components and multiple connect sentences in a loop. This functionality has been used to apply different discretization methods, such as the Method of Lines (MOL) [3, 4], Finite Elements (FE) [5, 6] or Finite Volumes (FV) [7, 8]. However, the description of PDE models using these features presents the limitations discussed below.

Modelica models require analyses and symbolic manipulations of the equations (e.g., removal of alias variables and trivial equations, index reduction, causalization, automatic formula manipulation, algebraic-loop detection and tearing) in order to be simulated efficiently. Modelica tools, such as Dymola [9] and OpenModelica [10], translate the Modelica code into a flat hybrid-DAE model, perform these analyses and manipulations automatically, generate simulation code written in a programming language (e.g., Dymola and OpenModelica generate C code) and compile it into an executable file. Detailed descriptions of these manipulations can be found in [3] and [11].

Modelica tools are designed to efficiently handle models involving a number of equations in the order of up to hundreds of thousands. Using a space discretization approach to describe realistic PDE models (i.e., where the space domain is discretized into a large number of divisions) usually leads to systems with a higher number of equations, typically in the order of millions and tens of millions of equations. The execution by the Modelica tools of the algorithms to translate the Modelica code into executable code is time consuming when applied to models with so large number of equations. The procedure for translating the Modelica code into executable code becomes slow and the generated code is very large. These issues are discussed in detail in Section 2.

On the other hand, some extensions to the Modelica language have been proposed for describing PDE models [12]. However, these extensions are not currently supported by any Modelica simulation tool.

Another approach that can be found in the literature is the use of an specialized tool to describe the PDE model, and combine it with Modelica. The combination can be performed by means of: simulating the PDE model with the external tool and using the results as parameters or inputs for the Modelica model or; co-simulation, where multiple tools are simultaneously synchronized to execute the simulation. In these approaches, the modeler needs to understand, use and combine different tools and methodologies to develop the model, which increases the complexity of the development. Also, in co-simulation the overall simulation becomes more complex because the time advance and data exchange between different algorithms (i.e., tools) have to be synchronized. The Functional Mock-up Interface (FMI) constitutes an effort to facilitate the model exchange and co-simulation between different tools [13].

An alternative approach is the use of discrete-event formalisms to describe the spatially dependent model. In particular, Cellular Automata (CA) are simple discrete models represented by a set of cells arranged into a lattice structure, that can be in any number of finite dimensions [14]. Each cell represents a region of the space with a particular state, that dynamically changes during the simulation.

Formally, CA can be defined as a tuple [15]:

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

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, $\omega : T \to X$); S is the state that is the same for all cells because the cellular space is homogeneous; $\delta : \Omega \times S \to S$ is the global transition function used to update the state of each cell ($\delta(\omega, s_i) \to \delta_l(N_i)$, 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 $\lambda : S \to Y$ is the output function used to observe the state of the automata.

An extension of CA models, named 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 [16]. 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.

The objective of the work performed by the authors is to facilitate the description of CA in Modelica, in order to use them as an alternative approach to model in Modelica dynamic systems with spatial-dependent behavior. The development of CA models using Modelica and the encountered difficulties will be detailed in the next section. Briefly, the authors developed the *CellularPDEVS* library to describe CA models in Modelica using the Parallel DEVS (Discrete EVent System specification) formalism [17] to model the automata. However, the performance of the simulation was not satisfactory as it will be discussed. As an alternative approach for CA modeling and simulation with Modelica, the authors developed the first version of *CellularAutomataLib* [18].

In this manuscript a new version of the library, named *CellularAutomataLib2*, is presented. The features included in this new version are: a) the use of Modelica external objects to describe CA models in order to avoid possible platform-dependent problems; b) improvement of the simulation performance; and c) support of hexagonal lattice structures and the description of Lattice Gas Cellular Automata (LGCA) models.

The manuscript is structured as follows. A discussion about the previous work on the development of CA models using Modelica is given in Section 2. The new library, *CellularAutomataLib2*, is presented in Section 3, including its design, use and combination with other Modelica models. Two case studies, CA and LGCA models, are included to demonstrate the functionality of the library. A fire spread cellular automaton model is presented in Section 4. The ARGESIM C17 benchmark comparison "Temporal and Spatial Evolution of a SIR-type Epidemic" [19] is presented in Section 5. Finally, some conclusions and future work ideas are given in Section 6.

2. Related Work: Cellular Automata in Modelica

Different alternative implementations have been developed to describe CA using Modelica. In this section three CA implementations are presented and discussed: first, a custom implementation directly programmed in Modelica; second, a CA library described using the DEVS formalism, and; third, an implementation using external functions combined with Modelica.

2.1. *Custom Implementation*

A custom implementation is described in [11] that demonstrates the feasibility of describing CA models using Modelica. The Game of Life is implemented using a two-dimensional matrix of integer numbers to describe the cellular space, the initial conditions are set using a vector that contains the coordinates of initially active cells and the simulation algorithm is implemented using periodical time events to update the state of the cells iterating the whole matrix using two nested `for` loops. In this model, the description of the automaton and its simulation are coupled (i.e., the description of the cellular space and the evaluations of the transition function in each cell are combined in the same code), difficulting its reutilization to describe other automata.

The custom implementation of CA models in Modelica leads to an additional problem. The automatic translation performed to Modelica models with so large number of equations (when even possible) is time consuming and huge executable files are generated. As an example, the CA model presented in [11], initialized with a glider structure (i.e., 5 active cells), is translated using Dymola 2015 into a C code of 64.9KB for $N = 10$, 6.9MB for $N = 100$, 28.2MB for $N = 200$, 65.4MB for $N = 300$ and 117.3MB for $N = 400$, where N^2 is the size of the two-dimensional space. The redundancy in the structure of CA models is not efficiently managed and leads to decreasing performance during the translation procedure when the size of the model is increased. Since the translation of the model is automatically performed by the Modelica tool, the modeler cannot modify it, and thus, has to use the features provided by Modelica to include any scalability improvement directly in the model (i.e., the modeler can not modify the translation procedure to improve the scalability of the simulation).

2.2. *DEVS-based Implementation*

An approach to describe CA models in Modelica was performed by the authors [20]. The *CellularPDEVS* package, distributed with the DESLib library [21, 22], supports the description of CA using the Parallel DEVS formalism [17, 23]. The cellular space is represented by a coupled Parallel DEVS model of interconnected cells, 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. Cell states and transition rules are represented using Modelica data structures and algorithms, while the CA simulation algorithm remains transparent to the modeler. *CellularPDEVS* also facilitates the combination of CA models with other Modelica models.

However, the performance and the scalability of *CellularPDEVS* are not satisfactory. Firstly, due to the large size of CA models (i.e. the quadratic or cubic growth of the number of cells in 2D and 3D) and, as mentioned above, that Modelica tools have not been conceived to efficiently manage models with millions of equations. Secondly, due to the occurrence of long chains of events during their simulation, because of the interactions between neighboring cells.

The performance of the simulation is affected by the way the events are managed. Event conditions are checked during the simulation and if any event is triggered, the solution of the DAE system is halted and the event is managed. After each event, consistent restart values for the variables of the hybrid DAE model have to be found before resuming the numerical integration. The management of the event may change some variables that change the value of some event condition, which immediately triggers a new event. This procedure is called event iteration, and stops when no additional events are triggered. During the event iteration, after handling each event, the entire model is re-evaluated. A similar approach is used in the OpenModelica compiler [24].

Following the event iteration procedure described above, unnecessary evaluations of some model equations may be performed, which may significantly degrade simulation performance. In the case of CA described using DEVS, a change in the state of a cell generates one additional event for each of its neighbors, that receive input messages with the updated state. The whole model is re-evaluated after each event, even when the events only affect the neighboring cells. Also, in the case of CA combined with other Modelica continuous-time models, the management of an event in the automaton may not involve any change in the continuous-time part of the model, and so that part should not be involved in the event iteration. The authors have proposed an improvement for the event iteration procedure [25], however this proposal has still not been implemented in any tool.

2.3. *External Implementation*

In order to provide a workaround for the problems described above, the *CellularAutomataLib* library was designed and developed by the authors using Dymola [18]. The *CellularAutomataLib* library was designed along the following principles:

(1) *Scalability and performance*: The main objective of the library was to improve the scalability and performance problems described above, when describing CA using Modelica. The CA simulation algorithm (i.e., the computation of the states of the cells at each time step) was implemented using external C functions. Also, the description of the automaton (i.e., the cell state and the transition function) was performed using external C functions. The C code is compiled with the rest of the C code generated by the Modelica tool, as a result of the automatic translation procedure described above. In this way the description and simulation of the automaton is not involved in the translation

procedure. This approach has three advantages: (1) reduces the time spent by the Modelica tool in the automatic translation procedure; (2) the generated C code is smaller; and (3) the simulation is more efficient.

(2) *Transparent to the user*: the CA simulation algorithm included in the library and the description of the automaton (i.e., state and transition rule) are separated in order to allow the user to focus on description of the behavior of the model and not the simulation algorithm. Different CA can be described without modifying the simulation algorithm.

(3) *Flexibility*: the library facilitates the description of multiple CA behaviors by allowing to describe the state of the cell and the transition function using arbitrarily complex data structures and algorithms.

(4) *Graphical visualization of the simulation*: the simulation of the CA model is displayed using a graphical animation that is automatically generated using Gnuplot [26].

(5) *Interface with other Modelica models*: interface models that facilitate the connection of CA with other Modelica models were included in the library. These interface models can be used to combine CA either with other CA or with other Modelica models. The 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.

(6) *Open-source*: the library was freely distributed and can be downloaded from `http://www.euclides.dia.uned.es`.

3. *CellularAutomataLib2*

Following the design principles of *CellularAutomataLib*, a new version of the library has been developed. In this section, the design of the new library, its use to describe new CA models and the combination of CA models with other Modelica models are presented. The library has been developed using Dymola FD1 2015, configured to use the Clang 3.4.2 compiler, on an Intel(R) Core(TM) i7-4720HQ 2.6GHz machine with 16GB of RAM and running Linux 3.16 x86_64.

3.1. *Library Design*

While the general architecture, design and use of the library remains the same, the following improvements have been introduced in the new version:

- *CellularAutomataLib2* makes use of Modelica *external objects* (cf. page 156 of [1]) in order to avoid possible problems depending on the platform used and the implementation of the Modelica tool.
- The simulation performance has been improved by considering only active cells (i.e., cells whose state is subject to change during the current step) in each simulation step. This improvement reduces the amount of memory used during the simulation and the execution time, specially for models with a reduced number of active cells spanning over a large spatial domain (e.g., a glider structure in the Game of Life model). Thus, the performance of the simulation does not depend on the size of the cellular space but only on the number of active cells. A comparison between simulation performances is shown in Table 1. Note that the simulation time shown does not include the time spent

in the compilation of the generated C code. The *Custom* model corresponds to the Game of Life model described in [11], initialized with a glider structure. This model could not be compiled and simulated with $N = 400$ due to memory allocation problems (the compilation needed more than the 16GB of memory installed in the computer). The *Glider* and *Random* models correspond to the Game of Life model described using *CellularAutomataLib2*, the former initialized with the same glider structure and the latter randomly activating the half of the cells of the whole space. Finally, the *Furnace* model represents a simple model of a heat diffusion included in the library, with the top row of cells set initially active. The size of the C code generated by Dymola is also included in the table to illustrate the effect in the generated code of using external functions to describe the CA model. Note that the size of the code for *CellularAutomataLib2* models is dependent on the number of initially active cells. It is easy to observe in the table that the simulation time for the *Glider* model is almost constant, since the glider is a repetitive structure and the number of active cells is almost constant. The scalability of the Random and

Table 1. Simulation time and size of C code generated by Dymola for Modelica CA models (N^2 is the number of cells in a 2D space).

Model	N				
	10	100	200	300	400
Custom	64.9KB 0.00464s	6.9MB 0.222s	28.2MB 1.079s	65.4MB 2.24s	117.3MB -
Glider	14.9KB 0.00291s	14.9KB 0.00314s	14.9KB 0.00313s	14.9KB 0.00334s	14.9KB 0.00324s
Random	14.3KB 0.00322s	14.3KB 0.048s	14.3KB 0.221s	14.3KB 0.947s	14.3KB 2.21s
Furnace	15.8KB 0.00476	32.9KB 0.0383s	52.4KB 0.0744s	71.9KB 0.111s	91.5KB 0.152s

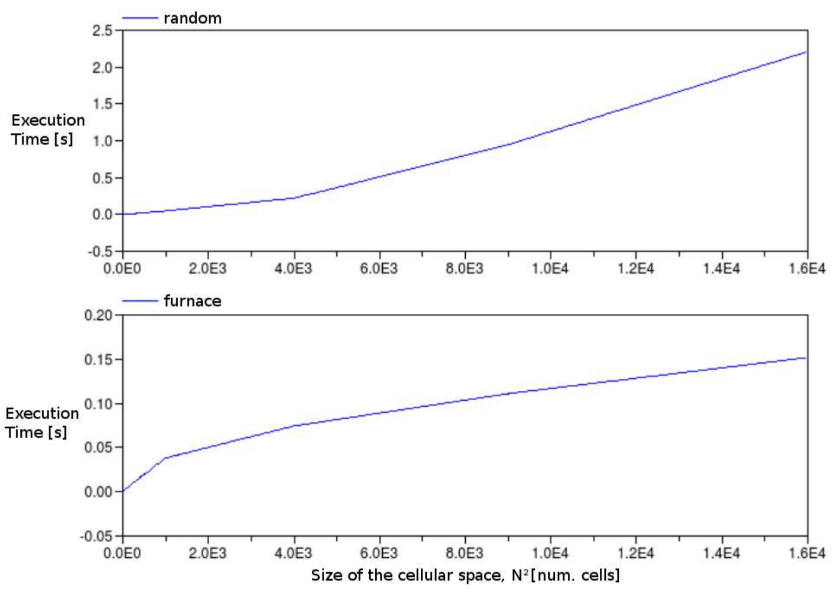

Figure 1. Scalability of *Random* and *Furnace* models.

Furnace models is also almost linear (cf. Fig. 1). The performance of these models is dependent on the size of the space because of its initialization (i.e., the bigger the space, the more initially active cells).

- Also, the new version of the library supports hexagonal lattice structures, that facilitates the description of some LGCA models. An example is shown in Section 5.

3.2. *CA Modeling*

Following the formal description of the automata, a model in *CellularAutomataLib2* is composed of a *cellular space*, that represents the 1D or 2D grid of cells, including their state (S) and the transition function (δ), and some models, named *interface models* (i.e., inputs X, Ω, and outputs Y, λ), used as interface between different cellular spaces, or between cellular spaces and other models.

CellularAutomataLib2 models are composed of a combination of Modelica code and C code. Modelica provides an external function interface that can be used to call external functions written in the C language from Modelica functions [27]. Cellular spaces and interface models are described using Modelica. However, the behavior of cellular spaces and interfaces, and the CA simulation algorithm are implemented using external C functions that are related to their corresponding Modelica models using the mentioned external function interface. The external C code is not involved in the automatic translation of the Modelica model, which improves the scalability and the performance of the simulations.

The relationship between the external C code and the Modelica code is summarized in Fig. 2. The file `CellularAutomataLib.c` contains C code used to describe general data structures (e.g., cellular spaces and cells) and functions to implement the CA simulation algorithm. These data structures and functions are common to all CA and should not be modified by the user. The behavior of the model is defined in another file (e.g., `Model.c`), where the user has to implement the data structure used to describe the state of the cells, its default and initial values and the transition

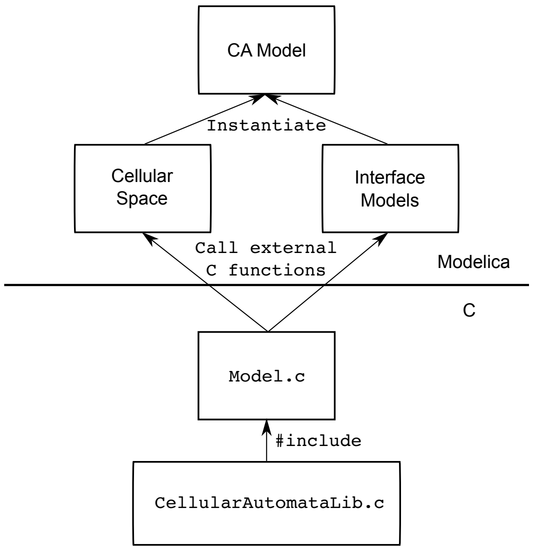

Figure 2. Relationship between Modelica and C code in *CellularAutomataLib2*.

rule. `CellularAutomataLib.c` has to be included in `Model.c` in order to use the basic functionality of the library. Once the behavior of the automaton is described in C, the user can describe the cellular spaces and their interfaces using Modelica and relate them with the behavior defined in C. Finally, cellular spaces and interface models can be combined, by means of instances, to describe more complex CA models (e.g., that may include multiple inter-connected cellular spaces and interfaces). A detailed description of the procedure to construct new CA models is included in the documentation of the library.

3.3. *Interfacing with Other Modelica Models*

CellularAutomataLib2 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 the *Input Region* and *External Input Region* models. The *Input Region* model allows to use the state of a region of cells from an automaton as inputs for the transition function of another region of cells of another automaton. Similarly, the *External Input Region* model allows to use a Real value, generated in another Modelica model, as an input for the transition function of a region of cells of an automaton. The outputs of the CA model (Y, λ) are described using the *Output Region* model. This model allows to observe the state of the automaton in different ways and use the observed state as an input for another Modelica model.

The technical details about the behavior of these interface models and their implementation are described in the documentation of the library. However, in order to illustrate the use of these models, and example is shown in Fig. 3. In this example, the following interface models are used:

- Model *extInitRegion* 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* (i.e., 0).
- Model *ExtInputRegion*, connected to *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.

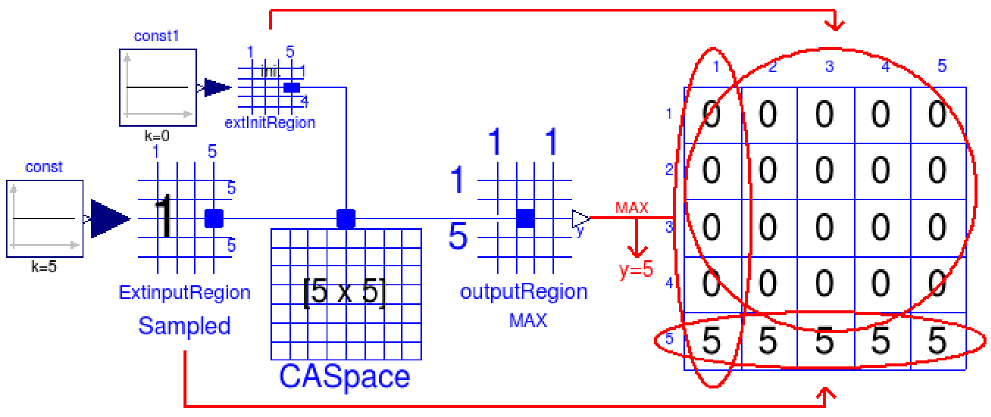

Figure 3. Example of behavior of external input region, external init region and output region models.

- Model *outputRegion* 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.

4. Application Example 1: Forest Fire Spread Model

This model describes the evolution of a fire in a two dimensional space that represents a forest. Modeling fire spread in wild-lands provides useful information in order to fight against it, e.g., if the simulations are used when the fire has already started, or to design appropriate preemptive actions, such as evacuation plans, distribution of fire suppression resources and strategies. Cellular automata have been previously applied to model fire spread [28–30]

The model implemented using *CellularAutomataLib2* is shown in Fig. 4. The forest model, named *Forest*, is connected to three external input regions that are used to set the changes in the direction (*WindDir* model) and speed (*WindSpeed* model) of the wind during the simulation and also the fire suppression actions (*Water* model). The forest model is connected to an output region (*FireSize* model), that is used to observe the evolution of the fire. The details of these models are discussed next.

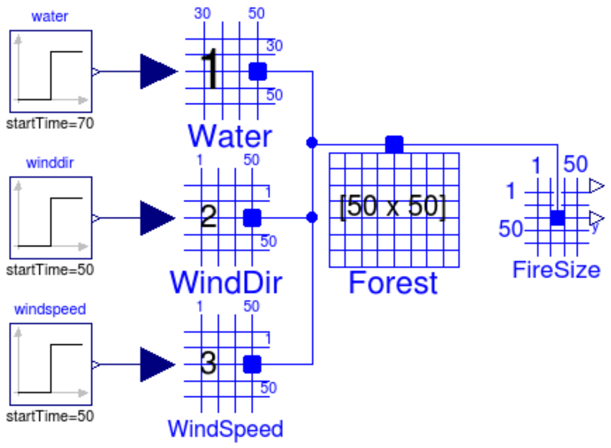

Figure 4. Fire spread model implemented using *CellularAutomataLib2*.

4.1. *Forest Model*

The *Forest* model has been described as a 2D cellular space using the `CellSpace2D` model from the library. Each cell represents a uniform section of the forest of 15 m × 15 m. The parameters of the model are shown in Table 2.

As detailed above, the behavior of the model has to be described using external C functions that are used in the Modelica model. These functions are included in the file `forest.c`. The state of the cell is described using the following variables: `fuelmodel` (type of fuel bed), `b` (data corresponding to the fuel model), `s` (cell phase: unburned, burning or burned), `wet` (if the forest in the cell is wet or not), `windspeed` (wind speed) and `winddir` (the direction the wind is blowing from). By

Table 2. Parameters of the *Forest* cellular space.

Name	Value	Description
`ncols`	50	Number of columns
`nrows`	50	Number of rows
`lattice`	0	Square space
`neighborhood`	[-1,-1; -1,0; -1,1; 0,1; 1,1; 1,0; 1,-1; 0,-1]	Moore's
`n_inputs`	3	Three external input regions connected
`wrapped_borders`	0	No wrapped borders
`Tstep`	1	One step per simulated second
`initial_step`	0	Initialization time
`plot_animation`	1	Animation active
`plot_range`	2	Max value displayed in the animation
`init_cells`	[40,25]	Initial cell
`name`	"Forest"	

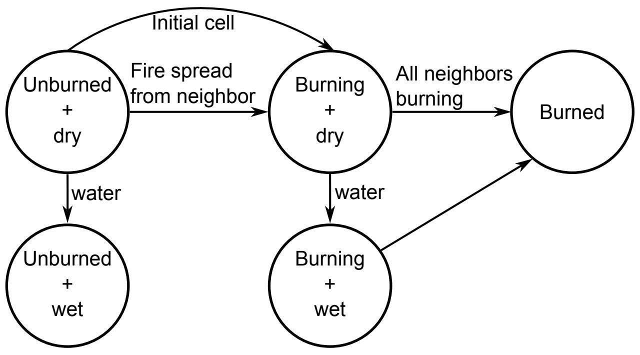

Figure 5. Diagram that describes the behavior of the cell and its phase transitions.

default, all the cells are dry and unburned, and the wind blows from the S (180°) at 8 km/h. The characteristics of the fuel are defined by the fuel model used. The fuel models defined by the National Forest Fire Laboratory (NFFL) have been used, in particular the model NFFL-4 named *Chaparral* [31].

The behavior of the cell is described by the diagram shown in Fig. 5. Fire starts burning in the cells indicated in `init_cells`. The fire will spread to the neighboring dry cells at the calculated rate of spread (*ros*), depending on the type of fuel and the weather conditions (wind speed and direction). When all the neighbors of a burning cell become burning, the fuel in the cell is supposed to be depleted and it becomes burned.

The *ros* is calculated using the Java implementation, named BEHAVE, of the Rothermel surface fire spread model [32] developed by Andreas Bachmann [33]. In this case, the Java code has been translated into C in order to combine it with the external C functions used in *CellularAutomataLib2*. A similar approach is adopted in [30] by combining the BEHAVE model with a DEVS-based forest fire spread model implemented using DEVSJAVA.

The calculated *ros* is uniform in the absence of wind and the propagation of the fire has a circular shape. The fire will spread to the N, S, W and E neighbors (i.e., those neighbors in the vertical and horizontal axis) in $15/ros$ minutes and to the NE, NW, SE and SW neighbors (i.e., those neighbors in the diagonals) in $21.2132/ros$ minutes, because diagonals suppose longer distances. In the presence of wind, the propagation of the fire has an elliptical shape following the direction of the wind. The BEHAVE model calculates the direction of maximum spread, the rate of spread in that direction and the eccentricity of the ellipsis. These values are used to calculate the effective rate of spread in the direction of each neighbor, and thus the time that will take to spread the fire to each neighbor.

If water is poured in a cell, for example due to a firefighting action, the cell becomes wet. Unburned and wet cells remain in that state and fire cannot propagate across them. A burning cell that becomes wet is set to the burned state, because the fire is suppressed but some fuel was already burned.

4.2. *Interface Models*

As mentioned above, four interface models are included. Three external input regions are used to describe the variations in the environmental conditions (i.e., direction and speed of the wind), and the firefighting actions performed (i.e., pouring water on the forest). An output region is used to observe the evolution of the fire by measuring its size (i.e., the sum of burning and already burned cells).

The inputs to the external input region are described using three *Step* models from the Modelica Standard Library [34]. However, more complex Modelica models could be used and connected as inputs to the external input regions for representing the environmental conditions and the firefighting actions. Similarly, the output of the *FireSize* model could be connected to another Modelica model in order to make use of the information provided by the forest fire model (e.g., using the size of the fire as an input for a firefighting resources management model). In this way, *CellularAutomataLib2* facilitates the connection of CA models with other Modelica models for describing complex hybrid models.

4.3. *Simulation Results*

The model parameters have been set to the values shown in Table 2. A cellular space of 50x50 is used to facilitate the visualization of the simulation results. *CellularAutomataLib2* includes the same example with a cellular space of 500x500 in order to demonstrate that larger CA models can be simulated. It can be remarked that both, the 50x50 and the 500x500 forest fire models, generate a source code of 35.8KB because the structure of the automaton is coded using external C functions and thus they are not handled by the Modelica tool. Similarly, the Game of Life model simulated using *CellularAutomataLib2* with a size of 500x500 generates a source code of 10.6KB.

The external input regions are configured as follows:

- Wind speed changes when time equals 50 min from 8 km/h to 3 km/h.
- Wind direction changes when time equals 50 min from 180° (i.e., blowing from the S) to 270° (i.e., blowing from the W).
- Firefighters pour water when time equals 70 min over the cells in the region defined between rows 30 and 50 and columns 30 and 50.

The simulation is run for 120 min of simulated time, and is executed in 1.6s using Dymola 2015. The evolution of the fire is shown in Fig. 6. The results obtained are equivalent to those obtained in [29] and [30], where the Rothermel fire spread model is also used. Note the change in the wind direction and speed when time equals 50 min, that makes the fire to spread towards E and at a lower rate. Also note the firefighting action of pouring water when time equals 70 min, that prevents the fire to spread across the poured region. Cells that were already burning inside the poured region remain burned. The evolution of the size of the fire is shown in Fig. 7. It can be observed that the growing rate of the fire decreases since time equals 70 min due to the firefighting action.

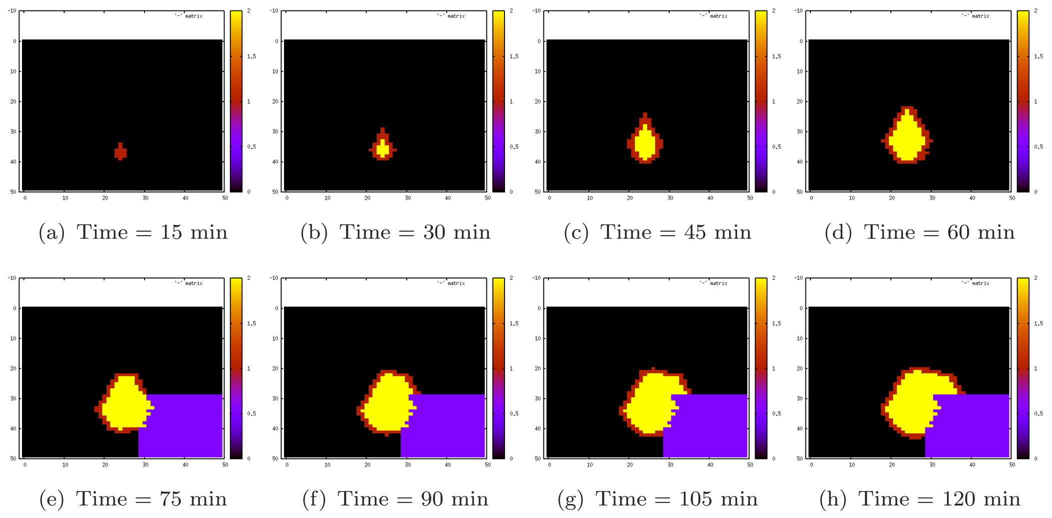

Figure 6. Evolution of the fire across the cellular space.

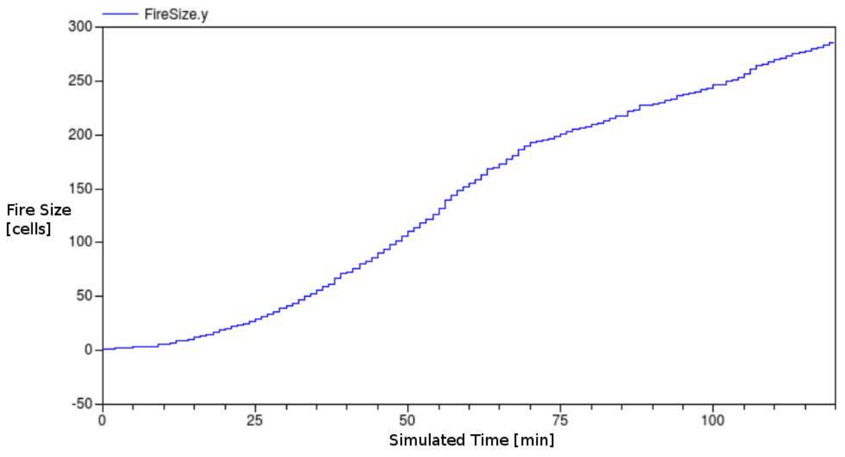

Figure 7. Evolution of the fire size.

4.4. *Scalability*

In order to analyze the scalability of the simulation, experiments with an increasing simulated time have been performed. The model is configured with a 1000x1000 cellular space and unwrapped boundaries. The cell [900,500] is the only one initially set active. The model has been simulated from 100 min to 1900 min, with increments of 200 min (each simulated step in the automata represents 1 min). In this situation, the number of active cells in the automaton grows with the simulated time, as the fire is spread across the forest. The execution time for each experiment is shown in Table 3, and also graphically in Fig. 8. The execution time depends on the number of active cells in each simulation step. The execution time for each simulation step has been computed and is shown in Fig. 9, together with the evolution of the number of active cells. Note that the execution time for each step grows almost linearly with the number of active cells in the space.

Table 3. Execution time for the forest fire spread experiments with increasing simulated time.

Sim. Time	100	300	500	700	900	1100	1300	1500	1700	1900
Exec. Time	0.00616s	0.0379s	0.102s	0.219s	0.411s	0.677s	0.977s	1.41s	1.89s	2.48s

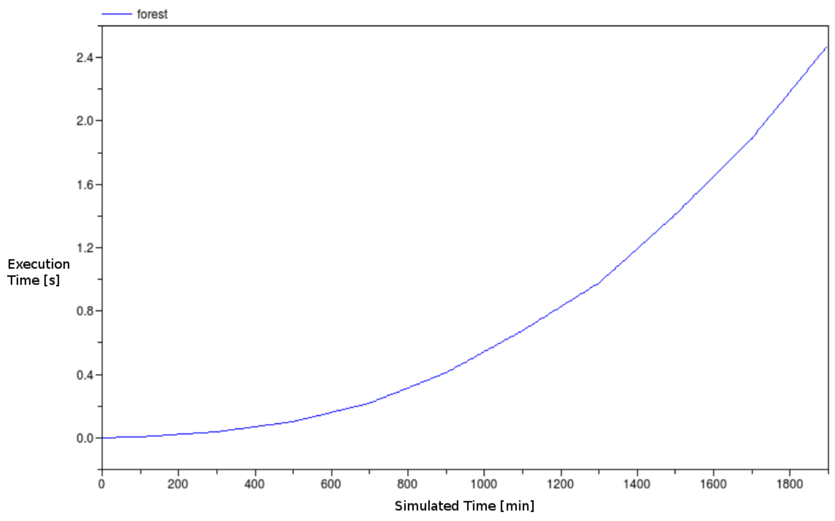

Figure 8. Execution time of forest fire spread experiments with increasing size of simulated time.

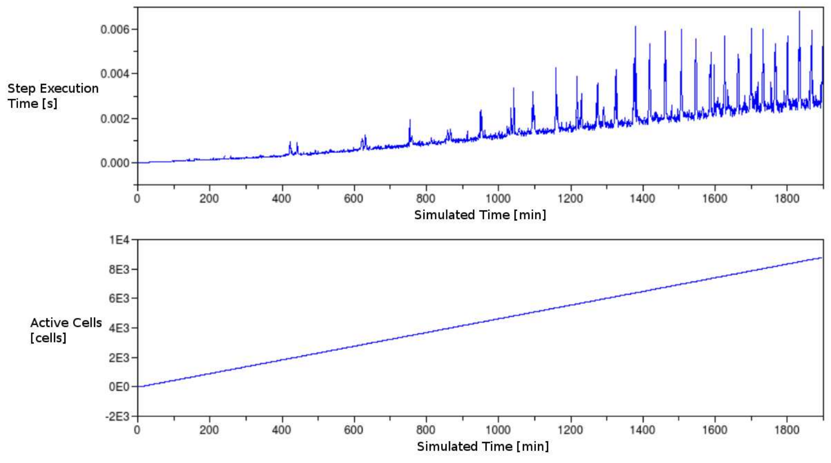

Figure 9. Evolution of the step execution time compared with the number of active cells.

5. Application Example 2: SIR-type Epidemic Spread (C17 ARGESIM Benchmark)

The following model represents a Susceptible-Infected-Recovered (SIR) system used to evaluate the spread of an epidemic across a population of individuals. This model was proposed by ARGESIM as a benchmark for comparing simulation results between ODE and LGCA models [19]. It serves as a demonstration of the *Cellular-AutomataLib2* functionality to describe LGCA models.

The SIR model is defined by the following equations:

$$\begin{aligned}\frac{\partial S(t)}{\partial t} &= -r \cdot S(t) \cdot I(t) \\ \frac{\partial I(t)}{\partial t} &= r \cdot S(t) \cdot I(t) - a \cdot I(t) \\ \frac{\partial R(t)}{\partial t} &= a \cdot I(t)\end{aligned} \tag{1}$$

where r is the infection rate, a is the recovery rate, $S(t)$ represents the number of individuals susceptible of being infected, $I(t)$ represents the number of infected individuals and $R(t)$ represents the number or individuals recovered from the illness, and therefore are considered immune. These equations suppose a simplified version of the model, where the population size is considered constant (e.g., no births or deaths), the incubation time of the infectious agent is zero and the infectivity time is equal to the duration of the disease.

The benchmark proposes to compare the ODE model with two LGCA models: the HPP [35] and the FHP [36]. The HPP model is a lattice gas cellular automaton represented over a square lattice using the von Neumann neighborhood (i.e., four adjacent cells). The state of each cell is represented by at most four particles, each one oriented in the direction of a different neighbor. The transition rule is defined in two phases: propagation, where each particle moves to the neighbor in the direction of its orientation; and collision, if and only if two particles collide in the same cell from opposing directions their orientation is changed by 90.

The FHP model is a lattice gas cellular automaton represented over an hexagonal lattice using the Moore's neighborhood (i.e., the six closest neighbors). The state of the cell is now represented by at most six particles, each one also oriented in the direction of a different neighbor. The transition rule is analogous to the HPP model, but in this case the collision involves more possibilities. The FHP-I collision rules are used, meaning that:

(1) Two particles that collide from opposing directions will change their orientation by 60, either clockwise or counter clockwise but the same for both particles.
(2) Three particles that collide from opposite directions, each one separated by 120, will change their orientation by 60, also clockwise or counter clockwise but equal for the three.

In order to represent the SIR model, each particle in the LGCA model represents an individual of a given type (i.e., susceptible, infected or recovered).

5.1. *HPP and FHP Models using* CellularAutomataLib2

The behavior of the HPP and FHP models has been included in the files `c17hpp.c` and `c17fhp.c`, respectively. The corresponding Modelica models are included in *CellularAutomataLib2* as examples of 2D CA (i.e. `CellularAutomataLib2.Examples2D.C17` and `.C19`). The development of the C code and the Modelica model are described next.

The implementation of the HPP model uses four integer variables to describe the particles, each oriented to one direction (N, S, E and W). A positive value of these variables identifies the existence of the particle in the cell, and the value represents the type of individual represented (1 susceptible, 2 infected and 3 recovered). The default state function sets the cell state as empty (i.e., all particles with value 0). The initial state function is used to uniformly distribute a pre-defined number of each type of individual in the cellular space. The transition function implements the propagation and collision of particles in the automaton, as well as the infection and recovery of susceptible and infected individuals, respectively, following Eq.(1).

The implementation of the FHP model uses an array of six integers to describe the particles, as an alternative to use individual variables for each particle like in the HPP model. The default and initial state functions are equivalent to the HPP model. The transition function is also equivalent but including the additional two- and three-particle collision cases.

Table 4. Parameters of the HPP and FHP cellular spaces.

Name	Value	
	HPP	FHP
`ncols`	100	100
`nrows`	100	100
`lattice`	0	1
`neighborhood`	[-1,0; 0,1; 1,0; 0,-1]	Moore's
`n_inputs`	0	0
`wrapped_borders`	3	3
`Tstep`	1	1
`initial_step`	0	0
`plot_animation`	1	1
`plot_range`	3	3
`init_cells`	[nrows,ncols]	[nrows,ncols]
`name`	"HPP"	"FHP"

Once the behavior of the HPP and FHP models is implemented in C, this code has to be linked with Modelica. Both Modelica models extend the cellular space

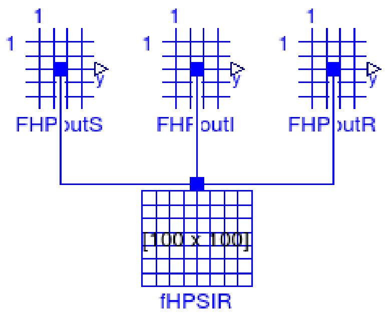

Figure 10. FHP cellular space and output region models.

model `CellSpace2D` and are configured using the parameters shown in Table 4. The `Default`, `Initial` and `Rule` functions of the cellular space are redeclared with new functions that call the defined C functions. In order to observe the evolution of the number of each type of individuals, three output region models are connected to the cellular space. Each output region is configured to sum the values of each type of individual in the whole automaton (i.e., S, I and R, respectively). The final structure of the FHP model, including the output regions, is shown in Fig. 10.

5.2. *Simulation Results*

Three simulation experiments, described as "tasks" in the benchmark, have been performed. The results obtained using *CellularAutomataLib2* are analogous to previous solutions of the same model [37].

Task A compares the results of the ODE model and the LGCA models using the initial conditions shown in Table 5. The comparison of the simulation results (number of susceptible, infected and recovered individuals) is shown in Fig. 11. The results from the ODE model are shown at the top, the FHP model in the middle and the HPP model at the bottom. Note that the propagation of the epidemic is slower in the LGCA models due to the influence of the spatial distribution of infected individuals, which is not considered in the ODE model. The propagation dynamics in the HPP model is actually slower than the FHP because of the smaller neighborhood used, that reduces the probability of infection (i.e., contact between susceptible and infected individuals).

Table 5. Initial conditions and parameters for Task A.

Name	Value
$S(t=0)=S_0$	16000
$I(t=0)=I_0$	100
$R(t=0)=R_0$	0
Infection rate r	$0.6 \cdot 10^4$
Recovery rate a	0.2

Task B is used to evaluate the influence of vaccination strategies in the CA models. In this case, the infected individuals are distributed across the half of the cellular space. The benchmark proposes the comparison of three vaccination strategies: a) *full*, uniformly distribute vaccinated individuals in the whole space; b) *half*, uniformly distribute vaccinated individuals in the half occupied by infected ones; and c) *border*, distribute the vaccinated individuals in the border of the half occupied by infected ones (i.e., surrounding the infected). The number of vaccinated individuals is 4000, that will be simulated as initially recovered individuals because for practical uses both are considered immune. The initial distribution of vaccinated individuals is shown in Fig. 12.

The comparison of the simulation results (number of infected individuals for each vaccination strategy) is shown in Fig. 13. Note that while the *full* and *half* vaccination strategies offer similar results, the *border* strategy reduces the maximum number of infected individuals but increases the duration of the disease. In the border strategy the infection is contained by the line of vaccinated individuals but when it crosses the vaccination line it spreads in across the whole space, increasing the duration of the disease.

A final Task, C, is proposed as an experiment to approximate the solutions of the

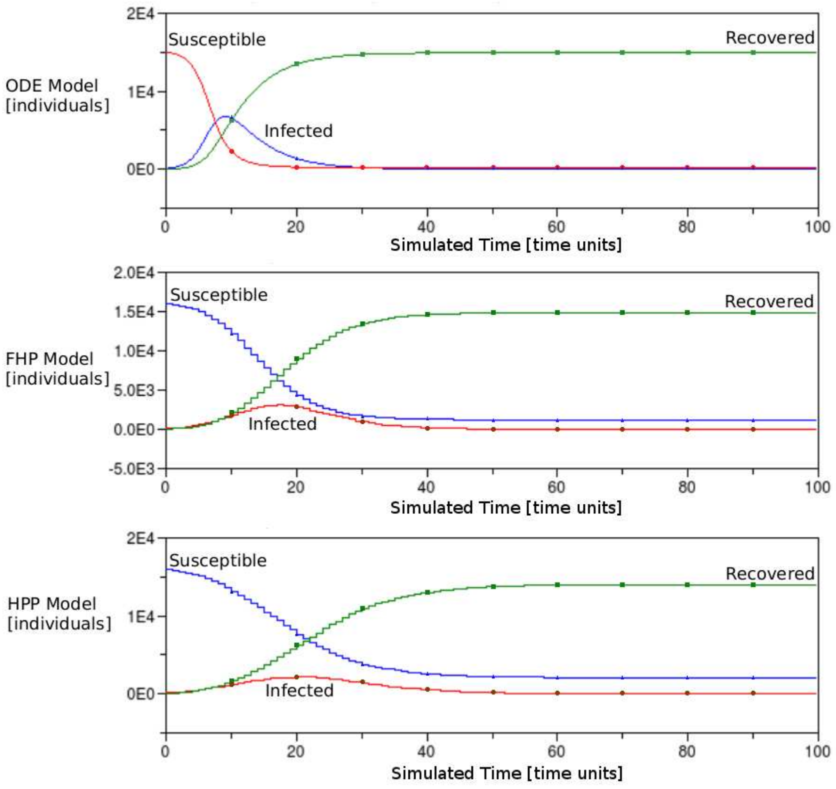

Figure 11. Task A simulation results.

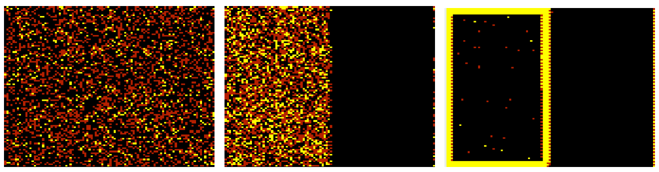

Figure 12. Initial distribution of vaccinated individuals: left) full distribution; center) half distribution; and right) border distribution.

ODE and the FHP models. The spatial distribution of the individuals of the FHP model delays the spread of the infection. A uniform re-arrange of the population in the FHP model after each step in the simulation is proposed to avoid the influence of the spatial distribution. Each individual in the space, of either type, is randomly re-distributed to another position in the space. This re-distribution is implemented

Table 6. Initial conditions and parameters for Task C.

Name	Value
$S(t=0)$=S_0	40000
$I(t=0)$=I_0	1000
$R(t=0)$=R_0	0
Infection rate r	$0.3 \cdot 10^4$
Recovery rate a	0.2

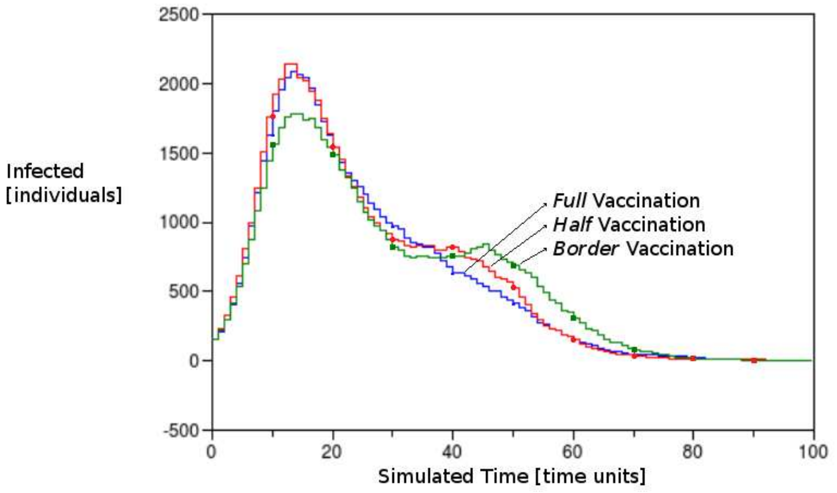

Figure 13. Task B simulation results. Number of infected individuals for each vaccination strategy.

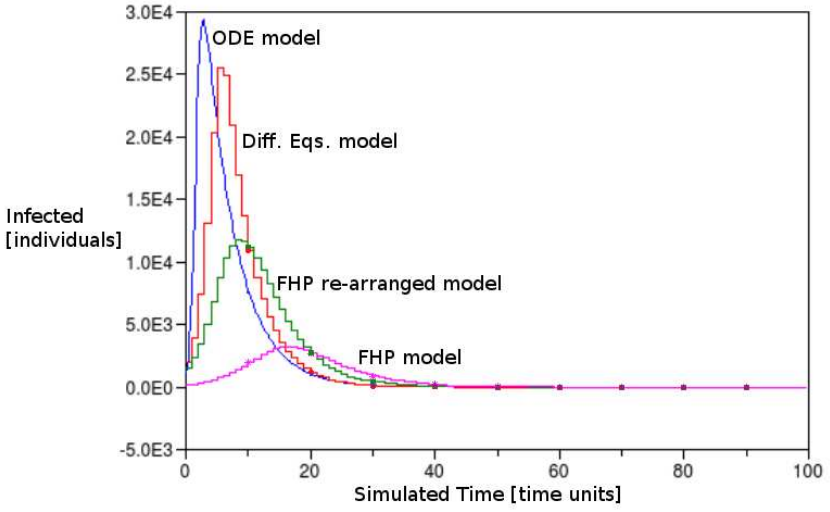

Figure 14. Task C simulation results. Number of infected individuals for the ODE, Diff. Eqs., re-arranged and normal FHP models.

by modifying the C function that is used to re-declare the `Rule` function of the FHP cellular space. Also the parameters of the model are modified, as shown in Table 6.

The simulation results are shown in Fig. 14. Observe that the spread of the infection is faster when the population is randomly re-arranged than in the normal FHP model, with values closer to the ODE result. The difference equation SIR model described in [19] is also compared, because its solution serves as an upper bound for the automaton [37].

5.3. *Scalability*

In order to analyze the scalability of the simulation, experiments with an increasing size of the cellular space have been performed. The initial individuals are randomly

Table 7. Execution time and initial values for the FHP experiments (N is the size of the space).

N	10^4	4×10^4	1.6×10^5	2.5×10^5	4.9×10^5	6.4×10^5	8.1×10^5	10^6	1.44×10^6	1.96×10^6
Exec. Time	0.286s	1.9s	10.5s	17.9s	25.4s	33.1s	47.9s	62.3s	109s	150s
Initial S	16×10^4		16×10^5		16×10^6					
Initial I	10^3		10^4		10^5					

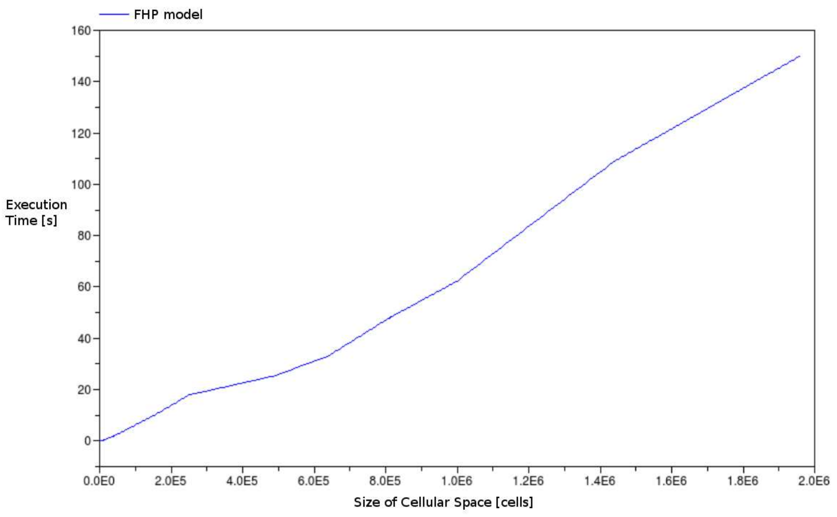

Figure 15. Execution time of FHP experiments with increasing size of cellular space.

distributed over the cellular space, and so, the number of active cells depends on the size of the space. In order to ensure that the number of active cells matches the size of the space, the initial number of individuals has been increased as shown in Table 7. Each experiment has been simulated for 100 time units. The size of the cellular space ranges from 10^4 to 1.96×10^6 cells. The execution time for each experiment is shown in Table 7. Also, the same values are graphically represented in Fig. 15. Note that the execution time grows almost linearly with the number of cells in the space.

6. Conclusions

Cellular Automata (CA) models can be helpful in several simulation problems, and since most of these problems are multi-physical and benefit from a component-based, object-oriented approach, the integration of CA models in languages like Modelica is an interesting and promising topic. Along this reasoning path, and building on previous preliminary research results, the *CellularAutomataLib2* library, developed to facilitate the description of CA models, was presented here. The functionality of this new library is focused on the description of the behavior of the model, while the CA simulation algorithm remains transparent to the user – an important characteristic indeed for somebody who is more an expert of the addressed modeling domain than of simulation techniques.

The library has been implemented as a combination of Modelica code, that is used to describe the model and its interface with other Modelica models, and external C code, that is used to describe the state and behavior of the automaton and the data translation performed in the interface. The use of external C code to describe models improves the scalability and simulation performance of the library. The scalability is improved because the external code is not involved in the automatic translation procedure of the Modelica code, reducing the size and complexity of the generated code, and the time spent in the translation of the model. The simulation performance is improved because: 1) the simulation of the CA is performed using a custom algorithm included in *CellularAutomataLib2*, reducing the number of events managed by the Modelica simulation algorithm and thus the number of event iterations performed; and 2) the number of equations involved in the event iteration is reduced due to the use of C code to describe the behavior of the automaton. The functionality of the library has been demonstrated by means of a fire spread model and a SIR-type epidemic spread model.

Some future work ideas are: to support the description of 3D models and additional extended CA formalisms, such as CA on graphs; 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.

Acknowledgements

This research was supported by Universidad Nacional de Educación a Distancia, 2013-026-UNED-PROY grant, and by the Ministerio de Economía y Competitividad of Spain, DPI2013-42941-R grant.

References

[1] Modelica Association, *Modelica - An unified object-oriented language for physical systems modeling. Language specification version 3.3* (2013), Available at http://www.modelica.org/documents.

[2] K.J. Åström, H. Elmqvist, and S.E. Mattsson, *Evolution of continuous-time modeling and simulation*, in *Proceedings of the* 12th *European Simulation Multiconference (ESM'98)*, Manchester, UK, 1998, pp. 9–18.

[3] F.E. Cellier and E. Kofman, *Continuous System Simulation*, Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[4] F. Dshabarow, *Support for Dymola in the modeling and simulation of physical systems with distributed parameters*, Master's thesis, ETH Zürich, 2007.

[5] F. Schiavo, L. Vigan, and G. Ferretti, *Object-oriented modelling of flexible beams*, Multibody Syst. Dyn. 15 (2006), pp. 263–286.

[6] S. Micheletti, S. Perotto, and F. Schiavo, *Modelling heat exchangers by the Finite Element Method with grid adaption in Modelica*, in *Proceedings of the* 4th *International Modelica Conference*, Hamburg, Germany, 2005, pp. 219–228.

[7] B.E. Hefni, D. Bouskela, and G. Gentilini, *Dynamic modelling of a condenser/water heater with the Thermo-SysPro Library*, in *Proceedings of the* 9th *International Modelica Conference*, Munich, Germany, 2012, pp. 745–758.

[8] S. Quoilin, A. Desideri, J. Wronski, I. Bell, and V. Lemort, *ThermoCycle: A Modelica library for the simulation of thermodynamic systems*, in *Proceedings of the* 10th *International Modelica Conference*, Lund, Sweden, 2006, pp. 85–95.

[9] Dassault Systèmes AB, *Dymola, Dynamic Modeling Laboratory. User's manual* (2014), Available at http://www.dymola.com/.

[10] P. Fritzson, P. Aronsson, P. Bunus, V. Engelson, L. Saldamli, H. Johansson, and A.

Karström, *The Open Source Modelica Project*, in *Proceedings of the* 2nd *International Modelica Conference*, Oberpfaffenhofen, Germany, 2002, pp. 297–306.

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

[12] L. Saldamli, *PDEModelica - a high-level language for modeling with partial differential equations*, Ph.D. thesis, Dept. of Computer Science and Information Science, Linköping University, 2006.

[13] Modelica Association Project FMI, *Functional mock-up interface for model exchange and co-simulation (v2.0)* (2014), Available at http://www.fmi-standard.org.

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

[15] H.L.M. Vangheluwe and G.C. Vansteenkiste, *The cellular automata formalism and its relationship to DEVS*, in *In* 14th *European Simulation Multi-conference (ESM)*, Ghent, Belgium, 2000, pp. 800–810.

[16] D.A. Wolf-Gladrow (ed.), *Lattice Gas Cellular Automata and Lattice Boltzmann Models: An Introducction*, Springer-Verlag, Berlin, 2000.

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

[18] V. Sanz, A. Urquia, and A. Leva, *1D/2D cellular automata modeling with Modelica*, in *Proceedings of the* 10th *International Modelica Conference*, Lund, Sweden, 2014, pp. 489–498.

[19] H. Hötzendorfer, N. Popper, and F. Breitenecker, *Temporal and spatial evolution of SIR-type epidemic – ARGESIM comparison c17 – definition*, Simulation News Europe 41/42 (2004), pp. 42–44.

[20] V. Sanz and A. Urquia, *An approach to cellular automata modeling in Modelica*, in *Proceedings of the* 5th *International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools*, Nottingham, UK, 2013, pp. 121–130.

[21] V. Sanz, A. Urquia, F.E. Cellier, and S. Dormido, *System modeling using the Parallel DEVS formalism and the Modelica language*, Simul. Model. Pract. Th. 18 (2010), pp. 998–1018.

[22] V. Sanz, A. Urquia, and S. Dormido, *Parallel DEVS and process-oriented modeling in Modelica*, in *Proceedings of the* 7th *International Modelica Conference*, Como, Italy, 2009, pp. 96–107.

[23] V. Sanz, *Hybrid system modeling using the Parallel DEVS formalism and the Modelica language*, Ph.D. thesis, ETSI Informática, UNED, Madrid, Spain, 2010.

[24] W. Braun, B. Bachmann, and S. Pross, *Synchronous events in the OpenModelica compiler with a Petri Net library application*, in *Proceedings of the* 3rd *International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools*, Oslo, Norway, 2010, pp. 63–70.

[25] V. Sanz, A. Urquia, and F. Casella, *Improving efficiency of hybrid system simulation in Modelica*, in *Proceedings of the* 6th *International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools*, Berlin, Germany, 2014, pp. 21–28.

[26] T. Williams and C. Kelley, *Gnuplot 4.6: An interactive plotting program* (2012), Available at http://www.gnuplot.info.

[27] H. Olsson, *External interface to Modelica in Dymola*, in *Proceedings of the* 4th *International Modelica Conference*, Hamburg, Germany, 2005, pp. 603–611.

[28] A.H. Encinas, L.H. Encinas, S.H. White, A.M. del Rey, and G.R. Sánchez, *Simulation of forest fire fronts using cellular automata*, Adv. Eng. Softw. 38 (2007), pp. 372–378.

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

[30] L. Ntaimo, B.P. Zeigler, M.J. Vasconcelos, and B. Khargharia, *Forest fire spread and suppression in DEVS*, SIMULATION 80 (2004), pp. 479–500.

[31] H.E. Anderson, *Aids to determining fuel models for estimating fire behavior*, Tech. Rep. INT-122, USDA Forest Service, Intermountain Forest and Range Experiment Station,

Ogden, UT, 2003.

[32] R.C. Rothermel, *A Mathematical Model for Predicting Fire Spread in Wildlands Fuels*, USDA Forest Service, Research Paper INT-115, 1972.

[33] A. Bachmann, *GIS-based wildland fire risk analysis*, Ph.D. thesis, Department of Geography, University of Zurich, Zurich, Switzerland, 2001.

[34] *Modelica standard library* (2014), Available at https://github.com/modelica/Modelica.

[35] J. Hardy, Y. Pomeau, and O. de Pazzis, *Time evolution of a two-dimensional model system I. Invariant states and time correlation functions*, J. Math. Phys. 14 (1973), pp. 1746–1759.

[36] U. Frisch, B. Hasslacher, and Y. Pomeau, *Lattice-gas automata for the Navier-Stokes equation*, Phys. Rev. Lett. 56 (1986), pp. 1505–1508.

[37] H. Hötzendorfer and F. Breitenecker, *A directly programmed implementation of ARGESIM comparison c17 "SIR-type epidemic" using MATLAB*, Simulation News Europe 41/42 (2004), p. 45.