Cellular Automata DEVS: A Modeling, Simulation, and Visualization Environment

Chao Zhang
Arizona Center for Integrative Modeling and Simulation, School of Computing, Informatics, and Decision Systems Engineering, Arizona State University
Tempe, Arizona
czhang72@asu.edu

Hessam S. Sarjoughian
Arizona Center for Integrative Modeling and Simulation, School of Computing, Informatics, and Decision Systems Engineering, Arizona State University
Tempe, Arizona
sarjoughian@asu.edu

ABSTRACT

Cellular Automata (CA) models are represented as a collection of independent dynamical cells having some specific spatial relationship to each other. These tessellation automata can have simple to complex behaviors due to both individual cell behaviors as well as their interactions. Code debugging, supported by advanced software development tools, is needed for developing CAs owing their complex dynamics to cells that have non-trivial event handling and timing. As such, it is useful to debug models during simulation through step-by-step examination of any number of cells using rich control and visualization means. In this paper, we show the CA-DEVS framework where cell and Cellular Automata models are derived from atomic and coupled Parallel DEVS models. This framework uniquely supports visualizations using run-time generation of input, output, and state linear and superdense time trajectories as well as run-time spatial animation with playback. Multi-modal visualization capabilities allow examining behavior of any number of cells independent of any other cell. We describe some key parts of the architectural design of the CA-DEVS and highlight some ongoing and future research.

CCS CONCEPTS

• **Computing methodologies** → **Modeling and simulation**; **Simulation by animation**; **Simulation tools**; *Modeling methodologies*; *Discrete-event simulation*; *Interactive simulation*; *Simulation environments*;

KEYWORDS

Cellular Automata, CA-DEVS, Cell-DEVS, DEVS-Suite, MASON, Mathematica, Simulator, Multi-Mode Visualization, Model-Façade-View-Control Architecture

ACM Reference Format:
Chao Zhang and Hessam S. Sarjoughian. 2017. Cellular Automata DEVS: A Modeling, Simulation, and Visualization Environment. In *Proceedings of 10th EAI International Conference on Simulation Tools and Techniques (SIMUTOOLS '17).* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3173519.3173534

SIMUTOOLS '17, September, 2017, Hong Kong, China

ACM ISBN 978-1-4503-6388-4/17/09…$15.00
https://doi.org/10.1145/3173519.3173534

1 INTRODUCTION

It has become common to develop simulation models exhibiting complex behaviors rooted in individual parts and their relationships. This is in contrast to simulation models such as Game of Life that are built from parts that have simple behavior and structure and yet collectively reveal intricate dynamics. As such it is helpful to have modeling and simulation methodologies that are not only strongly grounded in mathematical theories, but also complemented with expressive frameworks. Theory alone or a framework that treat modeling and simulation as "software programs" is limited. Considering a prototypical lifecycle for developing simulation models from concepts to execution and evaluation, it is understandable that formal specification combined with advanced code development frameworks such as Eclipse are necessary, but insufficient when systems to be modeled, simulated, and evaluated are inherently complex-i.e., hidden dynamics of simulations can be understood through use of formal model specifications with accurate simulation execution protocol, and configurable evaluation regimes including run-time time-based trajectory views.

While there exist a variety of methods, frameworks, and tools that very well support modeling and simulation of many kinds of complex systems (e.g., Smart Supply-Chain Enterprises belonging to Internet-of-Things), it is difficult to state the same for having tools that systematically aid users to evaluate results. Specifically, it is becoming necessary to "debug models" as opposed to "debug programs". That is, it is useful to have frameworks with tools that can aid users inspect the dynamics of any part of a model during simulation. Examining states, inputs, and outputs as time trajectories of individual parts, or input/output interactions among parts as animation, or dynamics of whole systems in a step-by-step manner through playback can be useful. Such capabilities for component-based, hierarchical DEVS models are already supported in the DEVS-Suite Simulator [1, 7].

Given the continuing importance and use of Cellular Automata (CA) models [19], it is beneficial to have tools that offer this kind of multi-modal evaluation means. In this paper we describe the rationale, approach, design, and development of visual means that can be used to view and examine non-trivial dynamics of cells and CAs. The concept and the kind of capabilities are developed in DEVS-Suite simulator that go beyond those that are offered in other frameworks and tools that are used in domain such as landscape dynamics [10, 12] and system biology [14].

2 BACKGROUND

The concepts, methods, and tools for Cellular Automata are well established and extensively studied and are commonly used for many application domains including social sciences, engineering, and system biology. Given the scope and the research presented in this paper, we briefly review Cellular Automata [22] and Discrete EVent System Specification (DEVS) [25] concepts and methods followed by highlighting some relevant aspects of Cell-DEVS, MASON, and Mathematica supporting CA modeling, simulation, and visualization.

2.1 Cellular Automata

The Cellular Automata [22] is one of the approaches for modeling and simulating complex system in sciences, engineering, and the arts. It is formed as a grid of cells, each of which has a finite number of states. A model can have simple (e.g., on or off, 0 or 1) to complex (e.g., a vehicle's speed relative to other vehicles, geospatial location, and direction) states. Inside the grid, each cell has a set of other cells that define its neighborhood. The major two common patterns of the cellular neighborhoods are von Neumann and Moore neighborhood structures. The former consists of the four orthogonally adjacent cells, and the latter covers von Neumann neighborhood as well as four other immediate adjacent cells. In a homogeneous structure, the sources of inputs to any cell are its neighbors; similarly every cell can be the input source for all its neighbors. The state of each cell changes in discrete steps (i.e., cell dynamics are defined using a set of functions operating on the cell's own (internal) state and the states of its neighborhood cells).

Cellular Automata model can be advance in discrete time stepped manner where any number of cells may under internal state change. The evolution of CA simulation can lead to homogeneous state, stable or oscillating structures, chaotic manner, and complex patterns with stable localized structures [22].

There exist a variety of CA tools that can model and simulate behaviors of CA models in 1D, 2D, or 3D. For instance, Golly [17] is an open source simulator software for exploring Conway's Game of Life and other cellular automata. JCASim [3] is another general-purpose tool for simulating cellular automata written in Java. It provides display for 1D, 2D (triangular, square, and hexagonal) and 3D grids, different neighborhoods, boundary conditions (periodic, reflective, or constant), and can represent cells using colors, text, or icons. CAME&L [11] named as "Cellular Automata Modelling Environment & Library" is devised to create a universal and extensible library for parallel and distributed computing. It uses Cellular Automata based computation experiment decomposition to complete arbitrary tasks.

2.2 DEVS Approach

An approach for developing Cellular Automata is based on Discrete Event System Specification (DEVS) [25]. This is a formalism where arbitrary complex structures and behaviors of any discrete system can be defined as modular atomic and hierarchical coupled models. Modularity is a key aspect of this formalism - it requires all interactions between model components to be strictly through I/O ports. From model abstraction, this affords significant benefits including encapsulation of a model's state as well as distinguishing state changes to be due to internal or external events. Atomic and coupled models are indistinguishable relative to one another from their I/O. However, this strict modularity leads to high computational cost. Nonetheless, this cost can be significantly reduced at the expense of relaxing I/O modularity.

A cell is a parallel atomic DEVS defined as a mathematical structure $\left\langle X^b, S, Y^b, \delta_{int}, \delta_{ext}, \delta_{conf}, \lambda, ta \right\rangle$ where X and Y are input and output sets, S is the state set, $\delta_{int} : S \rightarrow S$ is the internal transition function, is the external transition function with Q being the set of total states ($Q = \{(s, e), s \in S, 0 \leq e \leq ta(s)\}$), $\delta_{conf} : Q \times X^b \rightarrow S$ subject to $\delta_{conf}(s, \emptyset) = \delta_{int}$ is the confluent transition function for simultaneous external and internal events, $\lambda : S \rightarrow Y^b$ is the output function, and $ta : S \rightarrow R_0^{+\infty}$ is the time advance function.

A coupled model is also defined as the mathematical structure $\left\langle X^b, Y^b, D, \{M_d, d \in D\}, EIC, IC, EOC \right\rangle$. It is essentially a graph where its nodes are atomic and other coupled models and edges are couplings among input and output ports. This specification can be constrained such that it satisfies the requirements of Cellular Automata (e.g., rectangular mesh) which includes pre-defined relationships among neighboring cells. The I/O couplings between cells can be defined to be uniform and the structure to be non-hierarchical with Moore or Von Newman type. It has input and output sets which are X^b and Y^b, an index set D representing names of atomic and coupled models it has, a set of atomic and coupled models M_d, a set of external input couplings EIC for receiving inputs from X^b, a set of external output couplings EOC for sending outputs to Y^b, and a set of internal couplings IC for internal couplings between atomic/coupled models within M_d.

3 RELATED WORK

A vast amount of research has been devoted to cellular automata modeling approaches, simulation techniques and protocols, and visualization tools. In this paper, we restrict our discussion to two classes of modeling approaches and simulation/visualization frameworks. One of these is based on Agent-Based Modeling grounded in software engineering methods and principles. Software components (agents) are assigned to spatial grid models (e.g., 2D and 3D rectangular maps). Another is based on system-theoretic DEVS modeling and simulation methodology where atomic models are contained in coupled models that have strict structures (single or multi-dimensional maps). We choose Mathematica [24], MASON [8], and Cell-DEVS [20] methods and their respective tool sets among numerous frameworks have been developed to support CA modeling, simulation and visualization. These are intended to represent pure Cellular Automata models as well as variants of Agent-Based Modeling methodologies and frameworks.

3.1 Cell-DEVS

This approach is based on Cellular Automata and the Classic DEVS formalism with explicit time delays [20]. Each individual cell is an atomic DEVS model and the CA is a coupled model. In this approach, timing delays are defined based on time advance function ($ta(s)$). A cell is formalized as $TDC = \langle X, Y, S, N, type, d, \tau, \delta_{int}, \delta_{ext}, \lambda, D \rangle$ where X and Y are the set of external inputs and the set of external outputs with S representing state set. A cell uses a set of input

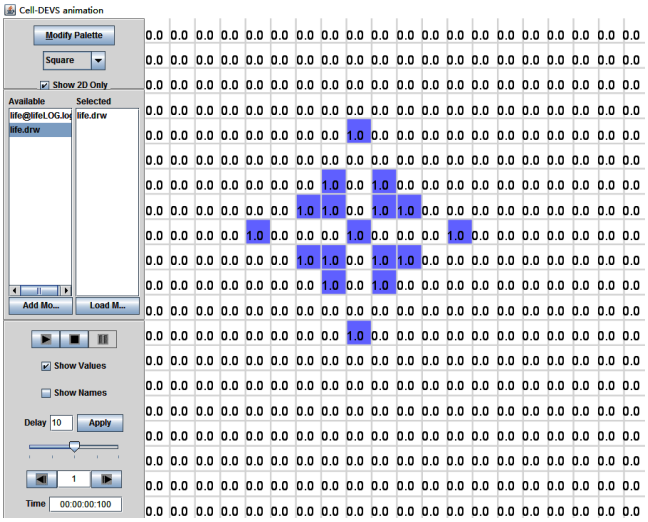

Figure 1: Cell-DEVS Animation Display [19]

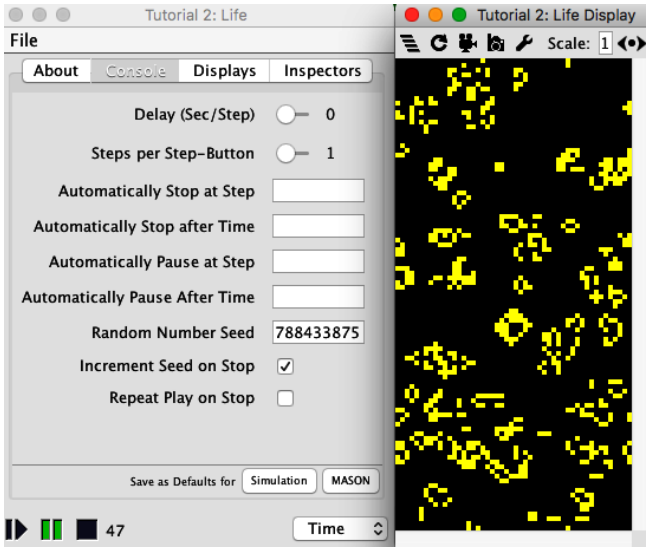

Figure 2: MASON CA Game of Life Simulation [8]

values N to calculate its future state with the local computing function τ. A delay is of *inertial* and *transport type* with duration d is assigned to each cell. Other types of delay may be added. D is the time advanced function $ta(s)$. The remaining elements are the same as those defined for the atomic DEVS model.

Akin to DEVS coupled model, Cell-DEVS coupled model is specified as $GCC = \langle X, Y, I, X_{list}, Y_{list}, n, \{t_1, \cdots, t_n\}, N, C, B, Z\rangle$ where X and Y are the set of external input/output events; X_{list} and Y_{list} are the input/output coupling lists; I is the interface of the model; n is the dimension size of the cell space; $\{t_1, \cdots, t_n\}$ defines the number of cells in each dimension; N represents the neighborhood set; C is the cell space; B is the set of border cells and Z is the translation function which defines the coupling of cells. The Cell-DEVS approach is implemented in CD++ [18]. The DEVS Atomic models can be programmed in C++, and CDL (cell description language) was used to define Cell-DEVS models. The cell space size, influence, neighborhood and borders are defined in this specification language. In addition, the local cell behavior is defined by a set of rules. The CD++ can be used to simulate and visualize results in a 2D space as shown in Figure 1 [6]. Results can be stored in a log file and then spatial CA data animated. Data can also be plotted as basic linear time trajectories [18].

Although Cell-DEVS and CA-DEVS share a common basis (DEVS formalism), they have some important differences in terms of their modeling, simulation, and visualization concepts. In general, any system that can be modeled in one can also be modeled in the other; however, CA-DEVS supports parallel DEVS. For example, the delay types can be modeled in CA-DEVS without any restrictions (e.g., the inertial and transport delays are straightforward to specify using existing abstractions in Parallel DEVS and the DEVS-Suite simulator [1, 7, 16]). From framework perspective, they have different architectures, designs, and visualizations. The CA-DEVS uses Model-Façade-View-Control architecture. It uses run-time data for spatial representations. Inputs, outputs, and states of any number of cell components can also be selected and plotted at initialization and/or during simulation execution. A unique aspect is an ability to model and simulate cells as parallel DEVS models as well as plotting their time trajectories in superdense time in addition to linear time. These visualization capabilities enable "model debugging" which is useful for developing and evaluating cellular automata that have complex dynamics.

3.2 MASON

An Agent-Based Modeling framework is MASON (Multi-Agent Simulator Of Neighborhoods) [8]. It supports developing and visualizing Cellular Automata models. As a discrete-event simulator, model dynamics are governed using events instead of a counter. The architecture of MASON can be viewed to consist of two layers. One is a model layer and another is a visualization layer. In the model layer, agents are defined by user to perform some activities with step function. After that, the agents are assigned a schedule which will call the step function once or repeatedly. The agents running according to the schedule are shuffled. Meanwhile, the fields should be defined for all the agents as the space and location for the simulation. The user can also register the model to the visualization layer for animation. The visualization layer is written in Java AWT/Swing. The visualization offers simple features, like step, run, stop, pause functions as shown in Figure 2. An important distinction between MASON and CA-DEVS (and Cell-DEVS) is the concept of component. In the former strict modularity (i.e., interactions between cells are not subject to strict input/output coupling) is not supported. Consequently, the speed of DEVS-based CA simulations, unlike MASON, exponentially decreases as the number of I/O ports and couplings linearly increase.

3.3 Mathematica

Mathematica is a powerful computational modeling tool supporting Cellular Automata [23, 24]. Cellular Automata models have finite number of dimensions and the cells have a finite number of states

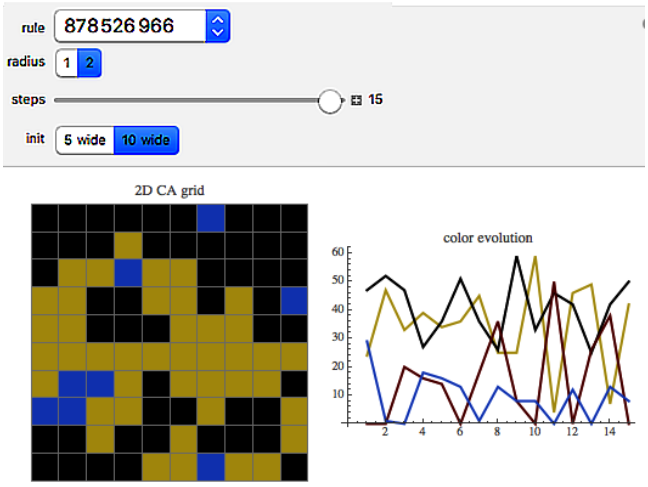

Figure 3: Mathematica CA Four Color Outer Median Model [13]

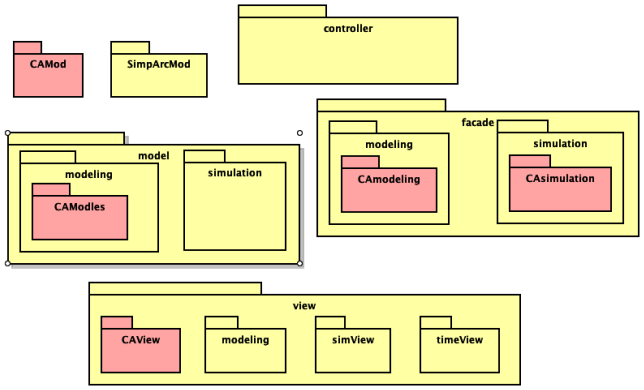

Figure 4: UML Package Diagram for CA DEVS-Suite Simulator

that generally behave in discrete steps. Users can define CA rules, for example as mathematical functions, or use any of many existing ones. Visualization features to load, control, execute, and observe the dynamics of model evolution at run-time are supported as shown in Figure 3. Mathematica can generate basic linear time trajectories at run-time.

Cellular Automata models in Mathematica and CA-DEVS offer similar, but also different, capabilities. In Mathematica, dynamics are specified as rules without the concept of cells being components and be composable. A CA model can be defined in terms of mathematical equations representing uniform structures such as multi-dimensional arrays and matrices. Numerous problems (e.g., traffic engineering [5]) can be modeled and efficiently simulated. Unlike Mathematica, CA-DEVS is based on system theory where CA is a collection of parallel atomic component models that are connected to each other in specific spatial structures using input and output ports and couplings. Mathematic and CA-DEVS also differ in supporting time-series tracking of individual cell. CA model in Mathematica are not as expressive as in CA-DEVS, in part, because component modeling subsumes equation modeling. Furthermore, basic linear time series (time trajectories) in Mathematica can be generated using scripts (see Figure 3). However, the same cannot be said about general-purpose time trajectories for cells that have complex behaviors. In particular, both linear and superdense time trajectories (time series) are supported in DEVS-Suite as such complex behaviors of cells and CAs. For example, plots of receiving and sending events at the same time can be key for debugging non-trivial dynamics within and between neighboring cells.

4 CA-DEVS SIMULATOR FRAMEWORK

DEVS-Suite simulator supports model development, execution, animation, and tracking for component-based parallel DEVS models [1, 7]. Although it can be used to define CA-DEVS models having common structures (i.e., von Numen and Moore), it does not provide built-in modeling constructs. The tracking environment consists of run-time hierarchal component animation (referred to as simView), linear and superdense time trajectory plotting (referred to as TimeView), and data storage for post processing [16]. However, component viewing (e.g., animation of input/output message passing and display of states) are not suitable for CA models. We have conceptualized a new animation for CA-DEVS simulation. This is developed by extending both the modeling and tracking modules of the DEVS-Suite. These new packages are highlighted in Figure 4. It supports run-time 2D animation of DEVS-based Cellular Automata model and independent TimeView tracking for any number of selected cells. It also offers feature of playback for 2D animations.

4.1 Architecture

The architecture of the DEVS-Suite is Model-Façade-View-Control (MFVC) [15]. To shield the complexity of model (i.e., modeling and simulation engines), the architecture uses Façade. The Controller and View modules access the data in the Façade layer instead of Model. This is fundamental for hiding the intertwined relationships between modeling and simulation engines. The Façade supports strong separation between simulation data to be displayed in different views. Based on this architecture, we have developed a CA View (see Figure 4). The relationships between packages are not shown for simplicity [15]. In this new environment, both Component DEVS and CA-DEVS models are supported. Specifically, one can use this new DEVS-Suite simulator to model and simulate these two distinct classes of Parallel DEVS models where each is supported with its appropriate animation component view (simView) and CA view CAView complemented with its separate TimeView.

The simulator can be launched in Component (Non-CA) and CA-DEVS modes. In Component mode, one can choose simView and/or Tracking. In CA mode, one can also chose to have CAView and/or Tracking. In both Component-DEVS and CA-DEVS, simulation data can be customized, stored, and viewed in the simulator's console. Although for managing Component-DEVS and CA-DEVS models, separate packages (e.g., *SimpArcMod* and *CAMod*) can be declared. Any Cellular Automata DEVS model consists of cell and coupled models. In CA-DEVS, we define the cells as parallel atomic models. The spatial aspect of these cells is defined as a parallel coupled DEVS

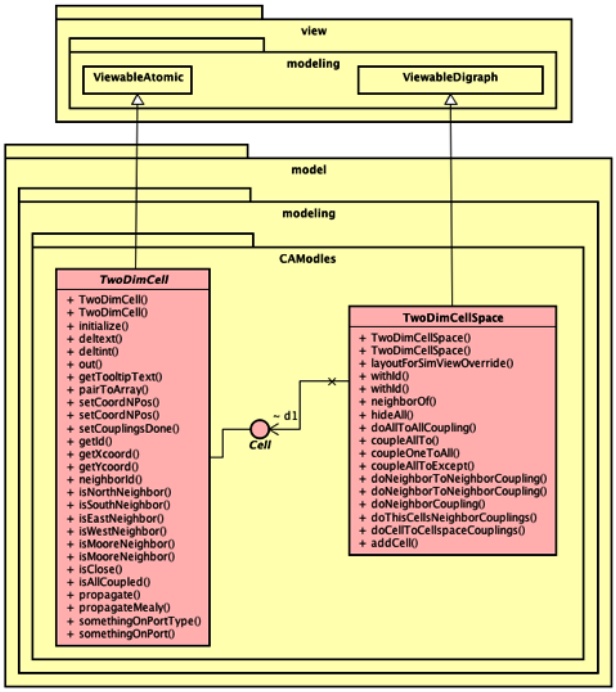

Figure 5: UML Class Diagram for CA-DEVS Model Module

mode. Each cell has an assigned location index and coupled to its immediate neighbors. For instance, the cell (1, 1) is coupled with cell (0, 0), cell (0, 1), cell (0, 2), cell (1, 0), cell (1, 2), cell (2, 0), cell (2, 1), and cell (2, 2) following the Moore neighborhood pattern. This coupling is defined in the template model *TwoDimCellSpace* class, and the individual cell atomic model is specified in *TwoDimCell* class. These classes are derived from predefined *ViewableAtomic* and *ViewableDigraph* classes, which are the base classes for Component-DEVS. These two CA template classes are included in *CAModels* package (see Figure 5).

The CASimulation package is defined inside the *Simulation* package of model module of the DEVS-Suite because the CA models are the same as other DEVS coupled models with additional dimension information. The same simulation protocol is used for both component and CA models.

For the CA-DEVS, two new Façade classes (i.e., *FCACellModel* for additional location index and *FCASpaceModel* for storing extra spatial size and dimension data) are defined. The *FCACellModel* and *FCASpaceModel* classes inherit from the *FAtomicModel* and *FCoupledModel* classes, respectively (see Figure 6). The *FCASimulator* class is extended from the *FCoupledSimulator* class. The *FCASimulator* and *FCoupledSimulator* together are responsible for simulating CA models.

The CAView (which animates state changes for all cells) is developed using JavaFX which is a successor to the Swing/AWT. The *CellView* class is extended from JavaFX StackPane and the *SpaceView* is the Scene of JavaFX. The new CA-DEVS UI runs the animation through data received from the Façade layer and controlled by the main View layer (see Figure 7).

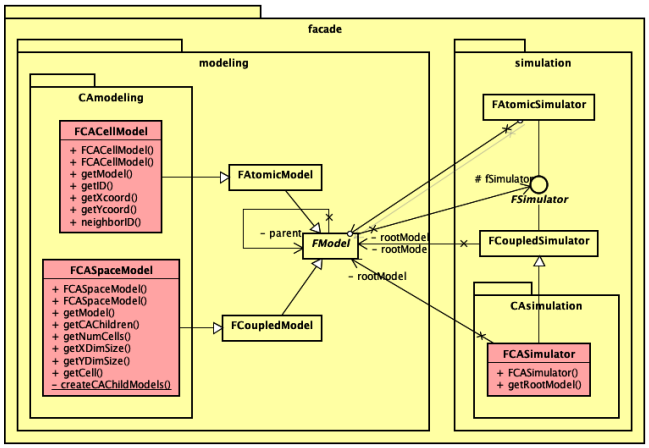

Figure 6: UML Class Diagram for CA-DEVS Façade Module

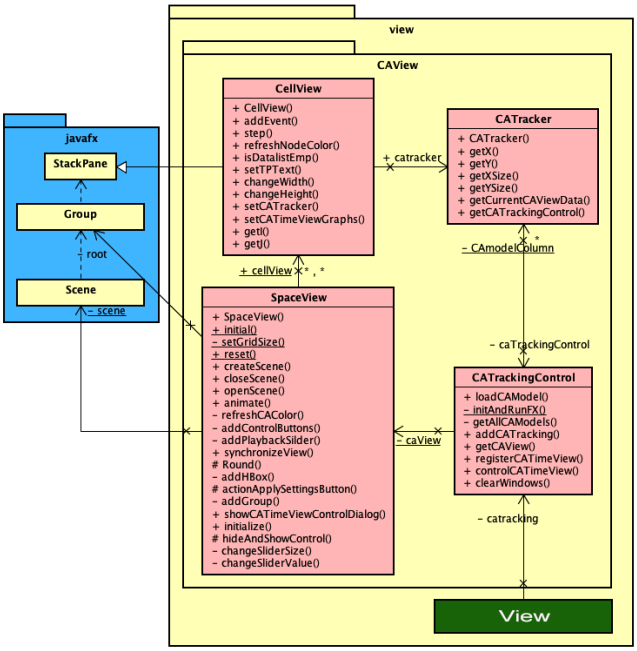

Figure 7: UML Class Diagram for CA-DEVS View Module

The CA tracking environment is registered to the View layer. Every time there is a control command to step simulation forward, the data generated by the Model will be passed through the Façade layer to the CA *SpaceView* and *CellView*. Some of the objects and interactions are shown in Figure 8. The animation in the CAView will play completely without interrupting the simulation, because the JavaFX thread is separated from the Java main thread. Animation runs more efficiently than the simulation as the scale of CA model increases. This means delay in CAView is due to simulation handling large number of input/output messages.

Given the DEVS-Suite simulator's façade, it is convenient to add this CAView to it. The *CATrackingControl* registers the *CATracker* to all the cells in the CA space. The *CATracker* and *CATrackingControl*

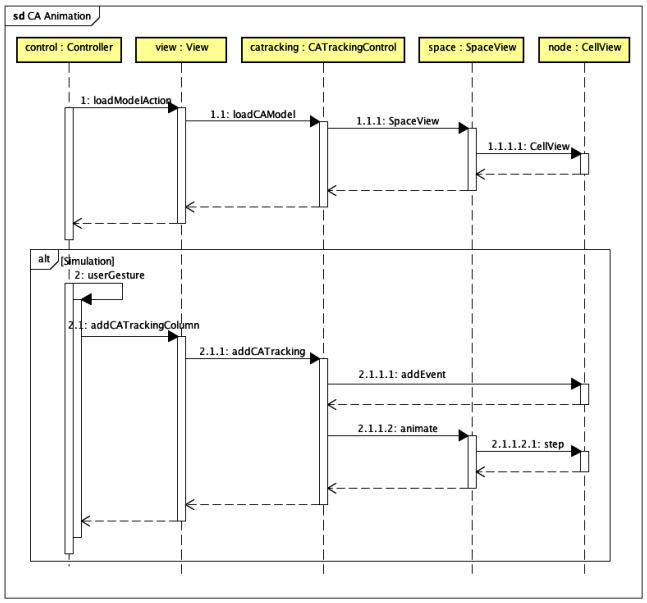

Figure 8: UML Sequence Diagram for CAView

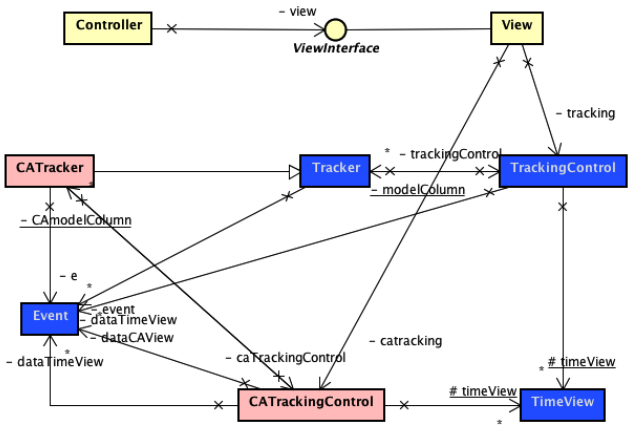

Figure 9: UML Class Diagram for CAView and TimeView

classes are derived from the Component-DEVS TimeView's *Tracker* and *TrackingControl* classes. Thus, the CAView component can use the same mechanism to control both the *TrackingControl* and the *CATrackingControl* for CA TimeView and CAView as shown in Figures 9 and 10. The TimeView is used for generating at run-time plots from time-series data sets collected through tracking state variables, input/output port and value pairs, and simulator's last and next event times [16].

4.2 Animation Playback

Since CA-DEVS models can have large dimensions, the data generated for all the steps of the animation can be very large. Considering a lot of resources has been used to do the modeling and simulation computation, we use a separate thread to record the history of the

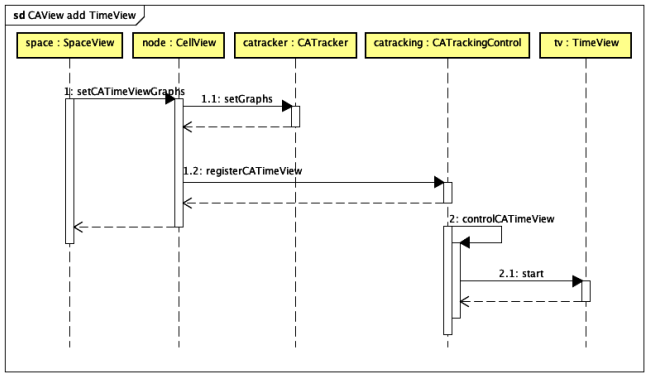

Figure 10: UML Sequence Diagram for CAView and TimeView

animation. Running CA View as a separate thread without interruption speeds up both the simulation execution and displaying CA animation. The animation data is now stored in memory thus it can be played back efficiently.

5 CA-DEVS USER INTERFACE

The user interface for CA-DEVS called CAView has Tracking, Animation, Play, and Simulator Control parts (see Figures 11 and 12). It also has 2D spatial and time trajectory visualization parts. The 2D representation of Cellular Automata can be visualized simultaneously with any number of time-based trajectories.

The *Tracking Control* panel can be used to select and track independently each cell's state, input, and output. This can be done at the beginning of the simulation or any time the simulation is paused. Certain states of all cells in CA are pre-defined for tracking. Any other state can be tracked by making it as an output variable assigned to one or more output ports.

The *Animation Control* panel allows the modeler to turn on or off CA spatial viewing and set up the animation speed which determines how frequently the view is to be updated. Choosing a large value results in showing fewer simulation steps (e.g., if speed is set to 10, then the updates to the states of the cells are updated every 10 steps). This way displaying the CA spatial view can be set to be coarse grain relative to the simulation step time granularity. Setting of the animation steps can help the simulation to execute faster.

The *Playback Control* is helpful for rewinding spatial viewing. The maximum length indicates the size of the buffer for playback storage and the tracking interval decides the update frequency of the playback frames. Both settings can limit the amount of data that is dynamically stored and thus can be replayed. It should be note that Playback is for shows 2D view; it is not intended for simulation rollback.

The *Simulator Control* is the same as the one for Component-DEVS. It provides choices for running the simulation in single-, multi-step, and continuous modes. This control panel allows increasing or decreasing simulation execution speed by choosing the real-time factor that sets the simulators' clock to be running faster or slower than the JVM clock.

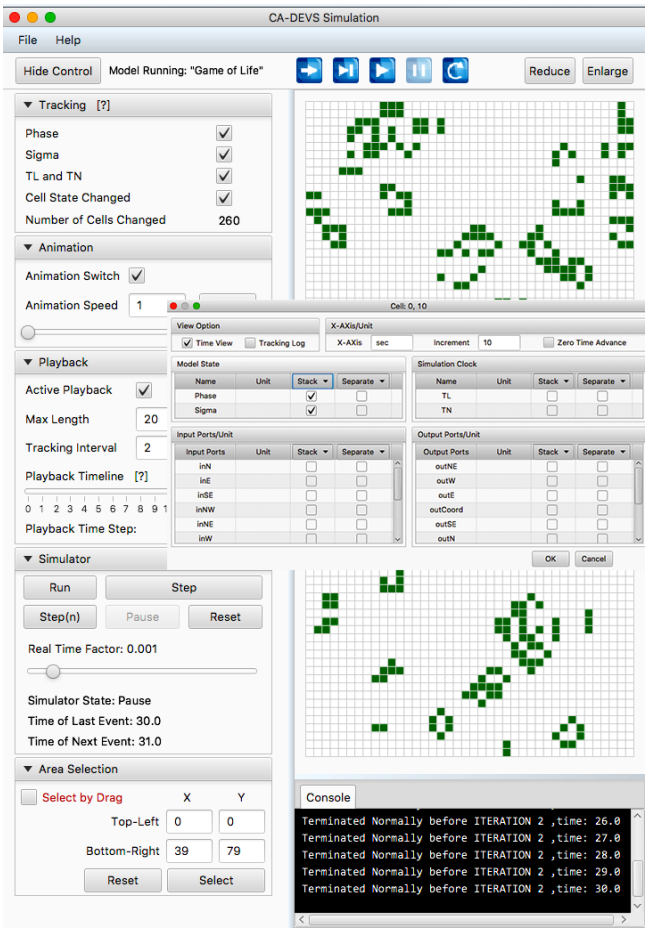

Figure 11: CA-DEVS Control Panel with CA Animation

The TimeView component is integrated into the CA-View component. By selecting the individual cell in the 2D visualization space, the user will be able to track any single cell's behavior at the run time alongside CA animation. To our knowledge, this is not supported in any other CA simulator. As shown in Figure 11, time to next event (*tN*), phase, and sigma variables for the cell located position (7,6) can be plotted. We note that TimeView as shown has a liner time base; the dotted line represent sigma set to infinity (see Section 2.2). This is because the cells in the Game of Life model do not have any dynamics where multiple events simulation steps occur in zero time. Details of superdense time trajectory with example trajectories for state, input, and output variables having multiple values indexed for a given time can be found [16]. Similarly, if cells have different time steps, then *tN* trajectory can reveal complex timing information for changes in the state of the cell as well as receiving input events and sending output events.

The CA viewing supports assigning colors to states (see Figure 11). Colors for cells can be changed during the simulation. When the color is modified in the selected cell, all the other cells having the same state also change their colors to the same color. Instead of coloring, icons may also be added, for example, to capture application domain knowledge.

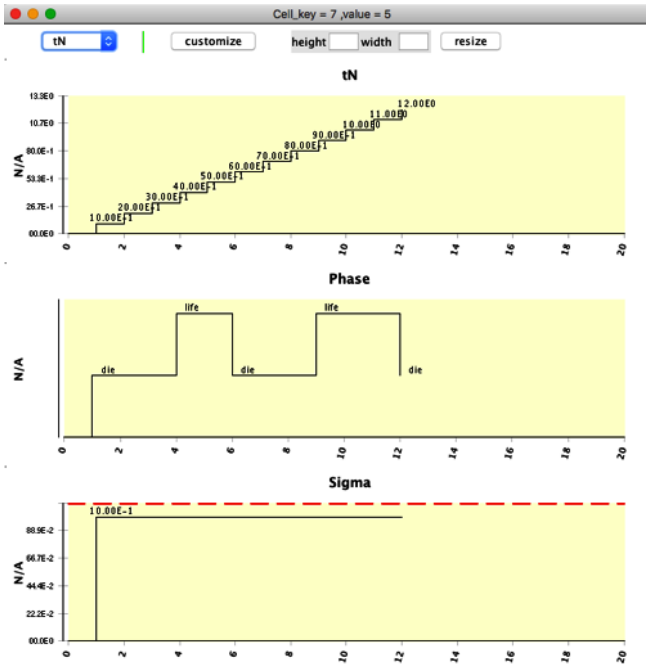

Figure 12: TimeView Tracking for Cells

6 EVALUATION

The cells in CA-DEVS communicate with each other via message passing through input/output couplings. Not only the cell, but also CA models have significant differences. To help gaining a better understanding of the CA-DEVS, we use the classic Game of Life model and compare it with Cell-DEVS, MASON, and Mathematica. This choice serves the key purpose of focusing on modeling and visualization. It can also serve to demonstrate component-based modeling with message passing although important for development and debugging models, their simulations are computationally inefficient unless they can be executed using multiprocessor computing platforms.

In Game of Life (GoL), each cell has two states: "Alive" or "Dead". The life of each cell is determined by its neighbors using rules such as (a) if a cell has less than two alive neighbors, then the cell dies, (b) if there are more than five alive neighbors, then the cell dies, and (c) if there are exactly three alive neighbors, the cell becomes alive. The Game of Life Model [4] serves as suitable model for evaluating some aspects of CA DEVS-Suite simulator.

The GoL model can be modeled in Mathematica by defining (1) the rules for how cells change their status, (2) the spatial size of the area cells occupy initialize each cell, (3) execution and visualization setting as shown in Figure 13.

We developed the Game of Life model having a Moore structure in the CA DEVS-Suite simulator. Each atomic cell's status is sent as messages via input/output couplings to all of its neighbors. When

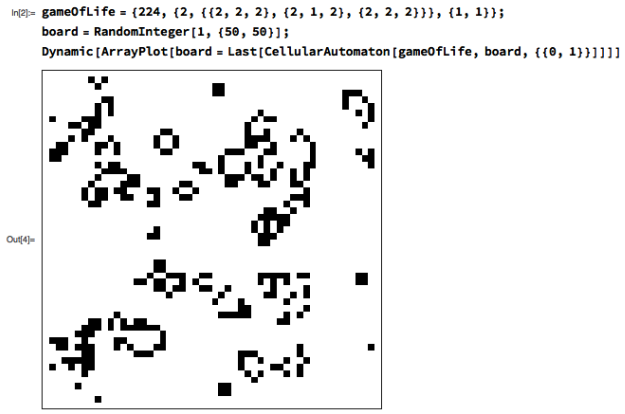

Figure 13: Program and Animation View for GoL in Mathematica [21]

a cell receives messages from its neighbors, it will run its external transition function to compute its future state (i.e., Alive or Dead). All the messages are delivered and processed at the same time (concurrently) via external transition function δ_{ext}. In this simple model, all cells have the same timing (i.e., the time advance function $ta(s)$ in every cell computes the same σ to the next event. The internal event function δ_{int} of every cell is executed. The σ for δ_{int} is set to infinity which causes the cell to wait indefinitely until it receives input events from any of its neighbors. Outputs for neighboring cells are generated at the end of σ time allowed for a cell to be either alive or dead. Outputs from every cell in the CA are sent to its neighboring cells just before the execution of the internal transition function. In general, the execution order for the external, internal, and output functions for every cell is determined by its own state and well as the inputs it may receive from its neighbors. The Parallel DEVS simulation protocols provided for both atomic and coupled models are responsible for ordering executions in both sequential and parallel cases. Therefore, it is not necessary for all cells to execute their functions in the same order.

Now considering computational aspect of CA-DEVS, it is easy to see as the size of the GoL model increases, the computation resource it uses will also rise. Figure 14 demonstrates the computing times used for running this model having 5×5, 20×20 and 100×100 dimensions. Each of these simulations is repeated 10 times, each for 20 steps. As expected, computation times for both CAView and TimeView are higher. If only simulation is processed without 2D animation and time trajectory plots, less time is needed. The result for the combined animation and time trajectory views shown in Figure 14 is for two Cells, each displaying 3 plots. The 100×100 experiment shows most of the computation time is used for running the simulation, not animating the CA. Of course, including (linear or superdense) time trajectories also consumes computing resources. For trajectory added the total computation time increases (i.e., visualization efficiency depends on the number of cells being tracked and the number of input/output ports and state variables each cell has). We note that it is not necessary to track selected cells in the same way (i.e., for one cell its state can be tracked while for another cell its input/output values can be tracked). All the

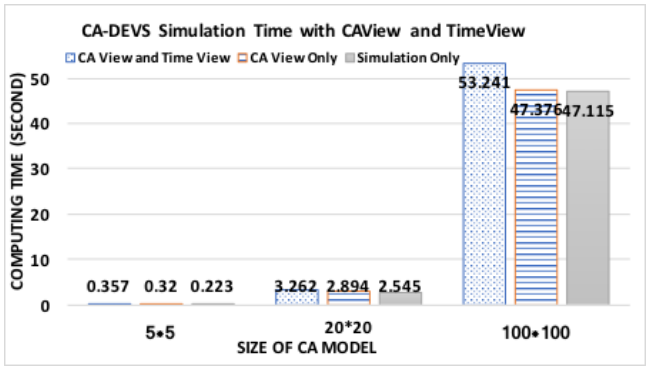

Figure 14: CA-DEVS Computational Efficiency for GoL

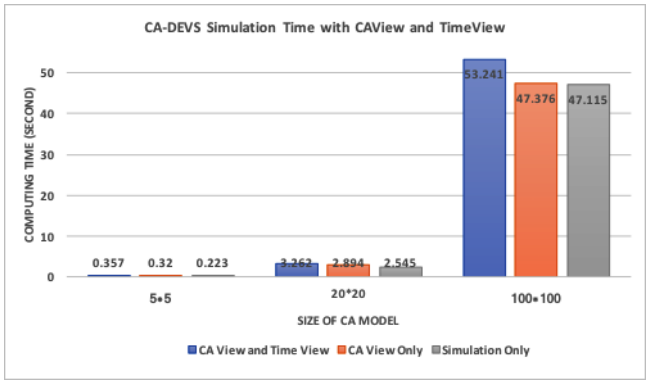

Figure 15: Computational Efficiencies for Simulating and visualizing GoL in CA-DEVS, MASON, and Mathematica

experiments in this paper are executed in a Mac Mini with 16GB Memory, 2.5 GHz Intel Core i5 processor, and in OS X 10.12 using DEVS-Suite 4.0.0 supported by JRE 1.8 and 1.9 [1].

7 CA SIMULATION IN DEVS-SUITE, MASON AND MATHEMATICA

Since the DEVS-based CA model requires data exchanges via message communication, its efficiency can rapidly degrade as the size of the model increase. In contrast, global data sharing does not. To demonstrate this, a comparison between MASON, Mathematica and CA-DEVS demonstrate these three CAs having 20×20 and 100×100 dimensions are developed for the Game of Life example. CA-DEVS, MASON and Mathematica all have identical rules with randomized initial condition that each cell can be 0 or 1 with 50 chance. The simulation progress for these models are the same for each of the lower and higher dimension CAs and each of the simulation is running for 20 steps. However, as shown in Figure 15, the actual execution time for CA-DEVS is significantly higher than Mathematica and MASON for the 100×100 CA. In terms of debugging models using animation, time trajectories, and playback, message-based cell-to-cell interactions are preferable when compared with data sharing used in Mathematica and MASON.

8 CONCLUSION AND FUTURE WORK

We can evaluate the run-time execution of CA-DEVS without any animation and compare it with that of Cell-DEVS. However, the purpose of CA-DEVS as compared with Cell-DEVS and other tools such as MASON and Mathematica is to provide a suite of capabilities that empower model development, particularly when the cells have rich behaviors with arbitrary timings and consequently interacting in non-trivial ways. While debugging code, basic run-time visualization as in Mathematica and MASON, or post animation as in Cell-DEVS are important, the CA-DEVS offers the new and useful capabilities.

In the DEVS-Suite framework, modelers can develop either component - based hierarchical models or Cellular Automata models. Modelers can both "debug the model at run-time" as well as post processing stored data. During simulation execution, modelers have the flexibility to examine changes in the state of any number of cells at different levels of granularity including choosing to have linear or superdense time trajectories. They can track the content of I/O messages exchanges between neighboring cells as well as examining customizable Cellular Automata animations. Animation can also be customized for playback. These capabilities do not require writing scripts or users developing code and integrating it to the simulator.

The research described in this paper can be furthered in different directions. To support computationally efficient simulations for largescale CAs, message passing among cells and CAs can be replaced with some efficient input/output communication mechanism. Considering that Component and Cellular Automate DEVS models serve distinct purposes, composing them can be useful and even necessary. In some application domains, multiple CAs (e.g., [2, 10]) also need to be composed. Therefore, composition of multiple Cellular Automata models as introduced in Composable Cellular Automata [9] is attractive approach to pursue. Achieving these kinds of model composability can lead to a new generation of hybrid modeling frameworks and tools where loosely coupled modeling, simulation and visualization modules is useful for building simulations for systems of systems such as Internet of Things and Multi-Cellular human biology.

ACKNOWLEDGMENTS

Support for Hessam S. Sarjoughian is partially provided with NSF grants # CNS-1639227 and # DEB-1313727.

REFERENCES

[1] ACIMS. 2017. DEVS-Suite Simulator 4.0.0. (2017). https://acims.asu.edu/software/devs-suite/.
[2] C Michael Barton, Isaac IT Ullah, Sean M Bergin, Hessam S Sarjoughian, Gary R Mayer, Joan E Bernabeu-Auban, Arjun M Heimsath, Miguel F Acevedo, Julien G Riel-Salvatore, and J Ramón Arrowsmith. 2016. Experimental socioecology: Integrative science for anthropocene landscape dynamics. *Anthropocene* 13 (2016), 34–45.
[3] Uwe Freiwald and Jörg R Weimar. 2001. JCASim - a Java system for simulating cellular automata. In *Theory and Practical Issues on Cellular Automata*. Springer, 47–54.
[4] Martin Gardner. 1970. Mathematical games-The fantastic combinations of John Conway's new solitaire game of Life. *Scientific American, October* (1970), 120–123.
[5] Richard J Gaylord and Kazume Nishidate. 2013. *Modeling nature: Cellular automata simulations with Mathematica®*. Springer.
[6] Kiril Kidisyuk and Gabriel A Wainer. 2008. CD++ Modeler: a graphical toolkit to develop DEVS models. In *Proceedings of the 2008 Spring simulation multiconference*. Society for Computer Simulation International, 8.
[7] Sungung Kim, Hessam S Sarjoughian, and Vignesh Elamvazhuthi. 2009. DEVS-suite: a simulator supporting visual experimentation design and behavior monitoring. In *Proceedings of the 2009 Spring Simulation Multiconference*. Society for Computer Simulation International, 161.
[8] Sean Luke, Claudio Cioffi-Revilla, Liviu Panait, Keith Sullivan, and Gabriel Balan. 2005. Mason: A multiagent simulation environment. *Simulation* 81, 7 (2005), 517–527.
[9] Gary R Mayer and Hessam S Sarjoughian. 2009. Composable cellular automata. *Simulation* 85, 11-12 (2009), 735–749.
[10] Gary R Mayer and Hessam S Sarjoughian. 2016. Building a hybrid DEVS and GRASS model using a composable cellular automaton. *International Journal of Modeling, Simulation, and Scientific Computing* 7, 01 (2016), 1541005.
[11] Lev Naumov. 2004. CAME & L–Cellular Automata Modeling Environment & Library. In *International Conference on Cellular Automata*. Springer, 735–744.
[12] Lewis Ntaimo, Xiaolin Hu, and Yi Sun. 2008. DEVS-FIRE: Towards an integrated simulation environment for surface wildfire spread and containment. *Simulation* 84, 4 (2008), 137–155.
[13] Abigail Nussey. 2017. Four-Color Outer Median Cellular Automata on the Grid from the Wolfram Demonstrations Project. (2017). http://demonstrations.wolfram.com/FourColorOuterMedianCellularAutomataOnTheGrid/
[14] Brenden K Petersen, Glen EP Ropella, and C Anthony Hunt. 2014. Toward modular biological models: defining analog modules based on referent physiological mechanisms. *BMC systems biology* 8, 1 (2014), 95.
[15] Hessam S Sarjoughian and R Singh. 2004. Building simulation modeling environments using systems theory and software architecture principles. In *Proceedings of the Advanced Simulation Technology Conference*. 99–104.
[16] Hessam S Sarjoughian and Savitha Sundaramoorthi. 2015. Superdense time trajectories for DEVS simulation models. In *Proceedings of the Symposium on Theory of Modeling & Simulation: DEVS Integrative M&S Symposium*. Society for Computer Simulation International, 249–256.
[17] A Trevorrow, T Rokicki, et al. 2009. Golly: open source, cross-platform application for exploring Conways Game of Life and other cellular automata. *URL http://golly.sourceforge.net* (2009).
[18] Gabriel Wainer. 2002. CD++: a toolkit to develop DEVS models. *Software: Practice and Experience* 32, 13 (2002), 1261–1306.
[19] Gabriel A Wainer. 2009. *Discrete-event modeling and simulation: a practitioner's approach*. CRC press.
[20] Gabriel A Wainer and Norbert Giambiasi. 2001. Application of the Cell-DEVS paradigm for cell spaces modelling and simulation. *Simulation* 76, 1 (2001), 22–39.
[21] Eric W. Weisstein. 2017. Game of Life. From MathWorld–A Wolfram Web Resource. (2017). http://mathworld.wolfram.com/GameofLife.html
[22] Stephen Wolfram. 1984. Cellular automata as models of complexity. *Nature* 311, 5985 (1984), 419.
[23] Stephen Wolfram. 1996. *The Mathematica Book, Wolfram Median*. Cambridge University Press.
[24] Stephen Wolfram. 2002. *A new kind of science*. Vol. 5. Wolfram media Champaign.
[25] Bernard P Zeigler, Herbert Praehofer, and Tag Gon Kim. 2000. *Theory of modeling and simulation: integrating discrete event and continuous complex dynamic systems*. Academic press.