Design of Persian Tapestry in CD++

Mohammad Moallemi, Gabriel Wainer
Dept. of Systems and Computer Engineering
Carleton University Centre of Visualization and Simulation (V-Sim)
1125 Colonel By Dr. Ottawa, ON, Canada.
moallemi@sce.carleton.ca, gwainer@sce.carleton.ca

Keywords: Cellular Automata, DEVS, Persian Tapestry

Abstract

DEVS is a formal modeling and simulation (M&S) framework based on generic dynamic system concepts. Cell-DEVS is a formalism for cell-divided models based on DEVS. Cellular automata can be used to model tapestries by implementing them as a 3-D cell space. In this work we simulate some models of Persian tapestry by Cell-DEVS formalism, considering each cell as a knot in tapestry. There are three planes (tapestry plane, front plane and rear plane) each containing n by n cells. The final shape of tapestry is represented in the tapestry plane. We simulate variety of design patterns with different sizes of tapestries to illustrate simple to complex designs. The simulator used in this work is CD++, an M&S toolkit that implements DEVS and Cell-DEVS theories. The results of the simulation are presented in 2D environment with CD++ modeler.

1. INTRODUCTION

Today simulation is becoming very important because of its useful application in design of complex systems. Modeling and simulation (M&S) methodologies have become crucial for implementing, designing, and analyzing a broad verity of systems. Among the existing simulation techniques, DEVS (Discrete Event System Specification) formalism[1] provides a discrete-event M&S approach which allows construction of hierarchical models in a modular manner. A cellular automaton is a discrete model which is composed of a network of cells that each cell has a finite number of states[2]. Time is also discrete. The state of each of the cells in time t is a function of states of its predefined neighbor cells in time t-1. Cellular automata has been used to simulate design of Persian tapestry[3]. The long term goal of this simulation is to design different models of tapestry by computer and use it in the business to make real tapestries. In this simulation each cell in the cellular automata can be considered as a knot in the real tapestry.

DEVS is an increasingly accepted framework for understanding and supporting the activities of modeling and simulation. DEVS is a sound formal framework based on generic dynamic systems, including well-defined coupling of components, hierarchical, modular construction, support for discrete event approximation of continuous systems and support for repository reuse. DEVS theory provides a rigorous methodology for representing models, and it does present an abstract way of thinking about the world with independence of the simulation mechanisms, underlying hardware and middleware. A real system modeled with DEVS is described as a composite of submodels, each of them being behavioral (atomic) or structural (coupled).

Cell-DEVS[4] has extended the DEVS formalism, allowing the implementation of cellular models with timing delays. A Cell-DEVS model is a lattice of cells holding state variables and a computing apparatus, which is in charge of update the cell state according to a local rule. This is done using the present cell state and those of a finite set of nearby cells (called its neighborhood). Cell-DEVS improves execution performance of cellular models by using a discrete-event approach. It also enhances the cell's timing definition by making it more expressive. Each cell is defined as a DEVS atomic model, and it can be later integrated to a coupled model representing the cell space. Cell-DEVS atomic models are informally defined as in Figure 1.

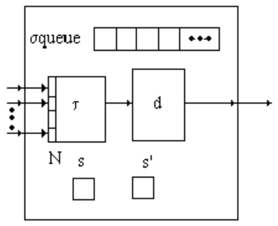


Figure 1- Description of a Cell-DEVS atomic model.

Each cell uses N inputs to compute its next state. These inputs, which are received through the model's interface, activate a local computing function (τ). A delay (d) can be associated with each cell. The state (s) changes can be transmitted to other models, but only after the consumption of this delay. Once the cell behavior is defined, a coupled Cell-DEVS can be created by putting together a number of cells interconnected by a neighborhood relationship. A Cell-DEVS coupled model is informally presented in Figure 2.

A coupled Cell-DEVS is composed of an array of atomic cells, with given size and dimensions. Each cell is connected to its neighborhood through standard DEVS input/output ports. Border cells have a different behavior due to their particular locations, which result in a nonuniform neighborhood. Finally, the model's couplings can be defined.

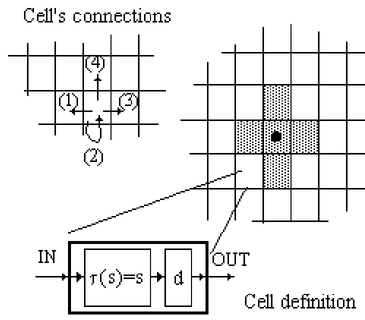


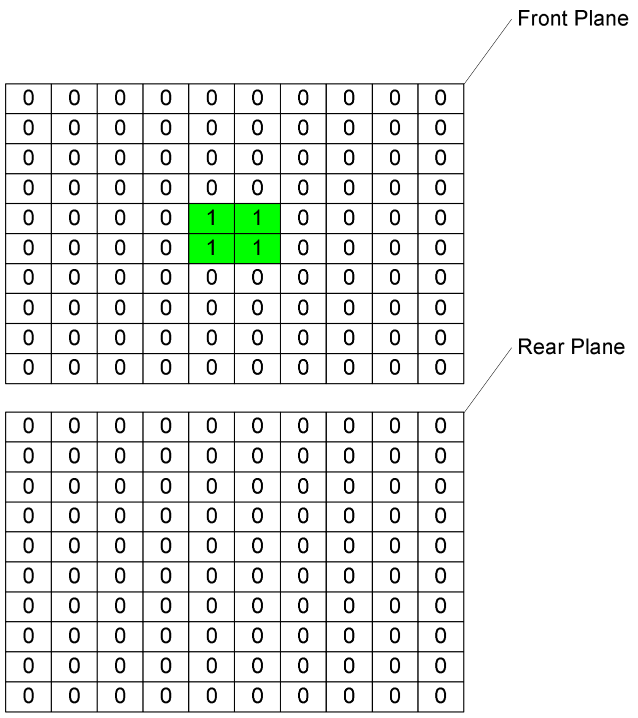
Figure 2- Description of a Cell-DEVS coupled model.

CD++ [5], [6] is a modeling tool that was defined using the DEVS and Cell-DEVS specifications. The toolkit includes facilities to build DEVS and Cell-DEVS models. DEVS Atomic models can be programmed and incorporated onto a class hierarchy programmed in C++. Coupled models can be defined using a built-in specification language. Cell-DEVS models are built following the formal specifications for DEVS models (informally presented in the previous section), and a built-in language is provided to describe them. CD++ makes use of the independence between modeling and simulation provided by DEVS, and different simulation engines have been defined for the platform.

CD++ is built as a class hierarchy of models related with simulation processing entities. DEVS Atomic models can be programmed and incorporated onto the Model basic class hierarchy using C++. Once an atomic model is defined, it can be combined with others into a multicomponent model using a specification language specially defined with this purpose. CD++ also includes an interpreter for Cell-DEVS models. The language is based on the formal specifications of Cell-DEVS. The model specification includes the definition of the size and dimension of the cell space, the shape of the neighborhood and borders. The cell's local computing function is defined using a set of rules with the form POSTCONDITION DELAY {PRECONDITION}. These indicate that when the PRECONDITION is satisfied, the state of the cell will change to the designated POSTCONDITION, whose computed value will be transmitted to other components after consuming the DELAY. If the precondition is false, the next rule in the list is evaluated until a rule is satisfied or there are no more rules.

2. CONCEPTUAL MODEL

Cellular automata can be used to model tapestries by implementing them as a 3-D cell space. The tapestry is represented by three planes (tapestry plane, front plane and rear plane) each containing n by n cells. Initially all cells of front plane are set to zero except for m central cells, where m<n. Figure 3. Tapestry architecture- initial state represents the initial scenario of both front and rear planes.



0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	1	0	0	0	0
0	0	0	0	1	1	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

Figure 3- Tapestry architecture- initial state

Each cell in front plane has four neighbors: N, E, S, and W (Von Neumann neighbors) and also two corresponding cells from rear and tapestry plane. After initialization phase, each cell in front plane counts the number of cells in its neighborhood whose value is 1. If the cell is surrounded by even number of alive (one) cells, the cell is set to 1, otherwise it is set to zero. On the other hand, the second plane (rear plane) works as a mentor and updates its cells accordingly by watching the state changes of the cells in front plane. Thus, rear plane takes on the old values form front plane right before changes occur to front plane's cells. Meanwhile the third plane (tapestry plane) which represents the design of tapestry is made by getting four values 0, 1, 2 and three representing colors white, green, blue and red respectively. As simulations goes forwards deferent shapes of tapestry are made.
Different scenarios can be modeled by changing the dimensions and number of central cells which are first initialized to one. This will lead to different patterns of tapestry e.g. 8 or 16 ones in the center, in the initial phase.

3. CELL-DEVS MODEL

The Cell-DEVS model will be based on three, two-dimensional square grids. The grid dimensions are at first 64x64. As mentioned before the first plane (front plane) is responsible for changes in Von Neumann neighborhood with the rules that says if the cell has odd number of true neighbors changes its value to zero, and if it has even number of true neighbors in the same plane changes its value to one.

The second plane (rear plane) always copies the old values of on step behind the first plane. The third plane which is the tapestry plane makes the final shape of tapestry by taking four values corresponding to four states that might happen in two corresponding cells of first and second planes. This way, four colors are shown on the tapestry. Figure 4 shows neighbor cell definition for the model.

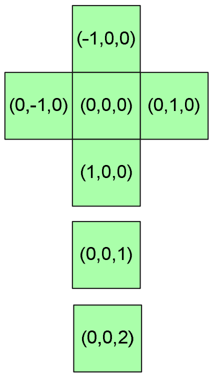


Figure 4- neighborhood Definition

The formal specification of a Cell-DEVS model for the following neighbor cells is given by:

M=<I, X, Y, Xlist, Ylist, η, N, {n1,n2,n3}, C, B, Z, select> Where:

Xlist=Φ

Ylist=Φ

η=7

I=<PX,Py>,with PX={Φ},Py={Φ};

N={(-1,0,0),(0,-1,0),(0,0,0),(0,1,0),(1,0,0),(0,0,1),(0,0,2)}

X=Y= {0, 1, 2, 3};

{n1, n2, n3}={64, 64, 3}

B= {Cij / (i=1v i=64) □ (j=1v i=64)} nowrapped;

C={Cijz / iε[1,64], jε[1,64], z ε[1,3]}

Z:

$$P_{ijz}\,Y_1 \rightarrow P_{i,j-1z}\,X_1 \qquad P_{i,j+1z}\,Y_1 \rightarrow P_{ijz}\,X_1$$
$$P_{ijz}\,Y_2 \rightarrow P_{i+1,jz}\,X_2 \qquad P_{i-1,jz}\,Y_2 \rightarrow P_{ijz}\,X_2$$
$$P_{ijz}\,Y_3 \rightarrow P_{i,j+1z}\,X_3 \qquad P_{i,j-1z}\,Y_3 \rightarrow P_{ijz}\,X_3$$
$$P_{ijz}\,Y_4 \rightarrow P_{i-1,jz}\,X_4 \qquad P_{i+1,jz}\,Y_4 \rightarrow P_{ijz}\,X_4$$
$$P_{ijz}\,Y_5 \rightarrow P_{ijz}\,X_5 \qquad P_{ijz}\,Y_5 \rightarrow P_{ijz}\,X_5$$

Select ={(-1,0,0),(0,-1,0),(0,0,0),(0,1,0),(1,0,0),(0,0,1),(0,0,2)}

Important Values in third plane:

0 (white): a zero in a cell of the first plane and a zero in the corresponding cell of the second plane.

1 (green): a one in a cell of the first plane and a zero in the corresponding cell of the second plane.

2 (blue): a zero in a cell of the first plane and a one in the corresponding cell of the second plane.

3 (red): a one in a cell of the first plane and a one in the corresponding cell of the second plane.

This model has been implemented in CD++ with dimensions of 3 planes each 64x64 cells. In two cases borders have been set to nowrapped and one case to wrapped, with default delay time of 100 milliseconds. Plane 1 is considered as front plane and four central cells (31, 31), (31, 32), (32, 31) and (32, 32) have been set to 1, others set to zero. In one case there are 16 ones in the center. Plane 0 is considered as tapestry plane and plane 2 is considered as rear plane.

Rules with CD++ code for each plane are as follows:

```
%[rearplane]
rule : {(0,0,-1)} 100 { cellpos(2)=2 }
%[tapestryplane]
rule  :  0  100  {  cellpos(2)=  0  and  (0,0,1)=0  and
(0,0,2)=0 } %white
rule  :  1  100  {  cellpos(2)=0  and  (0,0,1)=1  and
(0,0,2)=0 } %Green
rule  :  2  100  {  cellpos(2)=0  and  (0,0,1)=0  and
(0,0,2)=1 } %Blue
rule  :  3  100  {  cellpos(2)=0  and  (0,0,1)=1  and
(0,0,2)=1 } %Red
%[frontplane]
rule  :  1  100  {  cellpos(2)=1  and  even(truecount-
(0,0,0)-(0,0,1)-(0,0,-1))}
rule  :  0  100  {  cellpos(2)=1  and  odd(truecount-
(0,0,0)-(0,0,1)-(0,0,-1))}
```

4. SIMULATION RESULTS

In this project a DEVS model has been defined for the CD++ Modeler is a 2D visualization tool in CD++ toolkit that plots each plane separately. The color of each value or a range of values of cells can be defined using a pallet file. Each step of the simulation can be viewed using this application. After running different scenarios of the simulation (that takes almost four hours for the largest one), as time advances many nice tapestry designs are made. With larger dimensions we could have nicer and more detailed shapes but because of memory and processing time limitations we couldn't get results more than 64x64.

Figure 5 shows order of steps in which some models of tapestry with 64x64 dimensions and four ones in the center with no wrapped borders are made:

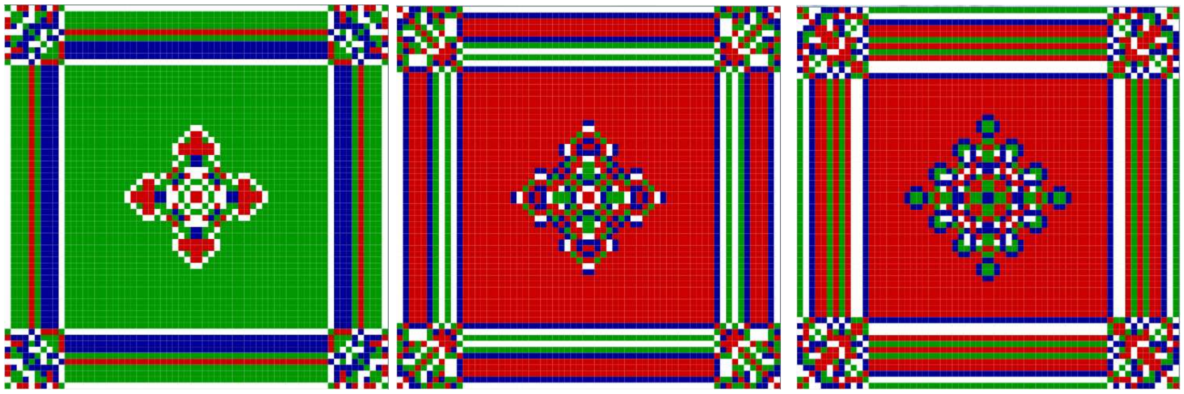


Figure 5- three Steps of tapestry with 64x64 dimensions and 4 ones in the center;

Figure 6 shows order of steps in which some models of tapestry with 64x64 dimensions and 16 ones in the center with no wrapped borders are made:

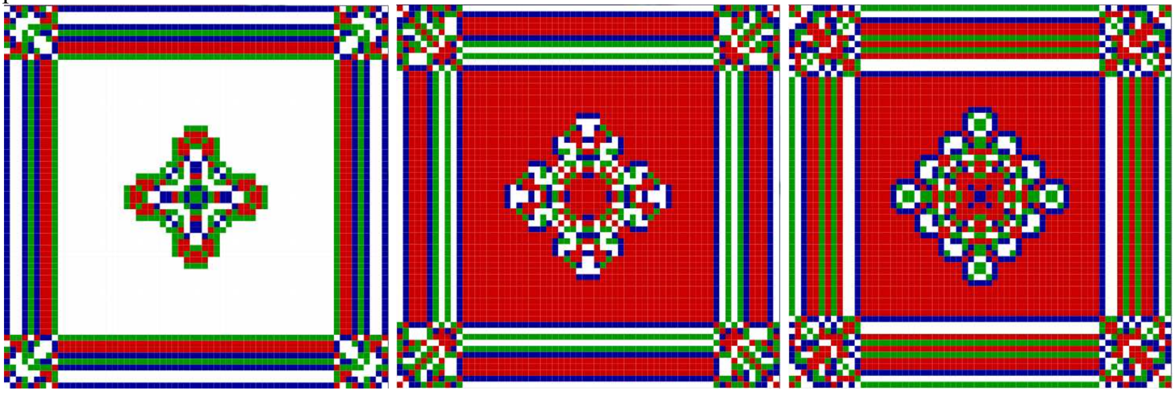


Figure 6- three Steps of tapestry with 64x64 dimensions and 16 ones in the center;

5. CONCLUSION

In this paper we simulated a kind of Persian tapestry using cellular automata. Each knot in the tapestry is considered a cell in the respective cellular automata. The simulation works with three planes of cells in which the first plane has an initial pattern and with a specified and simple rule changes the states of cells. The second plane follows the states of first plane after a constant delay time. The third plane which shows the shape of the tapestry uses some rules to obtain the color of each cell from respective cells of first and second plane.

CD++ simulator toolkit has been used to simulate the tapestry. Simulation has been run for dimension of 64x64 cells with initial pattern of 4 and 16 ones in the middle of the first plane. CD++ Modeler toolkit has been used to render the results in a 2D space. Because of memory and execution time limitations results with larger sizes could not be obtained as a real tapestry dimension are 500x500 or 1000x1000.

References

[1] ZEIGLER, B.; KIM, T.; PRAEHOFER, H. "Theory of Modeling and Simulation". Academic Press. 2000.

[2] Wolfram, S. "Theory and applications of cellular automata". Vol. 1. Advances Series on Complex Systems. World Scientific. Singapore. 1986.

[3] The Use of Cellular Automata in the Classroom, Lilly, H.A.; Supercomputing, 1995. Proceedings of the IEEE/ACM SC95 Conference.

[4] WAINER, G.; GIAMBIASI, N. "Timed Cell-DEVS: modeling and simulation of cell spaces ". In "Discrete Event Modeling & Simulation: Enabling Future Technologies", Springer-Verlag. 2001.

[5] WAINER G., "CD++: a toolkit to define discrete-event models". Software, Practice and Experience. Vol. 32, No.3. pp. 1261-1306. November 2002.

[6] Wainer, G. et al. "CD++ A tool for DEVS and Cell-DEVS Modeling and Simulation. User's Guide". Draft. August 2004.

[7] Wilson Venhola. "Visualization of Complex Simulations: A DEVS Visualization Tool". B.A Report. Department of Systems and Computer Engineering – Carleton University.