# AN EFFICIENT CELLULAR DEVS MODELING ENVIRONMENT

FAHAD A. SHIGINAH[1,2], BERNARD P. ZEIGLER[1]

[1] Arizona Center for Integrative Modeling and Simulation
Electrical and Computer Engineering Department
University of Arizona, USA

[2] Department of Electrical and Computer Engineering
Sultan Qaboos University, Oman

## ABSTRACT

Discrete EVent System specification (DEVS) supports object oriented modeling and simulation environments. DEVS is a sound mathematical formalism for representing complex dynamic systems such as cellular space models. This paper presents a new cellular DEVS environment that employs multi-layer modeling technique in order to improve model development and testing processes. In addition, it assures simulation efficiency of the developed models by replacing inter-cell communication messages by direct state access. Using the DEVS formal framework, the new development tool was designed to have automated specification and code generation. Applications to landslide models demonstrated the speedup and scalability attained by the new tool.

**Keywords:** Simulation Software, AI, Modeling, DEVS, Cellular Space.

## INTRODUCTION

The cellular space modeling approach divides space into discrete intelligent cells that perform local computations based on their own as well as their neighbors' states. In conventional DEVS implementation of cellular models (e.g. [1-5]), a cell is implemented as a DEVS modular atomic or coupled model. When detailed modeling of spatial dynamics is required, a large number of cells is typically employed. This results in a large number of atomic models that communicate through message passing to carry out the global simulation. Therefore, implementing large scale cellular spaces with highly active cells in DEVS will face the burden of huge numbers of inter-cell messages and hence a performance reduction. Many techniques were introduced to resolve this issue and to gain speedup. Examples of such work can be found in [1, 2, 6] where the cellular DEVS simulation engine was improved to handle messages and cell activity scanning in more efficient manner. On the other hand, the quantized DEVS approach [7-9] shows that quantization helps in improving the performance of DEVS simulations by reducing the number of state transitions as well as the number of messages while introducing acceptable errors.

A few related works cover the area of converting coupled DEVS models into atomic models by allowing the conversion during the compilation process. Reference [10] converted classical rather than parallel DEVS models and did not target the cellular space in particular but obtained speedup for small models. Reference [9] converted large hierarchical Dymola models into atomic ones with the conclusion that there is no advantage since the overhead of handling big model is much greater than the reduction in message overhead. In contrast, our initial work in [11] proved the significance of the approach for large cellular DEVS models. In this paper, we confirm this significance and show how to employ the conversion within the specification and development processes.

In this paper, we introduce a new cellular DEVS model development environment that employs the approach of converting coupled models into atomic ones through the closure under coupling property of parallel DEVS in order to ensure simulation efficiency. The objective of this tool is to minimize the coding efforts required by users to develop cellular DEVS models over DEVSJAVA. A multilayer approach is employed in the new tool with help of some automated processes namely: automated specification transformation, automated code generation, and automated testing and model verification. This environment made it possible for the user, for the first time, to develop cellular DEVS models using GUI without writing the full DEVSJAVA code. It was made as generic as possible to eliminate the need for the user to modify the generated code.

## PARALLEL DEVS FORMALISM

Discrete EVent System Specification (DEVS) [7] supports object orientation over modeling environments. Its theory provides a mathematical formalism for representing dynamic systems. The DEVS formalism was revised in [12] to reduce sequential processing and enable full parallel executions. The resulting parallel DEVS has the basic atomic model defined as:

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

Where $X$ is a set of input values, $S$ is a set of states, $Y$ is a set of output values, $\delta_{int}$ is the *internal transition* function, $\delta_{ext}$ is the *external transition* function, $\delta_{con}$ is the *confluent transition* function, $\lambda$ is the output function, and $ta$ is the time advance function.

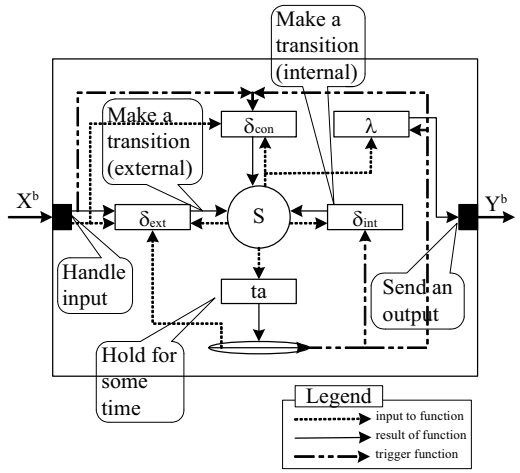


Figure 1. Atomic model structure in Parallel DEVS

An atomic model $M$ in parallel DEVS remains in a state $s \in S$ for an amount of time, *ta(s)*, if no external event occurs. When that time advance expires, i.e., when the elapsed time, $e = ta(s)$, the system outputs the values, $Y^b = \lambda(s)$, just before it changes to state $\delta_{int}(s)$. When an external event x in $X^b$ occurs before this expiration time, i.e., at $e < ta(s)$, the system changes to state $\delta_{ext}(s,e,x)$. However, if in case internal and external transitions collide, $\delta_{con}$ is employed to resolve the conflict and determine the next state. In all cases, the model then goes to some new state s′ with some new resting time, *ta(s')* and the same story continues[7].

Note that input or output values $X^b$ and $Y^b$ are bags of elements. This means that one or more elements can appear on a port at the same time. This capability comes from the parallel implementation of DEVS which allow components to send to the ports simultaneously. These basic components may be coupled in DEVS to form a multi-component model which is defined by the following structure:

$$CM = < X, Y, D, \{M_i\}, \{I_i\}, \{Z_{i,j}\} >.$$

Where $X$ is the set of input values, $Y$ is the set of output values, $D$ is the set of atomic models, $I_i$ is the influencees of $i$, *and* $Z_{i,j}$ is the $i$ to $j$ output translation function (coupling).

## CELLULAR DEVS MODELS

Parallel DEVS is an object oriented environment in which models can be defined as instances of atomic or coupled model classes. The cellular space is a hierarchical coupled parallel DEVS model (*CM*) that consists of a number of cells which are created as instances of atomic or coupled model. The cells should be arranged as a grid and hence each cell has a well defined address in the space. According to this addressing scheme, the coupling relations $\{Z_{i,j}\}$ will connect each cell to its neighboring cells defined by the neighboring rules followed by the model.

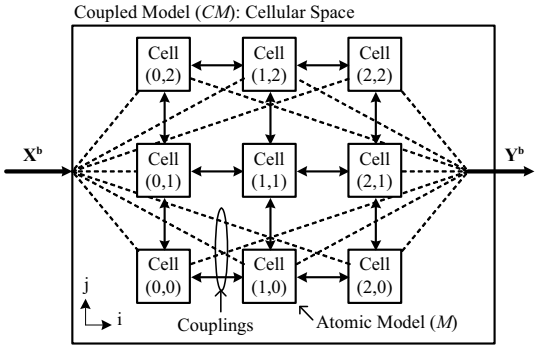


Figure 2. An example of 2-D cellular space in DEVS.

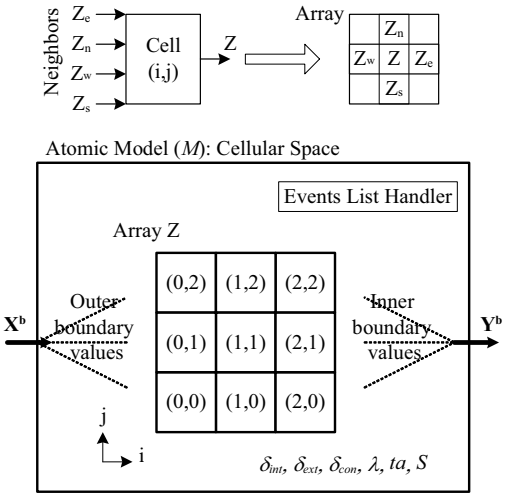


Figure 3. Converting a cellular space into an atomic model.

## INTER-CELL MESSAGES ELIMINATION

Closure under coupling, in parallel DEVS, implies that a coupled model (*CM*) can be treated as an atomic model (*M*) which is equivalent to *CM* (see [12] for detailed proof). This property supports the feasibility of implementing an approach in which a coupled model of a group of cells will be converted entirely into its equivalent atomic model. The resultant atomic model contains non-modular cells in which each cell can access the states of it neighboring cells and this eliminates the need for sending messages. The conversion process involves adding a discrete event list handler inside the atomic model to keep track of

activities among all cells that belong to the atomic model.

## Event List Handler

Discrete event simulation proceeds through executing the imminent events in a list of scheduled events. Executing an event might result in scheduling other events and the process goes on until the list of events is empty or the simulator reaches a predefined stopping point. In DEVS coupled models, the coordinator is responsible in processing and storing the event list. As a result of converting a coupled model into an atomic model, the event list processing task will be encapsulated inside the new atomic model's functions. This was achieved by introducing an *EVENTS* list that holds scheduled events such that the model processes the cells extracted from the event list. Consequently, the model spends a significant amount of time in processing the cell list for scheduling future events, extracting current events, and scanning cell lists. Therefore, implementing those cells will have a significant impact on the simulation execution and hence, the targeted speedup. Since the main goal in this work is to develop large scale high performance cell spaces, the list handler should be efficiently implemented that should process a large number of events very quickly to avoid adding latency to the overall cell space atomic model. The standard operations in event list processing include: adding a new record, and extracting the record with the minimum time stamp which entails finding that minimum time and then deleting that record. The first additional operation needed in the event list for our approach is the arbitrary removing of cells from the event list. The second one is advancing the time stamps in all records since the stamps are relative to the local current time of the model. In addition, the method used to extract the least time stamp record was designed to have more operations than just extracting a record. It should search for the minimum time stamp in the list and gather all the cells with that minimum time stamp so that it retrieves it as one cell list with the possibility of not removing cells records permanently.

## MULTI-LAYER APPROACH

In this work, two new layers of specifications were suggested on top of the DEVS specification layer which is represented in the implementation by DEVSJAVA. The first layer is the user specification layer that accepts the model specifications from the user through the GUI. With the help of code generator, this specification can be automatically transformed into the middle layer in which models are put in an efficient cellular DEVS format which is finally generated as a full DEVSJAVA code. Therefore, the new development environment is spans the two upper specification layers and produces models that must satisfy the third layer of specification. Figure 4 shows the design structure of the new cellular DEVS modeling environment which takes model specifications via the Graphical User Interface (GUI) and generates the code that can be run in a DEVSJAVA environment with an efficient cellular DEVS specifications.

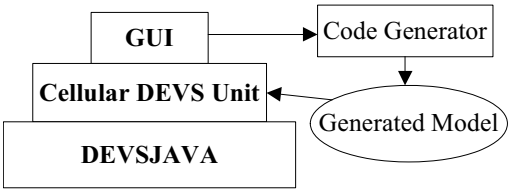


Figure 4. Structure of the new Cellular DEVS environment.

The main design units in this environment are: GUI, code generator, and the cellular DEVS Specifications unit. Each of these units contains its own classes and has different objectives. Generally speaking, the GUI unit is a DEVS independent unit that gathers information about the model from the user and passes them into the code generator. The code generator unit gathers that information and produces the model's code guided by the format of the new cellular DEVS specification. The generated code extends the standard classes given by the cellular DEVS specification. Those, in turn, extend the standard modeling classes in the DEVSJAVA modeling layer. The specifications unit hides most of the implementation details and includes large methods and classes that were hidden from the user to ease the user task in the model development process.

## The GUI

The main class in our environment is the Graphical User Interface (GUI), where the user inputs the cellular model specifications, generates models, and reloads previously generated models for modifications. It was implemented using JFrame containing two tabs: the model specifications tab and the cell's local transition function tab. The model specification tab is the main tab which allows the user to write the model name and its package name, select the space type either cell space or block space, select the neighboring rule to use for the model, select the input data file where the model should extract its initial data from, and fill the ports/variables mapping table. After entering all model specifications and functions, the user presses the generate button which will prompt him to enter the cell space size and the number of blocks in case of block space type. Then, all model parameters and functions will be sent to the code generator which will generate the code for that model. In addition, the GUI will generate a model property file that stores all user entries which can be used later to reload them in case of further modifications.

## TESTING AND VERIFICATION

The developed modeling environment was tested using standard software testing methodologies such as black box, white box, and gray box test case generation methods [13]. In addition, the dynamics of the environment was validated through testing the

generated models by applying software as well as model verification techniques. The first one involves Java reflection tests that checks whether the environment generates correct model code while the second tests whether the generated model is what the user intended to develop. The model verification involved adopting automated DEVS verification tools developed by [14] and improving them to correctly handle zero time transition cycles in cell spaces. Three test models were selected for testing and verification. The first one is the two dimensional game of life which is know for its popularity and simplicity. The second one is the simple 2-D wall following robot which is popular in artificial intelligence texts. The last one is the basic finite difference numerical solution of a one dimensional heat equation.

## LANDSLIDE APPLICATION

Landslides are among the major natural hazards that occur frequently on earth. The need of employing artificial intelligence techniques, such as cellular space modeling, in predicting their occurrences and behavior becomes a must to save lives and avoid major damages. The new developed environment was put into the challenge of correctly developing and running complex landslide models. The models implemented in this work are hexagonal cell space models in which each cell is represented as a hexagon that is surrounded with six other hexagons. The two developed landslide models are based on the models derived in [15] and [16]. The first one is a pure cellular automata model since the time plays no role in all equations while the other one is a discrete time cellular automata model that solves partial differential equations.

## EXPERIMENTAL RESULTS

The main purpose of the experiments is to show insight on the speed up claimed to be achieved using the new development environment. The landslide models were run over a 3.0 GHz Pentium 4 machine with 1GB of RAM. All the runs were done for 32×32 cell space where the data was an approximated portion taken from Fig.8 in [15]. Table 1 lists the time (in seconds) taken to execute 100 iterations of the landslide simulations for the new approach as well as the conventional one.

| | New approach | Conventional |
|---|---|---|
| Model-1 | 4 | 18 |
| Model-2 | 15 | 110 |

Table 1: Execution times of landslide simulation runs.

The new approach followed in the new development environment shows significant speedups compared to the conventional cellular DEVS models. That new implementation represents the elimination of all inter-cell messages inside the cell space. The more inter-cell messages present in the model, more is the speedup that can be achieved. The achieved speedups in the two landslide models are 4.5 and 7.3 respectively.

## CONCLUSION

The cell space environment was designed to accelerate cellular model development and make the coding level transparent to the user through employing a multi-layer approach. The environment can support wide varieties of modeling requirements. It was tested using some of the standard software testing strategies and found to perform correctly according to the test cases. In addition to the software testing, the generated models can be tested using the simulation-based DEVS verification approaches. A modification to the current automated cellular DEVS verification approach was implemented to correctly test models that account for zero time transitions. All automated testing classes are made available for the user to test and verify the developed models.

The message elimination approach that decomposes cell spaces entirely into atomic models enhances the simulation speed significantly. The landslide models developed in this work tried to push the new environment to the limits. These complex models required more modifications to be added to the generated code. It was shown that all of the requirements can be easily added to the environment in future work thus limiting the need to modify the generated models. In addition to the models presented in this paper, the new environment can be efficiently used to develop wide range of applications such as numerical solutions of differential equations, discrete time cellular models, natural dynamic models, and cellular space based artificial intelligence applications.

## REFERENCES

[1] G. Wainer and N. Giambiasi, "Application of the Cell-DEVS Paradigm for Cell Spaces Modelling and Simulation," *Simulation,* vol. 76, pp. 22-39, 2001.

[2] X. Hu and B. P. Zeigler, "A high performance simulation engine for large-scale cellular DEVS models," in *High Performance Computing Symposium (HPC'04), Advanced Simulation Technologies Conference,* 2004.

[3] G. A. Wainer and N. Giambiasi, "N-dimensional Cell-DEVS Models," *Discrete Event Dynamic Systems,* vol. 12, pp. 135-157, 2002.

[4] G. Wainer and N. Giambiasi, "Timed cell-DEVS: modeling and simulation of cell spaces," pp. 187-214, 2001.

[5] A. Muzy, E. Innocenti, A. Aiello, J. -. Santucci and G. Wainer, "Cell-DEVS quantization techniques in a fire spreading application," in *WSC '02: Proceedings of the 2002 Winter Simulation Conference (WSC'02) - Volume 1,* 2002, pp. 542-549.

[6] Muzy and J. J. Nutaro, "Algorithms for efficient implementations of the DEVS & DSDEVS abstract simulators," in *1st Open International Conference on Modeling & Simulation (OICMS),* 2005.

[7] P. Zeigler, T. G. Kim and H. Praehofer, *Theory of Modeling and Simulation.* San Diego, CA, USA: Academic Press, Inc, 2000.

[8] E. Kofman and S. Junco, "Quantized-state systems: a DEVS Approach for continuous system simulation," *Trans. Soc. Comput. Simul. Int.,* vol. 18, pp. 123-132, 2001.

[9] T. Beltrame, "Design and Development of a Dymola/Modelica Library for Discrete Event-oriented Systems Using DEVS Methodology," 2006.

[10] W. B. Lee and T. G. Kim, "Simulation speedup for DEVS models by composition-based compilation," in *Proceedings of Summer Computer Simulation Conference,* 2003, pp. 395-400.

[11] F. A. Shiginah and B. P. Zeigler, "Transforming DEVS to non-modular form for faster cellular space simulation," in *Proceedings of 2006 DEVS Symposium,* 2006, pp. 86-91.

[12] A. C. Chow and B. P. Zeigler, "Parallel DEVS: A parallel, hierarchical, modular, modeling formalism," in *WSC '94: Proceedings of the 26th Conference on Winter Simulation,* 1994, pp. 716-722.

[13] L. Copeland, *A Practitioner's Guide to Software Test Design.* Norwood, MA, USA: Artech House, Inc, 2003.

[14] G. Wainer, L. Morihama and V. Passuello, "Automatic verification of DEVS models," in Proceedings of the 2002 Spring Simulation Interoperability Workshop, 2002.

[15] D'Ambrosio, S. D. Gregorio and G. Iovine, "Simulating debris flows through a hexagonal cellular automata model: SCIDDICA S_3-hex," Natural Hazards and Earth System Sciences, vol. 3, pp. 545-559, 2003.

[16] Segre and C. Deangeli, "Cellular automaton for realistic modelling of landslides," *Nonlinear Processes in Geophysics,* vol. 2, pp. 1-15, 1995.